Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte de la thèse : SOA et sécurité

Notre société et le système économique sur lequel elle repose sont actuellement confrontés à des changements dramatiques dans leurs infrastructures et processus. Faire face à cette inexorable expansion des infrastructures de l'information et de la communication, à leur omniprésence et au rythme insoutenable du développement sociétal, exige des professionnels comme des particuliers une réactivité quasi-instantanée à des besoins de plus en plus versatiles et nouveaux. A titre d'exemple une entreprise a besoin d'ajuster sa chaîne d'approvisionnement instantanément si son fournisseur privilégié s'avère incapable de livrer voire même de faire appel à une communauté d'intermédiaires pour trouver un remplacement en quelques minutes, car elle ne peut pas se permettre d'imposer des temps d'attente à ses clients. D'une manière similaire, le particulier peut être amené à utiliser des services administratifs ou gouvernementaux pour déclarer et payer ses impôts ou immatriculer un véhicule, à n'importe quel moment, depuis des environnements différents et en utilisant des équipements de plus en plus sophistiqués et hétérogènes.

Ces changements donnent lieu à un important changement de paradigme dans la façon dont les systèmes et applications des TIC sont conçus, mis en œuvre et déployés. Pour adresser des exigences et des besoins versatiles des entreprises, au sein d'une fédération par exemple, les composants sont déployés en tant que *services* accessibles sur le réseau (internet par exemple) et composés à la demande d'une manière flexible.

La *composabilité* des services est l'un des principes de base de l'Architecture Orientée Service ou SOA (acronyme de Service Oriented Architecture). Elle en exploite deux autres principes : la ré-utilisabilité et l'inter-opérabilité. Dans ce sens chaque service peut s'appuyer sur l'existence et la disponibilité d'autres services (éventuellement découverts dynamiquement) pour effectuer son calcul. Ceci inclut l'adaptation dynamique et la combinaison explicite des politiques qui leur sont associées et qui déterminent les actions exécutées et les messages échangés par ces services. Par

exemple, un service octroyant à un partenaire, l'accès à une ressource disponible chez un tiers peut utiliser un service d'authentification ayant la confiance des deux pour vérifier l'identité du premier, et peut s'appuyer sur des services d'autorisation dans les deux extrémités qui combinent leurs politiques afin de décider si l'accès peut-être accordé.

Ce changement de paradigme vers les *architectures orientées services* permet une certaine *intelligence ambiante*, où les ordinateurs et les réseaux sont intégrés dans l'environnement quotidien et offrent des interfaces homme-machine facile à utiliser à une variété de services et d'applications. Ceci permet aux personnes d'accéder aux services où qu'elles soient et quand elles le désirent, et de la manière la plus naturelle pour elles. Ceci inclut des interactions avec le monde physique permettant la collecte de quantités substantielles de données personnelles, qui exigent de nouvelles solutions qui intègrent des garanties de confiance, de sécurité et de confidentialité.

L'orientation services et l'intelligence ambiante nécessitent de reconsidérer la notion de confiance, les exigences de sécurité, ainsi que des mécanismes utilisés pour assurer leur respect. Dans un tel contexte, les ressources sont de plus en plus exposées dans un un environnement de moins en moins contrôlé. En effet il n'y a plus de propriété centrale et encore moins de contrôle applicatif, les prestataires de services indépendants étant peu enclins à divulguer leurs détails d'implémentation dans le but de protéger leurs propriétés intellectuelles. Ces mêmes prestataires doivent d'un autre côté respecter leurs politiques de sécurité locales tout en les comparant à celles des propriétaires d'applications, débouchant sur la négociation des politiques et sur une évaluation de la confiance. Dans cette optique faire respecter les propriétés de sécurité désirées relève de la responsabilité de tous les prestataires de services. En outre les fonctionnalités de sécurité sont habituellement assurés par des services dédiés qui sont découplés des applications proprement dites. Cela soulève le problème de leur composition appropriée : en fonction de leurs fonctionnalités et du contexte donné, ces services peuvent être l'objet d'interférence nouvelles et subtiles quand ils sont composés.

1.1.1 Méthodes formelles, validation de propriétés de sécurité

Les systèmes de communication investissent des domaines de plus en plus variés tels que la téléphonie mobile, la télé à la demande ou de manière plus générale le commerce électronique, donnant lieu à de nouvelles préoccupations d'ordre sécuritaire. Dans un schéma de vente hors ligne, un client peut se rendre physiquement vers un magasin spécialisé dans les livres de science-fiction. Il choisit les livres qui l'intéressent avant de régler ses achats, par exemple à l'aide de sa carte bancaire. Le client insère alors sa carte dans le TPE (Terminal de Paiement Électronique) et saisit son code secret. Le TPE contacte alors le site de la banque du client sur une ligne privée (accessible uniquement à la banque du client et au magasin) et lui demande d'ordonner le paiement correspondant. Au vu de la confiance implicite que le client a en sa banque, il reste dans ce cas le seul responsable de la protection des données sensitives de sa carte bancaire, à savoir son numéro, sa date d'expiration et plus particulièrement son code secret qu'il doit, par exemple, protéger contre les regards indiscrets.

De nos jours, une telle vente peut se faire sur des réseaux ouverts au grand public tel que l'internet. Le magasin électronique propose alors une interface pour rechercher des livres (un formulaire de recherche dans une page web). Quand le client a fait son choix, il peut alors saisir les caractéristiques de sa carte dans un autre formulaire, afin d'ordonner le paiement de sa commande. Contrairement au cas classique, le magasin électronique peut communiquer les informations de paiement à la banque du client, via internet. Ceci pose le problème suivant : le client peut-il avoir la certitude que le code secret de sa carte bancaire ne puisse pas être intercepté par une tierce entité présente sur ce réseau public ? Cette propriété s'appelle *la confidentialité*

(ou le secret).

Le client peut aussi exiger d'autres propriétés pour le système de vente en ligne qu'il utilise. Par exemple, il voudrait des garanties sur l'identité du magasin auquel il a affaire. On appelle cette propriété *l'authentification*. Dans le cas d'une vente hors-ligne cette propriété est à la charge du client lui-même, qui peut s'en assurer en se rendant physiquement à l'adresse du bon magasin. Ceci n'est pas toujours évident quand il s'agit d'une vente en ligne où le client accède à l'interface du magasin en renseignant l'adresse d'une page web. D'autres propriétés qui pourrait intéresser le client sont *la non-répudiation* de son paiement qui garantit que le magasin ne peut pas répudier ou annuler une vente pour laquelle un paiement a déjà été correctement effectué ou encore *l'intégrité* des données échangées qui exprime par exemple le fait que le montant du paiement autorisé par le client ne soit pas modifié.

Ainsi des propriétés décrivant la sûreté du processus de vente en ligne peuvent être considérées. Pour assurer ces propriétés un ensemble de techniques, dont la cryptographie, sont utilisées. Le cryptographie définit des transformations sur les messages échangés ou visibles par une communauté d'intervenants, dans le but d'en rendre le contenu accessible uniquement à un sous-ensemble de cette communauté. Ces transformations son décrites par des algorithmes qui utilisent des clefs de chiffrement ou de déchiffrement pour respectivement protéger ou accéder au contenu de données sensitives. On distingue deux schémas pour le chiffrement (et le déchiffrement) de données : le symétrique et l'asymétrique. Dans le le schéma symétrique les clefs de chiffrement et de déchiffrement sont les mêmes, ou plus précisément elles sont facilement déductibles l'une de l'autre. Ce schéma permet en l'occurrence, l'échange de données d'une manière confidentielle entre deux pairs qui partagent au préalable une clef de chiffrement symétrique et ce en chiffrant toutes leurs communications à l'aide de cette clef. Dans le schéma asymétrique, ces clefs sont supposées être différentes et plus encore, très difficilement déductibles l'une de l'autre. En général, chaque intervenant dispose d'une telle paire de clefs, une qui sert pour le chiffrement (appelée sa clef publique) et une qui sert pour le déchiffrement (appelée sa clef privée). Ce schéma permet ainsi de garantir la confidentialité des échanges en partant du principe que seul de détenteur d'une certaine clef privée peut déchiffrer des données chiffrées à l'aide de la clef publique correspondante. L'intégration de ces techniques de protection aux procédés habituels d'échange de l'information donne lieu à des *protocoles cryptographiques*. D'autres techniques comme *la signature électronique* ou *le hachage* peuvent adresser d'autres besoins de sécurité, comme par exemple l'intégrité d'une donnée échangée, l'authentification d'un intervenant où l'identification d'une certaine donnée sans en divulguer la valeur.

Néanmoins le simple recours à ces techniques ne constitue pas pour autant une garantie pour les propriétés de sécurité qu'elles adressent. Ceci est du à certaines vulnérabilités des schémas cryptographiques utilisés eux-mêmes, mais aussi à la complexité des entrelacements des activités de communication dans les protocoles cryptographiques. Ceci motive l'existence d'un domaine de recherche, *la vérification des protocoles cryptographiques*, où les méthodes formelles sont d'un grand apport. La vérification des protocoles cryptographiques a pour but de vérifier si les communications spécifiées par un protocole cryptographique satisfont des propriétés de sécurité données, et ce en présence d'une entité malveillante appelé l'intrus capable d'intercepter et de modifier les messages échangés par les intervenants réguliers. Principalement, deux approches existent : l'approche computationnelle et l'approche symbolique. La première approche considère les messages échangés comme des séquences de bits qu'un intrus assimilable à une machine de Turing probabiliste peut manipuler. Cette approche permet notamment de prendre en compte les faiblesses directement reliées aux algorithmes cryptographiques mais elle reste difficilement automatisable. L'approche symbolique quant à elle considère les messages échangés comme des termes dans une certaine algèbre et représente l'intrus par l'ensemble des transformations qu'il est en mesure

d'appliquer sur les messages, par exemple, le déchiffrement d'un texte chiffré à l'aide de la clef correspondante. Contrairement à l'approche computationelle, l'approche symbolique considère les procédés cryptographiques comme des boites noires et néglige leurs éventuelles vulnérabilités propres.

Nous nous concentrons dans cette thèse sur l'approche symbolique. Nous donnons dans la suite un exemple de problème de sécurité pour un protocole cryptographique.

Le protocole d'authentification mutuelle de Needham et Schroeder

Nous considérons la variante à clef publique du protocole présentée par Needham et Schroeder dans [48]. L'objectif de ce protocole est l'authentification de deux agents A et B en utilisant un serveur de clefs publiques S ayant la confiance des deux. La spécification du protocole est décrite ci-dessous à l'aide de la notation semi-formelle Alice et Bob, très utilisée dans le domaine des protocoles cryptographiques.

```
A knows A, B, Na, KPa, KSa, KPs
B knows A, B, Nb, KPb, KSb, KPs
S knows A, B, KPs, KSs, KPa, KPb

1. A -> S : A,B
2. S -> A : {KPb, B}KSs
3. A -> B : {Na, A}KPb
4. B -> S : B,A
5. S -> B : {KPa, A}KSs
6. B -> A : {Na, Nb}KPa
7. A -> B : {Nb}KPb
```

Dans cette spécification A, B et S sont les noms des agents jouant le protocole. Les connaissances initiales de chaque agent sont décrites à l'aide du mot clef *knows*. Par exemple, A connait initialement sa propre identité, celle de B, sa paire de clefs publique et privée KPa et KSa, ainsi que la clef publique KPs du serveur S. D'une manière générale, pour tout $x \in \{a, b, c\}$ KPx (resp. KSx) est la clef de chiffrement ou clef publique (resp. de déchiffrement ou clef privée) de l'agent X. Après la déclaration des connaissances initiales, les différentes communications sont ensuite décrites. Ainsi $X- > Y : M$, représente le fait que l'agent X envoie le message M à l'agent Y. Par ailleurs, Na et Nb sont des valeurs aléatoires (ou nonces) générées respectivement par A et B. Les opérations possibles sur les messages sont décrites par des symboles particuliers. Ici $M1, M2$ représente la concaténation des deux messages $M1$ et $M2$, et $\{M\}_K$ représente le chiffré asymétrique du message M à l'aide de la clef K.

Une description textuelle du déroulement du protocole est décrite ci-dessous :

Étape 1. A signale au serveur sa volonté de commencer une session du protocole avec B en lui envoyant la concaténation de son identité avec celle de B.

Étape 2. S lui renvoie le certificat électronique de B qui relie l'identité de ce dernier à sa propre clef publique. Dans notre cas, ce lien est matérialisé par le chiffrement à l'aide de la clef privée de S, qui est à priori le seul agent à la détenir. Ainsi, en recevant le message de l'étape 2., A est en mesure de vérifier que ce message vient de S en le déchiffrant à l'aide de la clef publique KPs de S. Comme A a confiance en S, il peut en conclure que la première composante du message déchiffré est la clef publique de B.

Étape 3. A génère alors une valeur fraîche Na, la concatène à son identité avant d'envoyer le tout chiffré par la clef publique de B.

Étape 4. et 5. B récupère la clef publique de A de S.

Étape 6. B génère ensuite une valeur fraîche Nb la concatène au nonce Na précédemment reçu de A, chiffre le tout avec la clef publique de A avant de renvoyer le message obtenu à A.

Étape 7. Enfin A renvoie la valeur de Nb chiffrée par la clef publique de B à ce dernier.

Étant donnés la confiance de A et B en S et l'utilisation de la cryptographie dans leurs communications nous pourrions avancer les hypothèses suivantes :

- A l'étape 6., quand A reçoit son nonce Na il est convaincu qu'il vient d'authentifier B sur la valeur Nb.
- A l'étape 7., B a aussi la même la même conviction à propos de A et Na.

Les deux valeurs Na et Nb peuvent alors être utilisées par A et B pour générer une nouvelle clef symétrique, qui pourrait permettre d'échanger de nouvelles données d'une manière confidentielle.

Néanmoins, les hypothèses faites plus haut s'avèrent fausses. Nous présentons dans la section suivante un scénario qui le prouve.

Une faille de sécurité du protocole

Un intrus I peut usurper l'identité de A, après l'avoir incité à initier une session du protocole avec lui. Nous représentons ci-dessous les communications qui en découlent, en ignorant les échanges de messages avec le serveur de clés publiques S et en considérant uniquement les messages échangés entre les agents A et B, et l'intrus I. Par ailleurs, nous supposons que l'intrus, possède une paire de clés (KPi, KSi), et nous pouvons également supposer que les autres agents connaissent KPi.

```
i.3. A -> I : {Na,A}KPi
ii.3. I(A) -> B : {Na,A}KPb
ii.6. B -> I(A) : {Na,Nb}KPa
i.6. I -> A : {Na,Nb}KPa
i.7. A -> I : {Nb}KPi
ii.7. I(A) -> B : {Nb}KPb
```

L'étape $i.3$ correspond à l'étape 3. du protocole où A et I joue une première session régulière (i) du protocole. Par contre à l'étape $ii.3.$, l'intrus $I(A)$ se fait passer pour A auprès de B. A l'étape $ii.6$, B pense alors répondre à A en envoyant $\{Na, Nb\}KPa$, après avoir généré un nouveau nonce Nb. L'intrus intercepte ce message et comme il ne détient pas la clef privée de A, il ne peut pas le déchiffrer. Par contre il peut utiliser sa session régulière avec A pour y arriver. En effet il renvoie ce message à A à l'étape $i.6.$, et ce dernier pensant qu'il s'agit du dernier message de la session i le déchiffre et renvoie la valeur Nb à l'intrus après l'avoir chiffré par sa clef publique KPi. L'intrus peut alors déchiffrer et renvoyer Nb à B qui est alors persuadé à tort qu'il vient d'authentifier A pour la valeur Na.

Cette attaque a été trouvée à l'aide d'un outil de vérification automatique par Lowe [2]. Elle se base sur un intrus I ayant le pouvoir d'écouter ou d'intercepter avant de modifier les messages échangés entre A, B et S. Les modifications possibles sur les messages pour I sont celles décrites pas Dolev et Yao [37]. Cela signifie que I peut concaténer des messages ou décomposer une concaténation de messages, chiffrer des messages ou déchiffrer des messages à condition d'avoir

les bonnes clefs. En adaptant les messages I peut détourner l'utilisation du protocole, et sans avoir à casser les algorithmes de chiffrement utilisés, il peut invalider certaines propriétés de sécurité telle que l'authentification, dans l'exemple présenté. La simplicité de cette attaque qui fut découverte des années après la publication du protocole motive le recours à des outils de vérification automatisés et confirme l'insuffisance du simple recours aux techniques de cryptographie à assurer certaines propriétés de sécurité.

1.1.2 Composabilité, orchestration

Un exemple classique d'orchestration est celui de l'agence de voyages dématérialisée. Au départ, on dispose de services proposant respectivement : des vols disponibles à partir des dates allers-retours, des hôtels, des location de véhicules, des excursions etc. Ensuite on souhaite faire interagir ces services pour produire un service plus complexe, permettant de réserver et de régler en ligne en incluant tous les détails d'un séjour tels que les billets pour les vols, les frais d'hôtels, etc. Dans un niveau hiérarchique supérieur des comparateurs de prix peuvent utiliser les nouveaux services de réservation de voyages, pour proposer un meilleur service pour un client plus exigeant. Cette collaboration dont l'objectif est d'assurer un certain comportement global désiré, est d'autant plus critique que chaque service est aussi sujet à sa propre politique locale. Dans notre exemple, un service de réservation de voitures pourraient exiger une réservation exclusive à certains types de carte bancaires : ceci est une politique locale. Par ailleurs tous les services de l'orchestration intervenant dans une opération de paiement du client, sont tenues d'échanger son numéro de carte bancaire de manière confidentielle : ceci est une politique globale. Il est nécessaire pour la SOA de produire de manière automatique des services répondant à de nouveaux besoins spécifiques, soit planifiés d'une manière statique, soit à la demande d'une manière dynamique, tout en respectant les politiques locales de tous les services pré-existant et en garantissant une politique globale désirée : représentée dans notre exemple par les besoins et exigences du client. Une deuxième nécessité reste la validation automatique de la cohérence et de la sûreté de l'orchestration ainsi obtenue et incluant le nouveau service médiateur généré (dans notre exemple le site de réservation de voyages) ainsi que l'ensemble de services avec lesquels il interagit.

1.2 Travaux reliés

La composition automatisée de services web soumis à des politiques de sécurité est une tache complexe. Principalement, il existe deux approches pour composer les services : la chorégraphie et l'orchestration [57]. Dans l'approche chorégraphique il n'existe aucune entité centrale et chaque service Web est responsable de mettre en œuvre sa part du service composé, la communication se fait directement entre les les services. En revanche, lors d'une orchestration toutes les communications passent par une entité appelée le médiateur, une entité centrale qui expose le nouveau service proposant une fonctionnalité nouvelle, en réutilisant des services existants. L'interface d'un service Web est généralement décrite en utilisant des langages dédiés : par exemple selon le standard WSDL, un service web est défini par l'ensemble des opérations qui peuvent être indépendamment invoquées par l'utilisateur. En effet, un document WSDL définit les services comme un ensemble d'opérations, chaque opération est représentée par son message d'entrée et ses messages de sortie (pour une exécution normale ou erronée). Un service web tel que représenté par la description WSDL est une entité qui n'a pas d'état [34]. Généralement, cette information n'est pas suffisante pour utiliser correctement un service qui offre un fonctionnement complexe. L'utilisateur doit alors suivre un certain scénario pour interagir avec un service [17, 65]. Par

exemple, l'utilisateur peut d'abord être authentifié avant d'utiliser d'autres opérations sur le service web. Ce genre de scénarios est parfois appelé protocole comportemental de services web [18]. Par exemple, dans [18] les auteurs présentent une approche d'extraction automatique des protocoles comportementaux directement à partir des WSDL afin de les déployer. Un service est alors décrit comme un rôle de protocole, qui est modélisé en termes de strands [38] (une séquence de réception ou d'émission de messages), où les messages sont représentés par des termes du premier ordre. En plus de la liste des opérations (les modèles des messages d'entrée et de sortie), une description WSDL peut spécifier des liaisons sécurisées (comme HTTP sur SSL) et aussi une politique de sécurité à l'aide du standard WS-SecurityPolicy (WSSP). Par le biais de WSSP, un fournisseur de services peut attacher une politique de sécurité sur des parties spécifiques dans les messages des opérations : celles devant être signées électroniquement, chiffrées etc. Les services web sont publiés dans des annuaires UDDI [64] ou ebXML [50]. Ces spécifications offrent aux utilisateurs un moyen centralisé pour rechercher des fournisseurs de services par le biais d'un registre.

Le problème sur lequel on se penche dans cette thèse est un moyen automatique de composer des services web, tenant compte des politiques de sécurité qui s'appliquent à eux au niveau message. Nous résolvons le problème notamment dans le cas où le service médiateur ne se contente pas de transférer les messages entres services web existants et où il doit adapter les messages reçus pour construire ses messages à envoyer. En fonctions de ses connaissances initiales (comprenant par exemple quelques clefs de chiffrement), et de l'ensemble des messages qu'il reçoit, le médiateur peut en décomposer certaines parties, (déchiffrer un contenu à l'aide sa clef publique), procéder à des vérifications (de signature électronique par exemple) avant de construire les messages à envoyer aux services partenaires de l'orchestration.

1.3 Contributions de la thèse

Ce travail contribue à la plateforme de validation du projet AVANTSSAR [1] dont l'objectif global est le développement de technologies supportant *la spécification formelle et la validation automatique de la confiance et de la sécurité dans les SOA*. Le projet AVANTSSAR permet de formaliser et de raisonner automatiquement sur les services, leur composition, propriétés de sécurité et leurs politiques associées.

La plateforme de validation prend en entrée des spécifications formelles décrivant les besoins en terme de confiance et de sécurité (exprimés sous la forme de politiques) ainsi que les modèles des services considérés, incluant leurs comportements (les aspects de sécurité compris) et les politiques qu'ils se doivent de respecter. Ces spécifications peuvent être soit configurées statiquement soit découvertes dynamiquement. Plus de détails sont donnés dans le chapitre 2 dédiée à cette plateforme.

Nous avons contribué à la plateforme au niveau de l'orchestrateur (ou générateur de services médiateurs) de services. Notre travail couvre la formalisation et la résolution du problème d'orchestration [14, 27, 68], ainsi que la génération des spécifications formelles et des programmes exécutables des médiateurs correspondants [60]. Nous avons proposé plusieurs méthodes et outils pour construire des Services-web composés et pour les valider. Ces méthodes et outils ont pu être confrontés à des études de cas industrielles, proposées par des partenaires du projet et ont été publiés dans [14, 27, 68, 60, 61].

– Pour le domaine des services web sécurisés (utilisant la signature électronique ou l'horodatage, par exemple), nous proposons une nouvelle approche automatique pour la composition de services basée sur leurs politiques de sécurité. Étant donné une communauté de services

et un service objectif, nous réduisons le problème de l'existence d'un service médiateur permettant de simuler le service objectif en un problème de sécurité, où un intrus (le médiateur) intercepte, adapte et envoie des messages depuis et vers la communauté de services et un service client jusqu'à satisfaire les requêtes de ce dernier.

- Nous présentons un algorithme qui compile des traces décrivant des conversations de services web en l'ensemble de spécifications formelle et vers les servlets Java correspondants. Pour cela nous calculons d'abord pour chaque partenaires une spécification exécutable aussi prudente que possible de son rôle dans la conversation. Cette spécification est exprimé dans le langage ASLan, un langage formel conçu pour la modélisation des services Web liés à des politiques de sécurité. Ensuite, nous pouvons vérifier avec des outils automatiques que ces spécifications formelles satisfont certaines propriétés de sécurité telles que le secret et l'authentification. Si aucune faille n'est détectée, nous compilons les spécifications ASLan en des servlets Java, que les partenaires peuvent utiliser pour exécuter la chorégraphie. Cet algorithme peut aussi être utilisé dans le monde des protocoles cryptographiques, pour obtenir des implémentations réelles aussi prudentes que possible pour les différents rôles jouant un protocole donné.
- Les deux algorithmes cités plus haut ont été intégrés à la plateforme de validation AVANTSSAR (module Orchestrateur) et testés sur plusieurs études de cas. Les études de cas que nous avons considérées ont été proposées par des partenaires industriels du projet AVANTSSAR et sont décrites ci-dessous :
  - Un service de signature électronique de contrats (DCS) pour deux (ou plusieurs) partenaires s'adressant à travers un accès sécurisé à une tierce-partie : un portail métier. Cette étude de cas a été proposée par la société Opentrust et correspond à un produit qu'elle commercialise.
  - Un service d'enchères publiques (PB), réalisé par un échange sécurisé de documents entre participants et un portail web, permettant la mise en ligne de l'appel d'offres ainsi que la collecte des propositions. Cette étude de cas a également été proposée par la société Opentrust.
  - Une immatriculation à distance de véhicules (CRP), mettant en scène un scénario e-gouvernement où un citoyen dispose d'un point d'accès lui permettant d'adresser des services du secteur publics d'une manière sûre. Cette étude de cas a été présenté par Siemens SA.

  Les études de cas considérées soulèvent des problématiques telles que l'interopérabilité des Services, et les valeurs probante ou sensitive des documents échangés.

1.4 Organisation de la thèse

Dans le chapitre 2 nous décrivons la plateforme AVANTSSAR, où nous avons intégré et testé les outils proposés. Nous décrivons aussi ASLan, un des langages formels supportés par la plateforme AVANTSSAR.

Dans le chapitre 3 nous définissons formellement le problème de composition automatisée de services web, et nous présentons notre approche pour sa résolution. Nous décrivons par la suite l'intégration de notre approche dans le module Orchestrator de la plateforme AVANTSSAR.

Dans le chapitre 4 nous présentons notre algorithme de compilation pour la génération d'implémentation prudentes pour les services médiateurs solutions des problèmes de composition de services web. Les implémentations proposées peuvent alors être directement déployées et exécutées (à l'aide de servlets Java). Nous discutons aussi le fait que nos solutions peuvent être

utilisées dans un contexte où plusieurs sessions parallèles sont possibles, moyennant des hypothèses simples sur la corrélation des messages échangés par les service.

Enfin, nous concluons dans le chapitre 5 en résumant les résultats obtenus lors de l'expérimentation sur des études cas industrielles, en rappelant les principales contributions de ce travail et en donnant une liste non exhaustive de travaux futurs.

Certains contenus ont été mis en annexe. L'annexe A contient une preuve du théorème central dans la preuve de de la correction de l'approche présentée. Dans l'annexe B nous décrivons dans le détail les études de cas traités dans la plate-forme AVANTSSAR (DCS,PB,CRP).

Chapitre 2

La plate-forme de validation AVANTSSAR

Sommaire

Ce chapitre présente la plate-forme de validation de Services-Web sécurisés AVANTSSAR. Le projet AVANTSSAR a pour objectif global le développement de technologies supportant *la spécification formelle et la validation automatique de la confiance et de la sécurité dans les SOA*. Le projet AVANTSSAR permet de formaliser et de raisonner automatiquement sur les services, leur composition, leurs propriétés de sécurité, et leurs politiques associées, aussi bien au niveau réseau qu'au niveau applicatif. Ceci inclut des propriétés standards comme l'authentification et le secret mais aussi l'autorisation, le contrôle d'accès, la délégation de la confiance, les obligations et la gestion des identités. Les technologies développées dans le cadre du projet permettent ainsi le développement des SOA et de garantir leur correction dans le but de renforcer l'acceptation par le grand public des systèmes et applications avancés basés sur les TIC.

Dans ce sens la plateforme de validation AVANTSSAR procure :

- Un langage de modélisation formelle, entièrement dédié à la spécification de la confiance et des aspects de sécurité liés aux services, leur composition, ainsi que des propriétés qu'ils sont tenus de satisfaire et des politiques qu'ils se doivent de respecter ;
- De nouvelles techniques pour raisonner sur la composition dynamique de services, tenant compte de leurs politiques de sécurité associés ;
- Des outils automatisés pour la validation de la confiance et de la sécurité ;
- Une librairie de services composés et prouvés sûrs à l'aide de la technologie de validation proposée par le projet sur des études de cas (preuves de concepts) puisés dans le monde réel, en particuliers ceux fournis par les partenaires industriels du projet.

La plateforme de validation AVANTSSAR et son usage dans le contexte des SOA sont illustrés dans la figure 2.1 ou *TS* est une abréviation de *Confiance et Sécurité* (Trust and Security).

2.1 Structure générale de la plateforme AVANTSSAR

Les principaux composants de la plate-forme sont décrits ci-dessous :

- Le *TS Orchestrator* (ou orchestrateur AVANTSSAR) permet de composer les modèles des services de manière à respecter leurs politiques respectives. Dans le cas d'une composition dynamique, l'orchestration obtenue est synthétisée en utilisant des *TS Wrappers* qui rajoutent les fonctionnalités de sécurité requises par l'orchestration mais non fournies par l'ensemble des services initiaux.
- Le *TS Validator* (ou validateur AVANTSSAR) analyse automatiquement l'orchestration retournée par l'orchestrateur. La validation échoue quand certaines vulnérabilités de l'orchestration sont découvertes. Il est donc nécessaire de les corriger afin de garantir la sûreté de l'orchestration, par rapport aux politiques de sécurité globales.

En outre, si le service composé (ou l'orchestration) est vulnérable, une boucle est alors initiée entre le validateur et l'orchestrateur. Plusieurs alternatives (pouvant être combinées) sont alors possibles pour corriger les failles découvertes : modifier le problème de composition considéré, réviser les politiques locales des services impliqués ou encore enrichir la communauté de services et les politiques globales de l'orchestration. Indépendamment, la possibilité de demander une autre solution au problème de composition est toujours offerte.

La plateforme AVANTSSAR agit au niveau logique. Il est donc nécessaire de transformer les spécifications de services retournées et validées par l'orchestrateur et le validateur, depuis et vers les langages et modèles communément utilisés au niveau applicatif. Cette transformation n'est pas toujours évidente, dans la mesure où les techniques de modélisation utilisées au niveau applicatif ne disposent pas des concepts et de l'expressivité permettant une validation ou une composition automatique des services. Le projet AVANTSSAR adresse explicitement cette transformation, en prenant en compte les langages (et les bonnes pratiques) reconnus par l'industrie et les reliant de manière systématique. Il produit des outils qui permettent aux modeleurs d'étendre leurs modèles, pour supporter une validation automatique. Ceci est d'autant plus nécessaire que l'utilisation des standards industriels reste un fait avéré. Dans ce sens l'approche du projet AVANTSSAR reste constructive, dans la mesure où elle tend à s'intégrer dans les environnements existant plutôt que proposer des changements capitaux. Dans ce sens, la plateforme a été appliquée et testée sur des études de cas industriels, qui adressent des problèmes de sécurité variés, incluant des scénarios de e-business, e-government et e-health. Ces scénarios couvrent un large spectre de propriétés de la sécurité et de la confiance, ainsi que différents niveaux d'abstraction.

La figure 2.1 présente les différents composants décrits plus haut. La plateforme AVANTSSAR implémente aussi un *niveau de de connecteurs* fait de modules qui traduisent des spécifications écrites à l'aide des langages haut niveau (BMPN [32],HL-PSL++ [29], AnB [46],ASLan++ [12]) depuis et vers ASLan [12], le langage d'entrée et de sortie du niveau logique de la plateforme.

Une spécification d'entrée pour la plateforme est composé de :

- *la communauté de services Web disponibles*, c'est à dire l'ensemble des services pouvant assister l'orchestrateur dans sa tâche de recherche d'un médiateur ;
- *les politiques* définissant à la fois les contraintes fonctionnelles et de sécurité du service composé par l'orchestrateur.

Les objectifs fonctionnels expriment les fonctionnalités qui doivent être synthétisées en composant les services disponibles, tandis que les objectifs de sécurité expriment les rapports de confiance et les propriétés de sécurité qui doivent être remplis par le service à synthétiser.

La sortie est une orchestration validé, c'est à dire *un service composé* qui emploie les services disponibles et réalise à la fois les objectifs fonctionnels et de sécurité. Plus exactement :

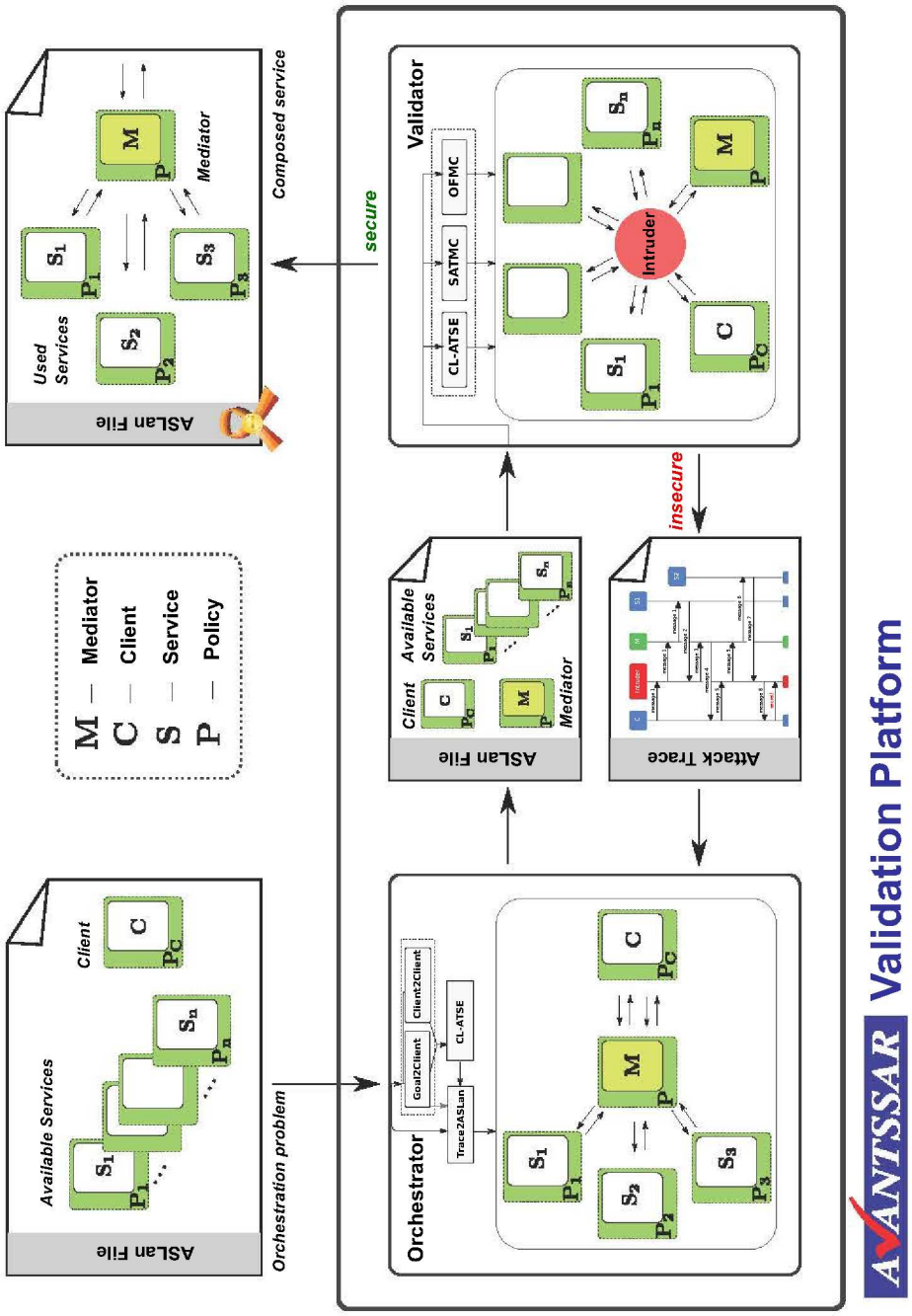


FIG. 2.1 – Avantssar Platform

- L'orchestrateur cherche des entrelacements d'exécution entre les services, où les messages sont au besoin adaptés par un médiateur, et réalisant les objectifs fonctionnels.
- Le validateur vérifie la sécurité de l'orchestration (soit la sortie de l'orchestrateur) en vérifiant si elle satisfait les objectifs de sécurité. Si aucun contre-exemple n'est trouvé, le problème est résolu. Sinon le validateur peut demander à l'orchestrateur de fournir une orchestration alternative.

Les spécifications ASLan se prêtent directement à l'analyse formelle. Des langages plus haut-niveau sont aussi supportés (ASLan++) et sont mieux adaptés pour les développeurs. Les problèmes fournis au niveau logique de la plateforme et résolus par elle, doivent être traduits depuis et vers les artéfacts de modélisation et de langages utilisées dans le niveau applicatif, tels que ceux décrits dans [5].

Ces transformations mises en œuvre dans la couche connecteurs jouent un rôle primordial dans la migration vers l'industrie. En particulier une version annotée de BPMN est utilisé dans la migration vers le développement industriel chez SAP, et chez OpenTrust comme un langage de spécification industriellement adaptées (ISSL) tel que décrit dans [6]. Par ailleurs, les spécifications ASLan peuvent être obtenues en traduisant une spécification ASLan++, à l'aide d'un traducteur automatique développé et intégré à la plateforme. D'autres langages peuvent être utilisées dans la plateforme AVANTSSAR en définissant leurs propres connecteurs vers ASLan. Afin de simplifier l'intégration de nouveaux connecteurs au sein de la platforme AVANTSSAR, nous fournissons également deux convertisseurs qui génèrent la représentation XML d'ASLan et du format de sortie commun des modules de la plateforme.

2.2 ASLan : le langage de spécification d'AVANTSSAR

ASLan : (*AVANTSSAR Specification Language*) est défini comme une extension du *Intermediate Format* (IF) [13]. IF est un langage expressif pour spécifier les protocoles de sécurité et leurs propriétés, et est basé sur la réécriture de multi-ensembles. Comme décrit dans [3], ASLan étend IF par un nombre de fonctions importantes pour adresser différentes politiques de sécurité, les objectifs de sécurité, la communication et l'intrus à un niveau d'abstraction approprié et ainsi permettre la spécification formelle et l'analyse de services complexes.

Plus particulièrement, ASLan étend IF par le support des clauses de Horn et des formules de la logique de temps linéaire LTL. Les invariants du système peuvent alors être définis par un ensemble déterminé de clauses de Horn. Les clauses de Horn ne nous permettent pas seulement de capturer les capacités de déduction de l'intrus de façon naturelle, mais aussi et surtout ils permettent l'incorporation de logiques d'autorisation dans les spécifications de services. Par ailleurs, les propriétés de sécurité complexes peuvent être spécifiées dans la logique LTL. Comme le prouve [4], ceci nous permet d'exprimer des objectifs de sécurité complexes que les services doivent réaliser ainsi que des hypothèses précises sur la sécurité offerte par les canaux de communication.

2.2.1 Syntaxe et sémantique

Ici, nous décrivons les principales caractéristiques de ASLan, pointant le lecteur à [9] pour de plus amples détails sur la langage. Un fichier ASLan consiste en plusieurs sections, parmi lesquelles :

La section Inits

Contient un ou plusieurs états initiaux du système de transition. Un état du système de transition est un ensemble de faits clos (sans variables).

La section Rules

Elle spécifie les transitions du système de transition. Une *transition* est une règle contenant deux parties, un côté gauche (CG) et un côté droite (CD). La règle peut-être activé dans un état donné, si les faits de son CG sont vraies dans cet état. Par ailleurs, une transition peut être étiquetés avec une liste de variables existentiellement quantifiés, dont objectif est d'introduire de nouvelles constantes représentant des données fraîches (des nonces).

Exemple 1 *Un exemple de transition :*

```
step sampleTransition(BankAgent):=
 state_BankingService(BankAgent,1).
 iknows(request)
 =>
 state_BankingService(BankAgent,2).
 iknows(response)
```

où :

- *step est un mot clef utilisé pour définir une nouvelle transition ;*
- *sampleTransition est le nom de la transition ;*
- *BankAgent est un paramètre de la transition ;*
- *state_ BankingService(BankAgent,1), iknows(request), state_ BankingService(BankAgent,2) et iknows(response) sont des faits ;*
- *state_ BankingService(BankAgent,1).iknows(request) est le CG de la transition ;*
- *state_ BankingService(BankAgent,2).iknows(response) est son côté droit (CD).*

Cette transition représente le comportement d'un service bancaire qui reçoit une requête request et réagit en répondant par response et en passant à un état différent. Plus particulièrement la transition peut être activée s'il existe une valeur val de la variable BankAgent telle que state_ BankingService(val,1) et iknows(request) sont dans l'état courant. L'effet de l'exécution de la transition est de remplacer state_ BankingService(val,1) par le fait state_ BankingService(val,2) et de rajouter le nouveau fait iknows(response).

Les envois et réceptions de messages sont représentés par les faits iknows : le fait iknows dans le CG d'une transition modélise la réception d'un message, tandis que dans le CD de la transition, il modélise un envoi de message. A titre d'exemple, le fait iknows(request) de l'exemple 1 modélise un phénomène persistent : une fois émis, une message ne disparait pas de la mémoire de l'environnement (l'intrus).

Si le CG d'une transition est vrai dans l'état actuel, il est supposé que les connaissances (représentées par un ensemble de faits clos) du service correspondant sont suffisantes pour construire le messages déclaré dans le fait iknows de son CD.

Pour spécifier les états des différents services nous utilisons un prédicat par service : state_ suivi par le nom du service, par exemple, dans l'exemple 1, state_BankingService.

La section Goals

Cette section contient les objectifs de sécurité définis soit par des états d'attaque (états spéciaux du système de transition) soit par des formules LTL.

Exemple 2 *Sample attack state.*

```
attack_state stateName(Msg):=
    fact1(Msg).
    fact2(Msg)
```

Ici l'état d'attaque stateName est atteint, s'il existe une valeur val de la variable Msg telle que fact1(val) et fact2(val) sont dans l'état courant du système de transition.

La section HornClauses

Elle contient un ensemble fini de clauses de Horn, permettant d'exprimer des logiques d'autorisation.[1]

2.3 Méthodes formelles et validation de propriétés de sécurité

2.3.1 Systèmes de contraintes de déductibilité

Dans cette section nous présentons comment l'analyse de protocoles de sécurité est ramenée à la résolution d'un système de contraintes de déductibilité. Le but est de donner un aperçu de l'apport des méthodes symboliques dans le domaine des protocoles de sécurité.

Nous nous concentrons sur une propriété de sécurité, la confidentialité. Étant donné un protocole cryptographique, où des agents échangent des messages particuliers, un intrus de Dolev et Yao peut-il accéder à certaines sous-parties de ces messages ? Ceci dépend des connaissances initiales de l'intrus, et des différents contrats offerts par les autres agents dits honnêtes qui vont permettre de les enrichir. Ces différentes facettes du problème sont traduites en un système de contraintes de déductibilité qui conditionne l'évolution des connaissances de l'intrus.

Nous décrivons le protocole du point de vue de l'environnement dans lequel il s'exécute ; un environnement contrôlé par un intrus de Dolev et Yao. Cet intrus un contrôle total sur tous les messages émis par des agents honnêtes du protocole. Il peut les intercepter, éventuellement les modifier, avant de les transférer à la destination voulue initialement ou à une destination différente, ou tout simplement bloquer leur transmission. Les modifications sur les messages accessibles à l'intrus sont décrites par des règles de déduction. Dans le modèle de Dolev et Yao, l'intrus peut décomposer des messages soit en déchiffrant des messages à condition de détenir les clefs correspondantes, soit en dé-concaténant une chaîne de messages. Il peut aussi en construire en chiffrant des messages qu'il connait à l'aide de clefs qu'il détient ou encore concaténer des messages dans ses connaissances.

D'abord les activités de communication locales de chaque agent sont regroupées en étapes faites de paires de réception/émission respectant l'ordre de la séquence du protocole :

$$M_1 \rightarrow M_2 \ldots M_{n-1} \rightarrow M_n.$$

Les différentes exécutions possibles du protocole sont alors décrites par les entrelacements possibles entre toutes les étapes locales de tous les agents participants. Ensuite des contraintes décrivent l'évolution des connaissances de l'intrus, pour chaque entrelacement possible. Les

[1] extraits de [9]

contraintes pour un entrelacement (ou ordre d'exécution du protocole) $\{u_i \rightarrow v_i\}_{1 \leq i \leq n}$ sont alors les suivantes :

$$\begin{cases} S & \triangleright & u_1 \\ S \cup \{v_1\} & \triangleright & u_2 \\ & \dots & \\ S \cup \{v_1, \dots, v_{n-1}\} & \triangleright & \end{cases}$$

Informellement, ces contraintes spécifient qu'à la fin de chaque pas d'exécution du protocole, les connaissances de l'intrus sont enrichies par le dernier message émis et qu'elles doivent être suffisantes pour construire le message attendu dans le pas d'exécution suivant du protocole. Les connaissances finales de l'intrus apparaissent dans la dernière contrainte. Il est possible d'encoder dans cette contrainte une violation de la confidentialité d'une certaine valeur $mySecret$ apparaissant dans les messages du protocoles en la transformant en :

$$S \cup \{v_1, \dots, v_{n-1}\} \quad \triangleright \quad mySecret$$

Le système de contraintes ainsi formé décrit un problème de sécurité bien particulier où à l'issue d'un certain ordre d'exécution du protocole l'intrus collecte suffisamment de connaissances pour déduire une valeur sensitive du protocole. Un système de contraintes est satisfiable s'il existe une valuation des variables pour laquelle toutes les contraintes sont satisfaites. Dans notre cas ceci correspond à une attaque possible sur le protocole portant sur la confidentialité d'une certaine valeur sensitive du protocole.

Exemple 3 *Nous considérons un exemple simple de protocole où un agent A envoie à un agent B un message chiffré par une clef symétrique kab. B déchiffre le message et le renvoie à A.*

```
1. A -> B : {M}kab
2. B -> A : M
```

Dans notre exemple M correspond à une valeur arbitraire, par contre kab est une constante fixée. Ceci veut dire que B s'attend à recevoir un message quelconque chiffré par la clef symétrique kab. Les activités de communications locales pour chaque agents sont alors décrites ci-dessous. Pour A cela donne :

$$\rightarrow \{X\}kab$$

et pour B :

$$\{Y\}kab \rightarrow Y$$

Nous supposons de plus que l'intrus connait déjà initialement $\{secret\}kab$ une certaine valeur secrète chiffrée à l'aide de la clef kab. Et nous voulons savoir si l'intrus arrive à déduire la valeur secret au terme d'une exécution du protocole.

Le problème est dans notre cas décrit par le système de contraintes suivants :

$$\begin{cases} 1. & \{\{secret\}kab\} & \triangleright & \\ 2. & \{\{secret\}kab, \{X\}kab\} & \triangleright & \{Y\}kab \\ 3. & \{\{secret\}kab, \{X\}kab, Y\} & \triangleright & secret \end{cases}$$

Les contraintes 1. *et* 2. *correspondent aux activités de communication de A et B. La contrainte* 3. *quant à elle décrit une violation de la propriété de secret où l'intrus arrive à déduire la constante secret, à partir des messages qu'il aura déjà observé.*

Un système de contraintes est satisfiable s'il existe une valuation des variables pour laquelle toutes les contraintes sont satisfaites. Dans notre exemple l'affectation suivante de variables est une solution :

$$\{Y \rightarrow secret\}.$$

Elle correspond au cas où l'intrus envoie la valeur qu'il détient $\{secret\}kab$ à B. Ce dernier la déchiffre et renvoie *secret* à l'intrus. Ceci correspond à une attaque du protocole, compte tenu de l'hypothèse faite sur la confidentialité de *secret*.

L'outil CL-Atse [62], l'un des outils de validation de la plateforme AVANTSSAR permet de résoudre des systèmes de contraintes de déductibilité tels que celui de l'exemple 3. Nous montrons dans le chapitre suivant comment un problème de composition de services web est encodé en un problème de secret. Ensuite nous utilisons l'outil CL-Atse pour analyser la satisfiabilité du système de contraintes de déductibilité correspondant et ainsi résoudre le problème de composition de service web.

2.4 Conclusion

Dans ce chapitre nous avons présenté la plateforme de validation de Services-Web sécurisés AVANTSSAR. Elle permet de formaliser et de raisonner automatiquement sur les services, leur composition, leurs propriétés de sécurité, et leurs politiques associées.

Dans le chapitre suivant nous décrivons la formalisation d'un problème de composition de services à l'aide du langage ASLan. Pour résoudre le problème nous utilisons ensuite l'outil CL-Atse [63] décrit dans la suite et permettant la vérification formelle de protocoles de sécurité. Nous décrivons comment nous le ramenons à un problème d'accessibilité d'un certain état à partir d'une configuration initiale. D'un point de vue de validation, résoudre un problème de composition de services reviendrait à trouver une "attaque", où le médiateur, pourvu des capacités de décomposition de messages définis par Dolev et Yao [37]), satisferait toutes les requêtes d'un client, représentant le contrat à remplir par le médiateur.

Chapitre 3

Composition automatique de services web

Sommaire

3.1 Introduction à la composition de services web

Pour le domaine des services web sécurisés (utilisant la signature électronique ou l'horodatage, par exemple), nous proposons une nouvelle approche automatique pour la composition basée sur leurs politiques de sécurité. Étant donné une communauté de services et un service objectif, nous réduisons le problème de l'existence d'un service médiateur permettant de simuler le service objectif en un problème de sécurité, où le médiateur intercepte, adapte et envoie des messages depuis et vers la communauté de services et un service client jusqu'à satisfaire les requêtes de ce

dernier. Ainsi, dans le cadre de la composition de services, le médiateur joue le rôle jouée par un attaquant lors de la validation de protocoles.

3.1.1 Travaux reliés

Une approche commune pour la composition statique [21] (ou syntaxique [22]) de services web est l'orchestration. L'orchestration de services web [56] fait intervenir une entité centrale appelé *orchestrateur* ou *médiateur* qui agit comme un liant entre un service client et une communauté de services disponibles. Cette interaction contrôlée par le médiateur fournit un service demandé par le client.

La plupart des travaux dans le domaine de la composition de services web s'articule autour d'un modèle comportemental [36] où les services mémorisent leurs états respectifs et où ils sont utilisés par le client selon plusieurs scénarios possibles. Le comportement d'un service est représenté par une structure de kripke, où les transitions sont labellisées par les opérations du service [55]. Dans cette section, nous proposons un survol des travaux reliés à la compositions de services web.

Une composition basée sur les conversations (une conversation étant une séquence de messages échangés entre pairs) a été présentée dans [23]. Les services disponibles sont représentés par trois ensembles finis : un ensemble de classes de messages, un ensemble de pairs (ou partenaires) abstraits (les services), et un ensemble de canaux de communication dirigés. Chaque canal est composé par deux extrémités (endpoints) et un ensemble de classes de messages pouvant être envoyés sur le canal. Le problème de composition revient alors à trouver une machine de Mealy pour chaque pair telle que l'ensemble des conversations possibles qui peuvent être vus par un observateur externe est équivalent à une spécification donnée de la conversation.

Un travail dans la même lignée a été publié dans [19]. Les auteurs suggèrent une manière de synthétiser et de vérifier les protocoles cryptographiques étant donné un ensemble de sessions multipartites qui représentent les conversations possibles entre les participants. Les sessions multipartites sont représentées par des graphes orientés dont les nœuds sont étiquetés avec les noms des rôles du protocole, et dont les arêtes sont étiquetées avec des descripteurs de messages. Pour chaque paire de nœuds reliés par une arête, les auteurs définissent deux ensembles de variables typées : le premier déclare les variables dans lesquelles le rôle étiquetant le nœud source peut écrire et le second déclare les variables à partir desquelles le rôle étiquetant le nœud destination peut lire. Étant donnée la description des sessions multipartites et les deux ensembles de variables typées, les auteurs proposent de construire d'abord la projection pour chaque rôle, avant de la renforcer en ajoutant une couche de sécurité afin de garantir des propriétés d'intégrité et de confidentialité. En outre, un prototype du compilateur qui implémente cette approche génère des interfaces et des implémentations en langage ML pour les protocoles ainsi générés.

Notons que ces approches visent à *générer* des implémentations pour des pairs ayant un comportement particulier. Nous nous intéressons aussi à cette problématique dans le chapitre 4 Néanmoins, dans le chapitre courant nous mettons l'accent sur la composition automatique de services existants dans le but de synthétiser un service qui est capable de satisfaire un client donné.

Dans le modèle Romain (Roman model) [40] les services disponibles sont représentés par des machines à états finis. Dans [55] l'auteur étend ce modèle par un non-déterminisme permettant le partage de la mémoire. En outre, l'approche permet de trouver un *générateur fini d'orchestrateurs* qui permet de déduire toutes les orchestrations possibles. Notez que dans tous ces travaux, les services acceptent uniquement des données avec des domaines finis. Ceci a motivé l'extension *COLOMBO* [16] pour la composition de services qui prend en compte des données avec des

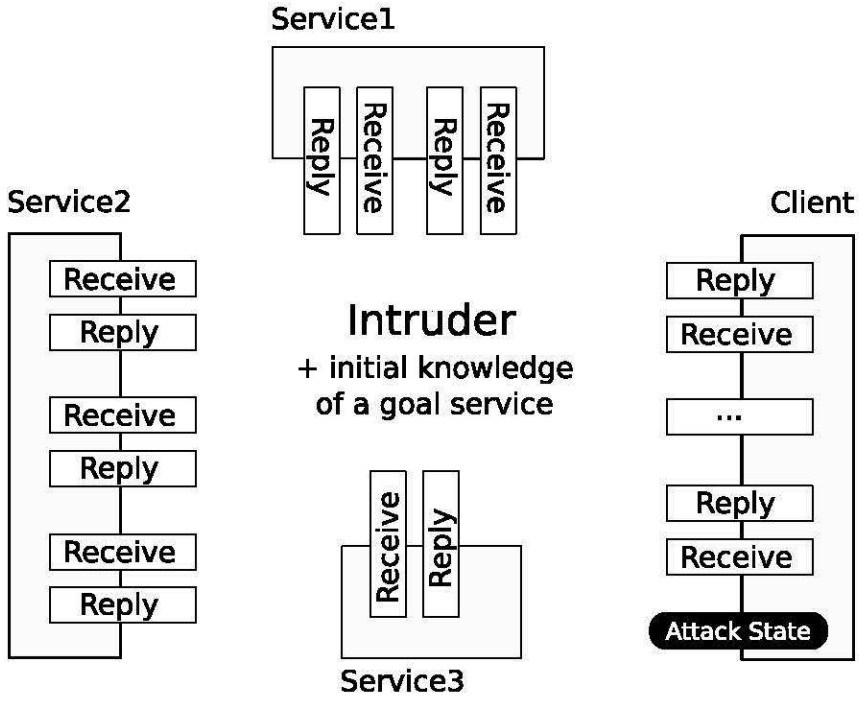


FIG. 3.1 – Les services web en tant que rôles de protocoles

domaines infinis. Le problème de composition revient à trouver un service *médiateur* qui utilise les messages pour interagir avec les services disponibles et un client de telle sorte que le système global simule le comportement d'un service objectif. Néanmoins, même dans cette dernière approche, le médiateur ne gère pas les primitives cryptographiques, telles que le déchiffrement d'un message chiffré.

3.1.2 Notre approche pour la composition de services web

Pour le domaine des services sécurisés (par exemple utilisant la signature numérique ou l'horodatage), nous proposons une nouvelle approche pour la composition automatique de services basée sur leurs politiques de sécurité. Étant donné une communauté de services et un service objectif, nous réduisons le problème de la composition du service objectif à partir de services dans la communauté en un problème d'insécurité, où un intrus (considéré comme un service médiateur) doit intercepter et rediriger les messages depuis la communauté de services vers un service client jusqu'à ce que l'ensemble atteigne un état satisfaisant. L'idée a été présentée en [30]. Nous utilisons des termes du premier ordre (avec des symboles spécifiques pour les fonctions cryptographiques) pour décrire le profil d'un service web. La partie comportementale du service est quant à elle décrite par un système de transitions dans le langage ASLan [1]. Puisque nous sommes à la recherche d'une orchestration, tous les messages émis par les services de la communauté ou par le client doivent être reçus par le médiateur. Réciproquement, les messages qui sont consommés par les services ou le client doivent être produits par le médiateur. Le médiateur est capable de produire de nouveaux messages en utilisant les règles dérivées du modèle de Dolev-Yao [37] pour les fonctions cryptographiques. Le problème de composition est alors posé comme un problème d'accessibilité : étant donnés les services disponibles, le client et les connaissances initiales du médiateur en tant que fichier ASLan, existe-t-il une séquence d'actions exécutée par le médiateur à l'issue de laquelle le client atteint un état final ?

TAB. 3.1 – Composition de services vs Insécurité des protocoles cryptographiques

Services	Protocoles
Services disponibles/Client	Rôles
Médiateur	Intrus
État final	Attaque

La même approche est également utilisée pour définir et résoudre le problème d'insécurité pour les protocoles cryptographiques. Le services étant similaires aux rôles des protocoles cryptographiques, et le médiateur étant basé sur le modèle classique d'intrus par Dolev et Yao nous utilisons un outil de vérification pour les protocoles cryptographiques pour résoudre le problème de composition. Nous illustrons une certaine équivalence entre le problème de composition de services et celui de l'insécurité des protocole cryptographiques dans le tableau 3.1.

Afin de vérifier si l'état final peut être atteint, nous avons utilisé une version du vérificateur de protocoles cryptographiques CL-Atse [62] : le résultat est une trace contenant la séquence des messages envoyés et reçus par le médiateur. De la trace nous extrayons un fichier ASLan exécutable correspondant à la spécification du médiateur. Cela inclut les opérations détaillées pour générer les messages envoyés par le médiateur et les tests de sécurité à accomplir par ce dernier sur les messages reçus.

L'idée générale est de représenter les services disponibles et le client en tant que rôles d'un protocole cryptographique. L'intrus avec les capacités de calcul de Dolev-Yao, qui a un contrôle total sur le réseau, va jouer le rôle d'un orchestrateur : il essaie de faire évoluer le système de transition donné à partir de son état initial vers un état final acceptable. Ainsi les états finaux sont encodés comme des états d'attaque (du point de vue de l'intrus). S'il réussit cela signifie qu'il est capable de satisfaire toutes les demandes du client à partir des connaissances initiales du service objectif et en utilisant les services disponibles dans la communauté (voir 3.1).

Pour vérifier si un tel état d'attaque peut être atteint, nous employons une version d'un outil d'analyse de protocoles cryptographiques CL-Atse. Le résultat est une trace contenant la séquence de messages envoyés et reçus par l'intrus. A partir de cette trace, nous extrayons une spécification ASLan exécutable du service objectif. Cette procédure d'extraction est détaillée dans le chapitre 4.

Comme le service objectif a au moins les mêmes capacités de construction de messages qu'un intrus de Dolev et Yao, si l'intrus peut effectuer les étapes nécessaires pour construire les messages qui apparaissent dans la trace, nous sommes assurés que le service qui en résulte est exécutable. La spécification de la trace d'attaque résultante n'illustre pas les opérations détaillées pour la construction des messages, elles fournit uniquement les invocations nécessaires des services disponibles avec des messages déjà construits. Si une spécification partielle du service objectif est fournie, nous procédons d'abord à la génération de son client hypothétique correspondant (voir 3.3) ensuite nous procédons comme décrit plus haut.

3.1.3 Limites de l'approche

Il existe quelques limites pour la spécification d'un problème d'orchestration en ASLan principalement dues à la puissance expressive du langage. Nous discutons ces limites dans la suite.

En générant une solution du problème de composition, chaque transition du client ou du services objectif ne peut être exécutée qu'un nombre borné de fois. Des restrictions plus sévères sont imposées en particulier si une spécification partielle du service objectif est fournie :

– Le service Objectif est décrit par un système de transitions uniquement en termes des

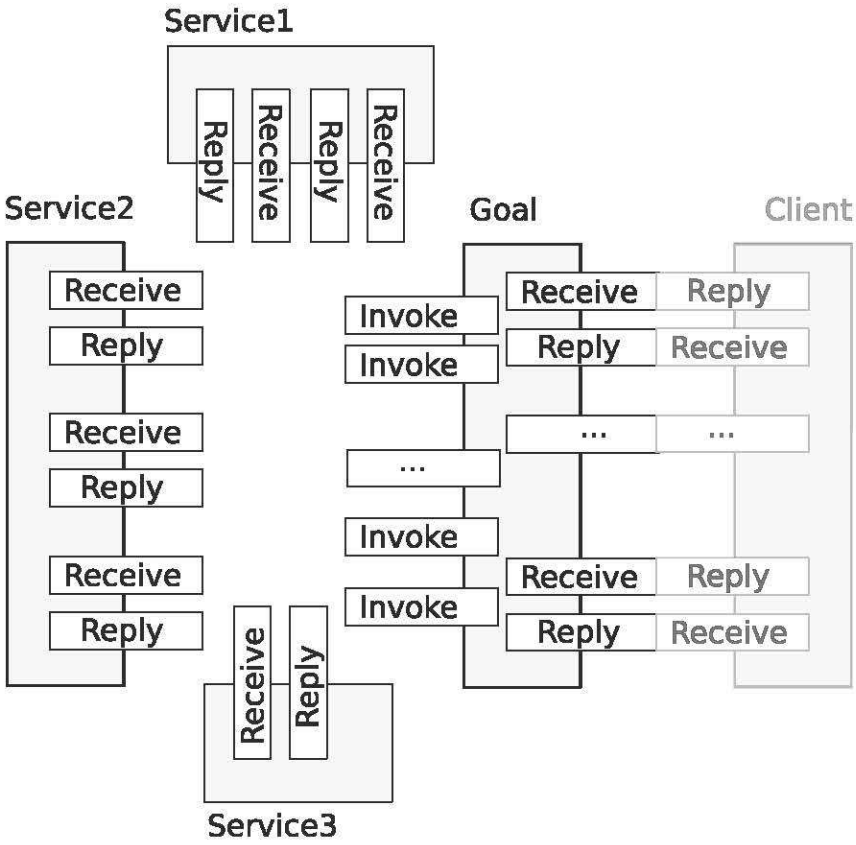


FIG. 3.2 – Solution du problème d'orchestration

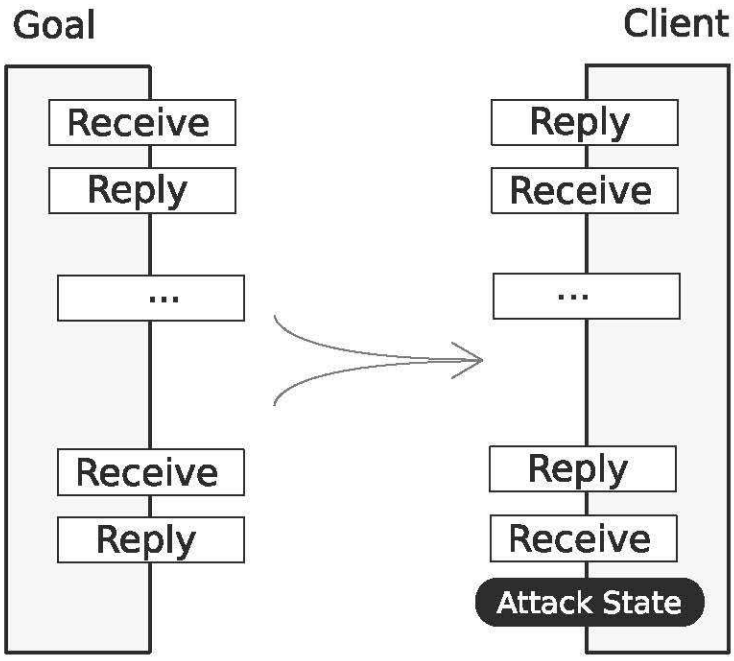


FIG. 3.3 – Service objectif et son client hypothétique

échanges de messages (conversation) avec le client (dans le le style d'Alice et Bob). Toutes les autres contraintes de composition doivent être exprimées dans la section section goals à l'aide de formules LTL.
- Le workflow du service objectif doit être linéaire (pas de boucles itératives ou de branchements). Ainsi au plus un fait iknows est autorisé dans les CG et CD d'une transition.
- Seuls les tests d'égalité sont autorisés dans les transitions du service objectif. (celle contenant un fait state_OrchestrationGoal)
- Toutes les variables utilisées dans la définition d'une transition contenant le fait state_OrchestrationGoal doivent avoir des types atomiques. (pas de variables de type message ou de types composés du style apply(fun, pair(text1,text2)))
- Les connaissances initiales du client ne sont pas prises en compte.

Ainsi un patron des transitions de communication pour le service objectif est :

```
step step_name(<params>):=
      state_OrchestrationGoal(<terms1>).
      iknows(<term2>) &
      equals(<term3>, <term4>) &
      equals(<term5>, <term6>) &
          ...
  =[exists <vars>]=>
      state_OrchestrationGoal(<terms7>).
      iknows(<term8>)
```

La transition doit avoir un fait state_OrchestrationGoal dans ces CG et CD. Par ailleurs elle doit définir uniquement des faits iknows et uniquement des tests d'égalité comme conditions. Des labels de quantification existentielle sur les variables sont aussi autorisés.

3.1.4 Exemple introductif pour la composition de services

La figure 3.4 illustre un problème de composition correspondant à la création d'un nouveau service dénommé *Goal*. Le service *Goal* rajoute un horodatage à la signature électronique d'un document *data* produite par une partenaire *Client*, avant de soumettre le résultat, accompagné d'un certain nombre de preuves et de certificats, à un tiers d'archivage pour une conservation de longue durée. Plus précisément, *Goal* attend un premier message de *Client* contenant un identificateur de session *sid*, son certificat contenant son identité et sa clef publique *ckey* et le document objet de la signature. Ensuite *Goal* répond avec un message contenant le même identificateur de session et une charte *footer* que *Client* devra signer avec le document. En effet c'est ce que *Client* est supposé renvoyer à *Goal* à l'étape suivante. *Goal* rajoute alors à la signature électronique *SIGNATURE* reçue un horodatage *TIMESTAMP* qui consiste en une valeur de temps *time* qui est associée à la signature du *Client* grâce à l'utilisation d'un hachage *md5* et où le tout est signé par un service horodateur de confiance à l'aide de sa clef privée *#2*. *Goal* doit aussi inclure un certains nombre d'assertions dans son message de réponse, constituant des preuves sur son contenu. Ces assertions sont stockées dans la variable *ASSERTIONS* décrite ci-dessous.

```
ASSERTIONS = ASSRT0,ASSRT1,ASSRT2,ASSRT3
ASSRT0 = assertion(cOCSPR,#0,crypt(inv(#0),cOCSPR))
cOCSPR = ocspr(name,ckey,time)
ASSRT1 = assertion(tsOCSPR,#0,crypt(inv(#0),tsOCSPR))
tsOCSPR = ocspr(#1,#2,time)
ASSRT2 = assertion(arcOCSPR,#0,crypt(inv(#0),arcOCSPR))
```

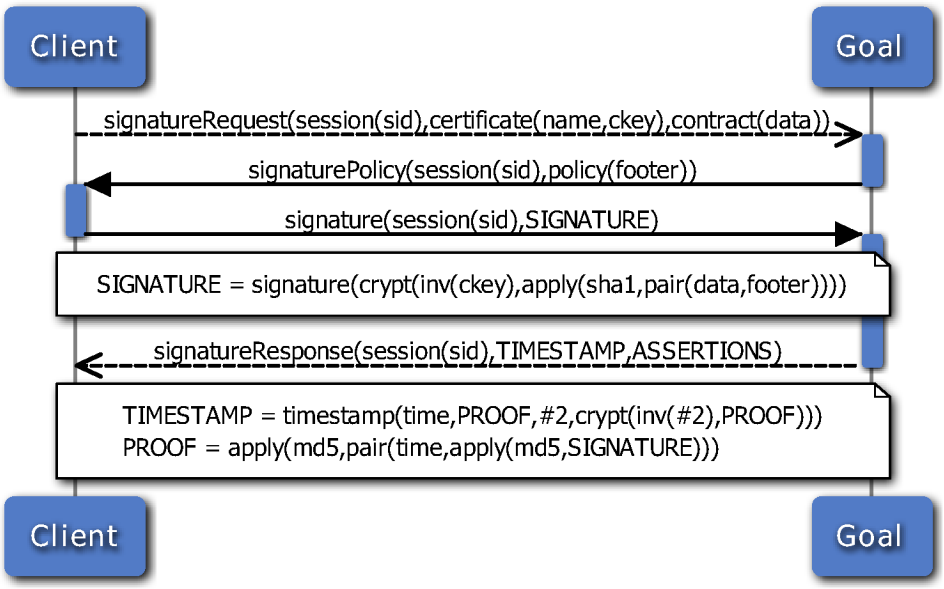


FIG. 3.4 – Horodatage et archivage d'une signature électronique

```
arcOCSPR = ocspr(#3,#4,time)
ASSRT3 = assertion(ARCH,#4,crypt(inv(#4),ARCH))
ARCH = archived(session(sid),certificate(name,ckey),
      contract(data), SIGNATURE,TIMESTAMP,ASSRT0,ASSRT1)
#0 in trustedCAKeys
pair(#1,#2) in trustedTSs
pair(#3,#4) in trustedARs
```

ASSRT0 est une preuve de la validité du certificat du *Client* à la date *time* signée par une autorité de certification de confiance pour le *Client*. Cette relation de confiance est modélisée par l'appartenance de la clef publique de l'autorité de certification à *trustedCAKeys*, l'ensemble des clefs publiques de toutes les autorités de certification digne de confiance. *ASSRT1,ASSRT2* représentent des preuves similaires de la relation de confiance avec les services d'horodatage et tiers d'archivage, toujours signées par la même autorité de certification. D'un autre côté *ASSRT3* représente une preuve du fait que le document devant être signé par le *Client*, sa signature électronique munie d'un horodatage et toutes les preuves obtenues pour tous les certificats impliqués ont été correctement archivés par une tiers d'archivage digne de la confiance du *Client*. Là encore, cette relation de confiance est modélisée par la contrainte *pair(#3,#4) in trustedARs*. Ces assertions sont encodées par des certificats faisant partie des messages échangés. Elles représentent soit une condition ϕ qui doit être évaluée par un service avant d'accepter un message reçu ou une garantie ψ qui doit être assurée par le service qui envoie le message.

Par ailleurs l'utilisation de lignes de communication discontinues dans la figure 3.4 représente des contraintes additionnelles sur les canaux de communication utilisés par *Client* et *Goal*. Dans notre exemple elles concernent le transport des messages qui doit être effectué selon le protocole *SSL*. Dans notre modèle cette contrainte peut s'exprimer par le fait que les messages concernés doivent être chiffrés avec une clef symétrique initialement partagée par les deux services. (La phase de partage n'est pas gérée par le service composé)

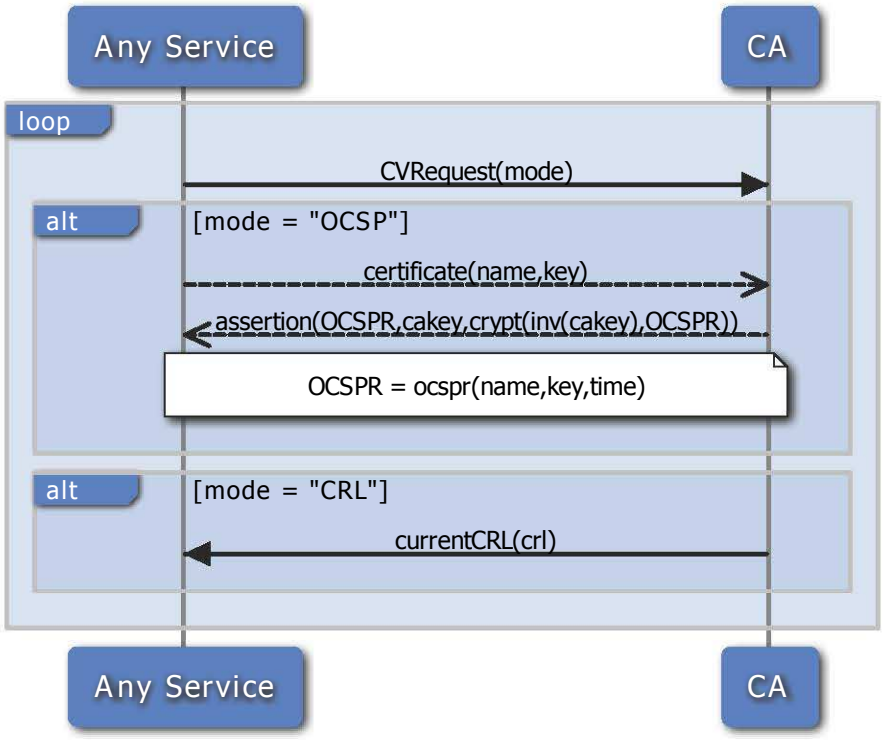


FIG. 3.5 – Services disponibles : L'autorité de certification

Afin de satisfaire les demandes du *Client*, *Goal* s'appuie sur une communauté de services disponibles comprenant horodateurs, tiers d'archivages et autorités de certification.

Ces services sont également donnés par leur interface, soit la description des patrons précis des message qu'ils acceptent et qu'ils fournissent en réponse. Par exemple, la figure 3.5 décrit une autorité de certification CA capable de fournir deux sortes de réponses lorsqu'on l'interroge sur la validité d'un certificat : la première est basée sur $OCSP$ (c'est à dire basée sur le protocole Online Certificate Status Protocol) et retourne une preuve de la validité d'un certificat donné en temps réel, tandis que la seconde se contente de fournie la liste des certificats déjà révoqués (ou CRL). En inspectant le problème de la composition, on peut intuitivement penser que pour satisfaire les requêtes du *Client* le second mode doit toujours être employé avec CA (à condition qu'elle soit aussi digne de la confiance du *Client*). On peut aussi déduire que certaines d'adaptation doivent être faites sur les messages du *Client* avant de les acheminer vers la communauté des services disponibles afin d'obtenir les réponses attendues (éventuellement contenant les bonnes assertions) depuis la communauté (par exemple l'utilisation du drapeau $OCSP$ avec CA).

La solution que nous proposons calcule, quand cela est possible, la séquence d'appels vers les services dans la communauté. Cette séquence peut-être enrichie par les adaptations devant être effectuées sur les messages déjà reçus et permettant de satisfaire les requêtes du *Client* exactement comme cela a été spécifié dans le problème de la composition.

3.2 Composition de services web par synthèse de médiateurs

3.2.1 Représentation des messages et des politiques de sécurité

Nous utilisons des termes du premier ordre pour représenter les messages échangés par un service. Nous rappelons cette notion ci-dessous.

Les termes

Nous considérons un ensemble infini de constantes libres $Consts$ et un ensemble infini de variables $\mathcal{X}$. Pour chaque signature $\mathcal{F}$ (comprendre un ensemble de symboles avec arités) on dénote par $\mathrm{T}(\mathcal{F})$ (resp. $\mathrm{T}(\mathcal{F},\mathcal{X})$) l'ensemble des termes sur $\mathcal{F} \cup Consts$ (resp. $\mathcal{F} \cup Consts \cup \mathcal{X}$). Le premier est appelé l'ensemble des termes clos (ou messages), tandis que le second est simplement appelé l'ensemble des termes sur $\mathcal{F}$. Étant donné une terme t, on dénote par $\mathrm{Var}(t)$ l'ensemble des variables apparaissant dans t. Un terme t est dit *clos*, si $\mathrm{Var}(t)$ est l'ensemble vide. Une substitution σ est application idempotente de $\mathcal{X}$ dans $\mathrm{T}(\mathcal{F},\mathcal{X})$ telle que $\mathrm{Supp}(\sigma) = \{x \,|\, \sigma(x) \neq x\}$, où l'ensemble de définition (ou *support* de σ dénoté $\mathrm{Supp}(\sigma)$) est un ensemble fini. L'application d'une substitution σ à un terme t (resp. à un ensemble de termes E) est notée $t\sigma$ (resp. $E\sigma$) et vaut le terme t (resp. E) où toutes les variables x ont été remplacées par le terme $x\sigma$. Les termes sont manipulés par des *opérations* définies par un sous-ensemble $\mathcal{F}_p$ de la signature $\mathcal{F}$ appelé *l'ensemble des symboles publics*. Un contexte $C[x_1,\ldots,x_n]$ est un terme dans lequel tous les symboles sont publics et où les symboles finaux sont dans l'ensemble $\{x_1,\ldots,x_n\}$. $C[x_1,\ldots,x_n]$ est aussi noté C quand il n'y a pas ambiguïté et n est appelée sa longueur. On définit *l'application* d'un contexte C de longueur n sur la séquence de messages $m_1,\ldots,m_n$ comme étant l'image de $C[X_1,\ldots,X_n]$ par la substitution $\{X_j \to m_j\}_{1\leq j\leq n}$. Une *théorie équationnelle* $\mathcal{E}$ est définie par un ensemble d'équations $u = v$ avec $u, v \in \mathrm{T}(\mathcal{F},\mathcal{X})$ et on écrit $s =_{\mathcal{E}} t$ la relation de congruence correspondante entre deux termes s et t. Cette théorie équationnelle est introduite pour spécifier les effets des opérations sur les messages et les propriétés de ces derniers.

Messages XML

Nous avons pour objectif de représenter un fragment significatif des messages *XML* en utilisant des termes du premier ordre définis sur une signature donnée ci-dessous. Le fragment que nous adressons correspond au éléments *XML*, décrits comme une séquence de types complexes, soit les éléments ayant une séquence ordonnée, de cardinalité fixe d'enfants. Nous faisons également abstraction des attributs dans les messages *XML*. Pour représenter les messages *XML* décrits plus haut, nous définissons la signature suivante :

$$\begin{aligned}\mathcal{F} = \quad & \left\{ node_a^n, child_{\frac{i}{a}}^n \mid i \leq a \in \mathbb{N}, n \in Consts \right\} \cup \\ & \{scrypt, sdcrypt, crypt, dcrypt, sign, verif, inv, invtest, \top\}\end{aligned}$$

où le symbole $node_a^n$ représente un noeud *XML* nommé n, ($n \in Consts$) et ayant a fils. Pour tout symbole $node_a^n$ nous définissons l'ensemble des symboles $child_{\frac{1}{a}}^n, \ldots, child_{\frac{a}{a}}^n$ permettant d'extraire ses enfants. Pour modéliser les contraintes de sécurité qui portent sur les messages *XML* échangés, nous représentons les primitives cryptographiques usuelles en utilisant les symboles suivants : $scrypt/sdcrypt$ pour le chiffrement et déchiffrement symétrique, $crypt/dcrypt$ pour le chiffrement et déchiffrement asymétrique, $sign/verif$ pour la signature électronique et sa vérification. En outre, inv dénote les inverses de clefs et $invtest$ permet de tester si une paire de clefs $\{t, t'\}$

vérifie $t' = inv(t)$ et où la constante $\top$ est le résultat d'un test positif. Dans la suite du chapitre nous notons $\mathcal{F}_p$ l'ensemble des symboles publics avec l'hypothèse suivante : $\mathcal{F}_p = \mathcal{F} \setminus \{inv\}$.

Certains de ces symboles représentent les opérations possibles sur les des messages. Leur sémantique est définie avec la théorie équationnelle suivante :

$$\mathcal{E}_{XML} \left\{ \begin{array}{rcll} sdcrypt(y, scrypt(y,x)) & = & x & (D_s) \\ dcrypt(inv(y), crypt(y,x)) & = & x & (D_{as}) \\ verif(y, x, sign(x, inv(y))) & = & \top & (S_v) \\ child^n_{\frac{i}{a}}(node^n_a(x_1, \ldots, x_a)) & = & x_i & (P_{\frac{i}{a}}) \\ invtest(x, inv(x)) & = & \top & (I_v) \end{array} \right.$$

3.2.2 Représentation formelle des services

Nous remarquons que la spécification *WSDL* d'un service web ne précise pas l'ordre d'invocation de ses opérations, elle en donne uniquement une liste exhaustive. En outre, cette spécification ne mentionne pas les liens entre paramètres d'entrée et paramètres de sortie pour une opération donnée. Le langage *BPEL* [53] permet de raisonner sur de telles propriétés en permettant d'abord de préciser une certaine logique pour le service (workflow), ensuite de mentionner toutes les manipulations nécessaires pour construire les messages envoyés à partir de ceux reçus par le service.

Dans la suite du chapitre nous considérons des services qui ne contiennent pas d'itération ou de réplication. Nous faisons aussi abstraction des traitements internes d'un service et nous nous concentrons sur la communication. Ainsi un service sera considéré comme une séquence de réceptions et d'envois de messages, dénotés respectivement par *RCV(m)* et *SND(m)*

Nous faisons aussi l'hypothèse que les services sont décrits par leur spécifications *BPEL* représentant des processus métier linéaires. Ainsi un service S est décrit par la grammaire suivante :

$P, Q := services$	
0	null service
$RCV(m) \cdot P$	input message
$SND(m) \cdot P$	output message

La mise en parallèle de deux services S_1 et S_2 est notée $S_1 \parallel S_2$. Elle est associative et commutative et a un élément neutre noté 0, appelé le processus nul. Une communauté de services est alors définie comme la mise en parallèle de tous les services qu'elle comprend.

$P, Q := services$	
$P \parallel Q$	AC parallel composition

Sémantique de transition

Nous introduisons une sémantique de transition pour définir comment les services sont exécutés en interaction avec leur environnement, et en particulier avec leurs clients. Par exemple un service dans l'état $RCV(r) \cdot S'$, qui attend un message correspondant au format r, continuera son exécution avec $S'\sigma$ s'il est mis en parallèle avec un service dans l'état $SND(m) \cdot S$ et si m est unifiable avec r via une substitution σ. Dans ce cas le dernier service continuera son exécution avec S. La configuration globale est une paire $(\mathcal{S}, \mathcal{E})$ où le premier argument est l'ensemble des états des services disponibles et où le second argument est l'ensemble de tous les messages émis jusque-là. L'évolution de la configuration est donnée par la règle de transition suivante :

$$(RCV(r) \cdot S \parallel (SND(m) \cdot S' \parallel \ldots, \mathcal{E}) \xrightarrow{m} (S\sigma \parallel S' \parallel \ldots, \mathcal{E} \cup \{m\}) \quad \text{si } \exists \sigma, r\sigma = m$$

De telles transitions sont appelés *des transitions de communication.*

La réception d'un message instancie les variables dans le patron du message correspondant. Cette instanciation est appliquée sur les variables restées libres dans le processus décrivant le service. Une *dérivation* est une séquence de transitions de communications. La *taille* d'une dérivation est définie comme la taille de sa séquence de transitions. On dit qu'un service a *terminé* dans une dérivation donnée, s'il est réduit au processus nul.

3.2.3 Problème de composition de services web

Objectif de la composition

Pour répondre à la requête d'un client $\mathcal{C}$ nous avons souvent besoin d'un nouveau service $\mathcal{T}$ à obtenir comme la composition de certains services disponibles dans la communauté. Nous définissons l'objectif de la composition comme la liste ordonnée des messages que $\mathcal{C}$ et $\mathcal{T}$ devraient échanger. Ainsi l'objectif de la composition est aussi un service qui peut être spécifié à l'aide de la grammaire donnée plus haut.

Médiateur de la composition

Nous exploitons une dérivation pour générer un médiateur de la composition. Les messages envoyés par les services sont dispatchés par le médiateur et peuvent être adaptés avant d'être assignés à un service récepteur. Pour exprimer ces capacités d'adaptation du médiateur, nous définissons la règle de transition ci-dessous :

$$(\mathcal{P},\ \mathcal{E}) \xrightarrow{C} (\mathcal{P},\ \mathcal{E} \cup \{m\})$$

s'il existe un contexte C et $t_1, \ldots, t_n$ dans $\mathcal{E}$ t.q $C[t_1, \ldots, t_n] =_{\mathcal{E}_{XML}} m$

Nous appelons ces transitions des *transitions d'adaptation* Le problème qui nous intéresse est alors de vérifier si un client $\mathcal{C}$ peut être satisfait par une composition de services dans la communauté. Plus formellement le problème est posé comme suit :

Problème de composition de services

Input:	Une communauté de services $\mathcal{S} = \{S_1, \ldots, S_n\}$ Un objectif de la composition $\mathcal{C}$ (spécifié par les requêtes du client)
Output:	Une dérivation depuis l'état initial $(\mathcal{S} \cup \{\mathcal{C}\},\ \emptyset)$ vers un état où $\mathcal{C}$ a terminé et où chaque service dans $\mathcal{S}$ a soit terminé, soit est toujours dans son état initial si une telle dérivation existe, ou le symbole $\perp$ sinon.

En d'autres termes, nous devons vérifier l'existence d'une dérivation (en appliquant les règles de transition) depuis un état initial où le client est mis en parallèle avec la communauté de services et où aucun message n'a encore été envoyé, à un état où toutes les requêtes du client ont été satisfaites ($\mathcal{C}$ a terminé) et où les services de la communauté qui ont été utilisés ont aussi terminé.

3.2.4 Résolution du problème de composition

Théorème 1 *Le problème de composition de services web est NP-complet.*

Résumé de la preuve : Nous réduisons le problème de composition de services à la caractérisation d'une attaque sur un protocole cryptographique construit à partir du client et la communauté

de services disponibles, étant donnée la théorie équationnelle $\mathcal{E}_{XML}$. Pour assurer la terminaison des services impliqués dans l'interaction avec le client, nous devinons dés le départ le sous-ensemble $\{S'_1, \ldots, S'_m\}$ de services qui seront effectivement utilisés. Ensuite nous réduisons le problème de composition à l'atteignabilité d'une configuration $(0, \mathcal{E})$, à partir de la configuration $(\mathcal{C} \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$ avec $\{S'_1, \ldots, S'_m\} \subseteq \{S_1, \ldots, S_n\}$.

Pour tout service $S \cdot 0$ dans $\{\mathcal{C}, S'_1, \ldots, S'_m\}$ nous introduisons une nouvelle constante c_S et nous transformons le service $S \cdot 0$ en un service $\overline{S} = S \cdot SND(c_S) \cdot 0$. Il est alors clair qu'un service S se réduit au service nul si, et seulement si, $\overline{S}$ envoie c_S. En dernier lieu nous rajoutons un service moniteur M à la communauté de services qui teste si toutes les constantes ont été envoyées. On définit donc :

$$M = RCV(c_{\mathcal{C}}) \cdot RCV(c_{S'_1}) \ldots RCV(c_{S'_m}) \cdot SND(secret) \cdot 0$$

Il est clair que M envoie *secret*, si et seulement si, tous les services $\mathcal{C}, S'_1, \ldots, S'_m$ terminent. Ainsi nous avons transformé le problème de l'atteignabilité d'une configuration $(0, \mathcal{E})$ depuis une configuration $(\mathcal{C} \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$, en celui de l'atteignabilité de la configuration $(P, \mathcal{E}')$, avec $secret \in \mathcal{E}'$, depuis la configuration initiale $(M \parallel C \parallel S'_1 \parallel \ldots \parallel S'_m, \emptyset)$. Ce dernier problème est un classique dans le domaine des protocoles cryptographiques, on l'appelle *Problème d'insécurité de protocoles*. Puisque ce problème est connu pour être NP-complet [58], nous pouvons donc conclure. □

Le problème d'insécurité de protocoles correspondant à notre problème de composition peut ensuite être soumis à n'importe quel outil de vérification de protocoles disponible dans la littérature et capable de vérifier une propriété d'atteignabilité. Si le problème admet une solution de composition, nous obtenons une trace de conversation, décrivant comment le médiateur a réussi à satisfaire les demandes du client en appliquant ses compétences d'adaptation sur les messages échangés avec certains services dans la communauté.

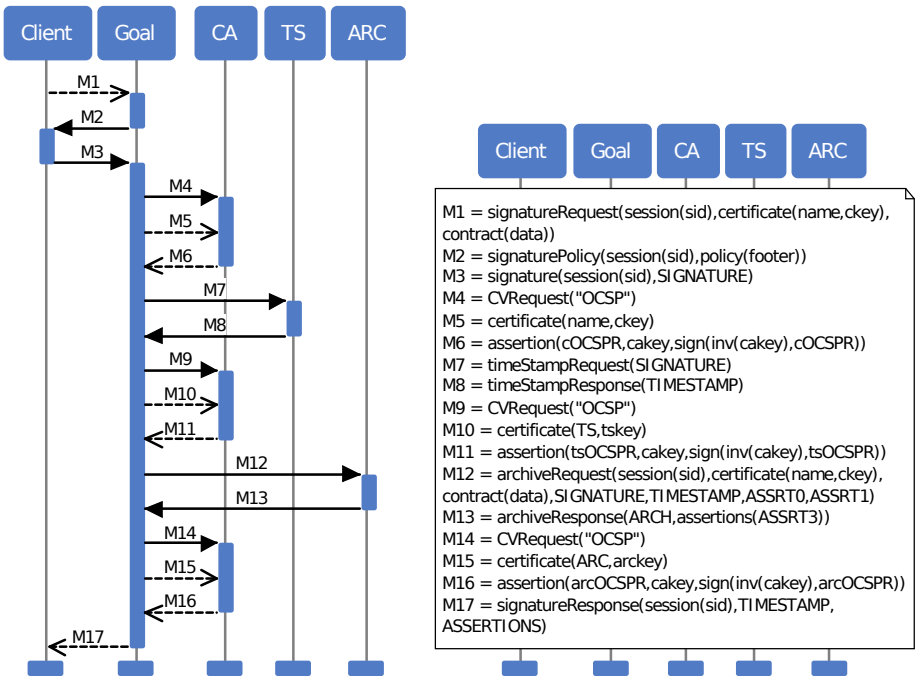
La figure 3.6 illustre la solution pour le problème de composition posé dans l'exemple introductif.

Pour satisfaire les requêtes du *Client* (messages $M1$ and $M3$) le médiateur d'abord invoque l'autorité de certification CA avec les messages $M4$ et $M5$, et obtient une assertion $M6$ prouvant la validité du certificat du *Client*. Ensuite, il invoque le service d'horodatage TS digne de la confiance du *Client* avec le message $M7$ et obtient un horodatage correspondant $M8$. Le médiateur invoque alors de nouveau CA avec les messages $M9$ et ensuite $M10$ pour obtenir une assertion $M11$ prouvant la validité du certificat de TS. Ensuite il invoque le service d'archivage ARC digne lui aussi de la confiance de *Client* avec le message $M12$ et obtient les assertions (contenues dans le message $M13$) prouvant que la signature électronique du *Client* a bel et bien été correctement archivée pour une longue conservation. Enfin, le médiateur invoque CA avec les messages $M14$ et ensuite $M15$ pour obtenir la dernière assertion requise $M16$ qui stipule la validité du certificat de ARC avant de répondre avec succès à la dernière requête du *Client* par le message $M17$.

On remarque que le service de médiateur M est facilement extractible à partir de la trace de la conversation. Cela peut être fait parcourant toutes les étapes de communication dans la trace de la conversation, en mettant l'accent sur ceux impliquant le médiateur et en mettant à jour sa description de service comme suit :

- si l'étape de communication est $S \rightarrow M : t$, rajouter $RCV(t)$ à M ;
- sinon, rajouter $SND(t)$ à M.

D'autre part et par définition du problème de composition, une solution correspondante doit également décrire toutes les étapes d'adaptation qui doivent être effectuées par le médiateur.



(a) Trace (b) Messages

FIG. 3.6 – Solution du problème de composition de l'exemple introductif

Ces étapes sont abstraites dans la trace de la conversation décrit dans la figure 3.6, qui est un résultat typique des outils de vérification de protocoles présents dans la littérature. Par exemple on peut intuitivement voir que le message $M5$ envoyé par le médiateur à CA peut être extrait du message $M1$ qui lui a été auparavant envoyé par *Client*, en prenant son deuxième fils. Nous présentons dans le chapitre 4 une procédure automatisée qui permet de calculer toutes ces étapes d'adaptation à partir d'une trace de conversation similaire à celle illustré par la figure 3.6

3.3 Implantation de l'orchestrateur de services web AVANTSSAR

Pour un service client C et une communauté de services disponibles $\{S_1, \ldots, S_n\}$, le problème de composition revient à construire un service objectif G capable de répondre à toutes les requêtes de C, en utilisant les connaissances initiales fournies pour G et éventuellement des invocations de services dans la communauté. En d'autres termes G doit être tel que $C|G|S_1|\ldots|S_n$ (où $|$ dénote la mise en parallèle) puisse évoluer depuis son état initial vers un état où C et G ont terminé.

Nous considérons aussi une variation du problème où une caractérisation abstraite du service objectif G est donnée à la place de C. Dans ce cas, le problème de composition revient à trouver un service G' qui simule G en utilisant les connaissances initiales initialement fournies pour le premier et éventuellement en invoquant des services disponibles dans la communauté.

3.3.1 Format d'entrée : problèmes de composition de services web en ASLan

Une spécification d'entrée pour un problème de composition est un fichier ASLan où certains mots clefs réservés ont une signification particulière. Un service est défini par la donnée de

l'ensemble de ses connaissances initiales et de son comportement : un système de transitions avec un état initial.

Deux types de spécifications d'entrée sont prises en charge par la plateforme AVANTSSAR. Dans les deux cas, les services disponibles sont définis par leurs systèmes de transitions respectifs.

La première option est de définir un service client dont les requêtes doivent tous être satisfaites. Résoudre le problème de composition revient alors à construire un service (service objectif) qui est capable de répondre à toutes les demandes du client. Pour ce faire, il peut utiliser le services disponibles et toutes les communications du service objectif doivent être reflétées dans le résultat.

La deuxième option consiste à définir partiellement le service objectif, où seule ses communications avec un client hypothétique sont représentées. Le résultat est un nouveau service objectif, qui enrichit le premier avec toutes les communications requises avec la communauté de services disponibles.

Problème de composition de services web en ASLan

Pour spécifier un problème de composition, nous avons besoin de distinguer les parties clefs : le service objectif (partiellement décrit) ou le services client (dont les requêtes doivent être satisfaites).

L'état du service objectif est décrit par le fait :

$$\text{state\_OrchestrationGoal}(<\text{Liste de paramètres}>)\,;$$

D'une manière similaire, celui du service client est décrit par le fait :

$$\text{state\_OrchestrationClient}(<\text{Liste de paramètres}>).$$

Un exemple de transition pour un client est donnée dans l'exemple 4. Le communauté des services disponibles mise à part, un problème de composition de service peut aussi être contraint par une formule LTL qui doit être satisfaite par le service composé calculé. A titre d'exemple, pour calculer une composition où un service (*Emergency*) doit rester dans son état initial tandis que client atteint l'état state_OrchestrationClient(A,99), (pour une certaine valeur de A), nous définissons la forume LTL suivante :

```
goal orchestrationConstraint(A) :=
       and(G(state_emergency(0)), F(state_orchestrationClient(A,99)))
```

Le mot clef orchestrationConstraint est réservé pour exprimer le fait que cette contrainte doit être interprété par l'orchestrateur. Ceci contraste avec les autres buts de sécurité qui doivent être interprété par le validateur.

Une autre manière de spécifier un problème de composition est de définir un état d'attaque spécial, en utilisant le mot clef orchestrationFinalState. Le problème de composition est alors résolu, si un tel état d'attaque est atteint. L'exemple 4 utilise cette manière de procéder.

D'un autre côté les connaissances initiales du service objectif peuvent affecter la résolution du problème de composition. En effet les capacités du service objectif dépendent étroitement de ses connaissances : les certificats dont il dispose, et les fonctions ou primitives cryptographiques qu'il peut utiliser.

Nous utilisons le prédicat state_OrchestrationGoal pour spécifier les connaissances initiales du services objectif. Ces connaissances sont représentées par les arguments du prédicat. La même chose vaut pour le prédicat state_OrchestrationClient qui permet de préciser les connaissances initiales du client.

Nous rappelons que les émissions et réceptions de messages sont spécifiées à l'aide des faits iknows. Placé dans le CG d'une transition, le iknows représente une réception, tandis qu'il représente une émission s'il est placé dans son CD.

Problème de composition avec un service client Dans le cas où un le problème de composition est posé avec un service client, il faut fournir les données suivantes :

- les spécification ASLan des services disponibles ;
- la spécification ASLan du service client ;
- les connaissances initiales du service objectif ;
- attack_state orchestrationFinalState et/ou goal orchestrationConstraint.

Par ailleurs, toutes les communications définies dans la spécification du services client doivent être alors exclusivement menées avec le services objectif.

Pour définir le problème de composition comme un problème d'atteignabilité, nous identifions la dernière transition qui doit être activée pour satisfaire le client et nous rajoutons à son CD *iknows(constante_fraiche)*, où *constante_fraiche* est une constante qui n'est pas utilisée ailleurs dans la spécification. Ensuite nous définissons un état d'attaque nommé *orchestrationFinalState*, comme l'état contenant le fait unique *iknows(constante_fraiche)* (voir exemple 4).

Exemple 4

```
        ....
        step step_5(Ag, Stp, Dummy_A):=
        state_OrchestrationClient(Ag, 5,Dummy_A).
        iknows(a)
    =>
        state_OrchestrationClient(Ag, 6, a).
        iknows(finish)

        ...
    attack_state orchestrationFinalState(Dummy) :=
    iknows(finish)
```

Problème de composition avec un services objectif partiellement spécifié Dans le cas où le problème de composition est posé avec un service objectif partiellement spécifié, il faut fournir les données suivantes :

- la spécification ASLan des services disponibles ;
- la spécification ASLan du service objectif, concernant uniquement ses communications avec le client ;
- les connaissances initiales du service objectif.

La spécification d'une contrainte d'orchestration du style goal orchestrationConstraint est optionnelle et un éventuel état d'attaque attack_state orchestrationFinalState sera ignoré.

Par défaut la sortie de l'orchestrateur comprendra aussi une spécification d'un client hypothétique (correspondant à la communication avec le service objectif partiellement spécifié). Pour la supprimer de la sortie, il suffit de rajouter cette ligne dans la spécification ASLan d'entrée :

```
% @orchestrator(no_client)
```

Cette méta-information sera interprétée par l'outil et la spécification du client hypothétique n'apparaîtra pas dans la sortie. Ceci peut être utile dans certains cas pour faciliter la validation du résultat de la composition.

3.3.2 Format de sortie : ASLan

Le format de sortie pour l'orchestrateur est le langage ASLan. Ceci permet de procéder à la validation du résultat par le validateur AVANTSSAR. Étant donné un fichier ASLan d'entrée (obéissant aux règles décrites dans la section 3.3.1), l'orchestrateur l'enrichit par les nouvelles transitions concernant :

- un service *client* hypothétique quand il est généré par l'outil (et non pas fourni par le modeleur) ;
- un service *objectif* permettant de répondre correctement aux requêtes du client.

Les transitions introduites représentent les états des services client et objectif respectivement par state_OrchestrationClient(<Liste de paramètres>) et state_OrchestrationGoal(<Liste de paramètres>).

3.3.3 Aperçu de l'architecture de l'orchestrateur AVANTSSAR

L'orchestrateur prend en entrée un fichier ASLan contenant une spécification des services disponibles et soit une spécification du client, soit une spécification partielle de l'objectif. Il produit en sortie un fichier ASLan comprenant la spécification des services disponibles, une spécification complète du service objectif, et éventuellement une spécification du service client (un client hypothétique, si une spécification particulière n'a pas été fournie en entrée par le modeleur).

La figure 3.7 illustre l'architecture de l'orchestrateur. Elle est logiquement divisée en deux parties : le générateur (Generator) et l'intégrateur (Integrator).

La spécification d'entrée est d'abord envoyée vers `Client2Client`, un outil qui prépare l'entrée pour CL-Atse. Si `Client2Client` détecte que le format de l'entrée ne correspond pas à un problème de composition avec service client fourni, il termine avec un code d'exécution différent de zéro et ensuite l'outil `Goal2Client` est appelé avec la même spécification en entrée. `Goal2Client` prépare l'entrée pour CL-Atse quand il s'agit d'un problème d'orchestration avec services objectif partiellement spécifié. Il génère aussi certaines données pour aider l'outil `Trace2ASLan` dans l'intégration du service objectif obtenu dans un fichier ASLan. Un autre rôle de `Client2Client` et de `Goal2Client` est détecter et désactiver les buts de sécurité éventuellement fournies par le modeleur, afin de ne pas gêner la recherche d'orchestration. Le résultat obtenu par CL-Atse et la spécification ASLan initialement fournie par le modeleur sont alors donnés à `Trace2ASLan`. Enfin `Trace2ASLan` compile et intègre le service objectif résultant à la spécification ASLan initiale. `Trace2ASLan` replace alors les buts de sécurités initialement fournis par le modeleur afin de préparer la phase de validation de la spécification globale.

3.3.4 Composants de l'orchestrateur AVANTSSAR

`Client2Client`

Le rôle de ce composant est de préparer l'entrée pour CL-Atse, dans le cas où il s'agit d'un problème de composition avec un service client fourni. Il vérifie l'adéquation de l'entrée par rapport aux exigences requises données dans les sections 3.3.1 et 3.1.3 et initialise les connaissances de l'intrus (jouant ici plutôt un rôle de médiateur) par les connaissances initiales de l'objectif.

`Goal2Client`

Le rôle de ce composant est de préparer l'entrée pour CL-Atse, dans le cas où il s'agit d'un problème de composition avec un services objectif partiellement spécifié. Plus exactement il :

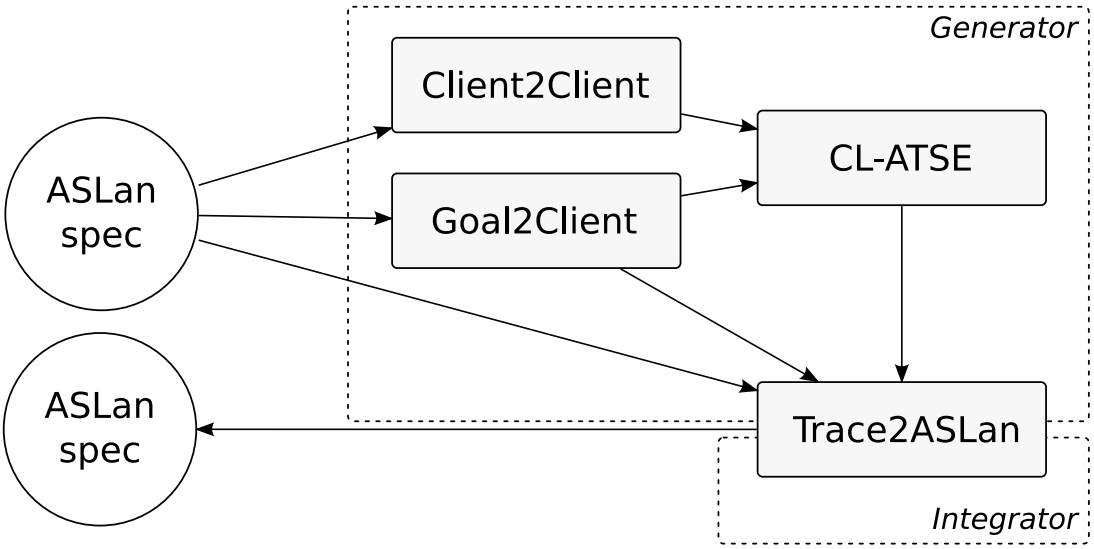


FIG. 3.7 – Tool architecture

1. vérifie l'adéquation de l'entrée par rapport aux exigences requises données dans les sections 3.3.1 et 3.1.3 ;
2. génère une spécification ASLan d'un services client hypothétique à partir de la spécification partielle donnée pour le service objectif, et remplace la dernière par la première dans l'entrée ;
3. initialise les connaissances de l'intrus par celles fournies pour le services objectif ;
4. génère une constante fraîche (qui permettra d'encoder l'état d'attaque correspondant à l'atteignabilité d'un état satisfaisant pour le client) et le nom de l'agent exécutant le nouveau service client généré
5. mémorise la séquence ordonnée des transitions du service objectif partiellement spécifié et la table de renommage des noms de variables auquel il aura procédé pendant la transformation de l'entrée dans deux fichiers auxiliaires. L'exécution de cette étape doit être confirmée par des options à rajouter sur la ligne de commande. Elle permet de garder un lien (nécessaire pour le bon fonctionnement de `Trace2ASLan`) avec la spécification initiale.

Nous donnons un exemple de la transformation d'un service objectif partiellement spécifié à l'aide d'une seule transition vers le service client hypothétique correspondant. Supposons que nous avons la spécification suivante du service objectif comme entrée :

Exemple 5

```
step step_2 (GOAL,D_I1,D_I2, D_Set_36,SID,I1,I2) :=
 state_OrchestrationGoal(GOAL,1,D_I1, D_I2,D_Set_36,SID).
 iknows(pair(I1,I2))
 =>
 state_OrchestrationGoal(GOAL,2,I1,I2, D_Set_36,SID).
 iknows(apply(times,pair(apply(plus,pair(I1,I2)),
    apply(plus,pair(I1,I2))))).
 secret(apply(times,pair(I1,I2)),sec_prop,D_Set_36).
 contains(GOAL,D_Set_36)
```

`Goal2Client` retourne alors un fichier ASLan où la spécification ci-dessus est remplacé par le service client correspondant.

Exemple 6

```
step step_0001(ClientAgent_0000):=
      state_OrchestrationClient(ClientAgent_0000, 1).
      iknows(start)
  =>
      state_OrchestrationClient(ClientAgent_0000, 2).
      iknows(pair(var_Renamed_I1_0000_0000, var_Renamed_I2_0000_0000))

step step_0000(ClientAgent_0000):=
      state_OrchestrationClient(ClientAgent_0000, 2).
      iknows(apply(times, pair(apply(plus,
         pair(var_Renamed_I1_0000_0000, var_Renamed_I2_0000_0000)),
         apply(plus, pair(var_Renamed_I1_0000_0000,
         var_Renamed_I2_0000_0000)))))
  =>
      state_OrchestrationClient(ClientAgent_0000, 3).
      iknows(finish_0000)
```

Dans sa première transition step_0001, le client envoie un message qui pourrait être reçu par le service objectif de l'exemple 5. Dans sa deuxième transition le client s'attend à recevoir un message qui correspond à celui renvoyé par le services objectif. Dans cette spécification les constantes var_Renamed_I1_0000_0000 et var_Renamed_I2_0000_0000 correspondent respectivement aux variables I1 et I2. Grâce à l'hypothèse d'atomicité sur les variables utilisées dans la spécification partielle du service objectif, cette transformation des variables en constantes permet d'observer une solution correcte au problème de composition. Quand le client hypothétique est construit seulement les prédicats relatifs à la communication sont considérés. Ceux qui expriment des propriétés de sécurité et qui par conséquent ne relève pas de la composition sont donc ignorés, par exemple : secret(apply(times,pair(I1,I2)),sec_prop,D_Set_36) dans l'exemple 5. Ces prédicats de sécurité sont par la suite réintégrés dans le fichier ASLan de sortie, avant que ce dernier ne soit envoyé au vérificateur AVANTSSAR.

CL-Atse

C'est le composant clef de la partie générateur de l'orchestrateur AVANTSSAR. CL-Atse est un solveur de contraintes pour les protocoles de sécurité et les services. Il s'appuie sur une technique de déductibilité pour la résolution de systèmes de contraintes bien formés. Pour cela, CL-Atse parcourt tous les entrelacements d'exécution possibles d'un ensemble de services en représentant les familles de traces obtenues jusque là par l'ensemble des contraintes portant sur les connaissances de l'intrus, sur les valeurs des variables ou les ensembles utilisés. Chaque exécution d'une étape de communication d'un service donné rajoute de nouvelles contraintes sur les états de l'intrus et de l'environnement, réduisant progressivement le système de contraintes à une forme normalisée pour laquelle la satisfiabilité est facilement décidable, et permettant de décider si une propriété de sécurité a été violée jusqu'à cette étape de communication.

Étant donné une spécification ASLan correspondant à un problème de composition, cet outil vérifie l'atteignabilité d'un état d'attaque spécial, correspondant à un état final satisfaisant du client. Le résultat obtenu est une trace : une séquence d'événements de communication (envoi/réception) entre le médiateur et les services disponibles qui peuvent être utilisés dans la composition.

`Trace2ASLan`

En cas de succès l'orchestrateur produit une conversation entre la communauté de services disponibles, le client et le médiateur qui montre l'existence d'une orchestration satisfaisant le client. Cependant cette trace ne fournit que les activités de communication du médiateur. Par exemple lorsque le médiateur doit adapter certains messages reçus avant de les renvoyer à un service disponible ou au client, ces étapes d'adaptation ne sont pas explicites dans la trace.

Exemple 7 *Considérons la séquence d'activités de communication suivante :*

$$RCV(scrypt(k, scrypt(k', m))) \cdot RCV(pair(k, k')) \cdot SND(m)$$

Pour pouvoir émettre m dans sa dernière étape, une implémentation de s doit déchiffrer deux fois son message reçu à la première étape, respectivement par k et k' reçues à la deuxième. Cet ensemble de calculs ordonnés n'apparait pas dans le strand s et en présence de primitives de chiffrement, sa recherche n'est pas simple.

Pour obtenir une description opérationnelle exécutable du médiateur, nous avons développé un algorithme de compilation prudente des traces de conversation, qui spécifie les communications entre différents services web. En fait, deux objectifs principaux sont à considérer après la découverte d'un médiateur solvant un problème de composition de services : (i) générer une distribution exécutable du médiateur, et (ii) de procéder à la validation des services impliqués dans l'orchestration, y compris le médiateur, vis à vis des propriétés de sécurité requises par le modeleur.

Il est nécessaire de considérer (ii) pour avoir des garanties de sécurité concernant le médiateur (plus précisément dans le contexte de la composition) avant de générer son code exécutable. En effet l'entrée de l'orchestrateur exprime un double problème : un problème de composition, pour lequel la trace éventuellement retournée par CL-Atse est une solution (partielle) et un problème de validation qui devrait prendre en compte la présence du médiateur. Afin de permettre l'utilisation de la plateforme de validation AVANTSSAR, nous devons d'abord traduire la trace décrivant le médiateur en une spécification ASLan, puis l'intégrer à celle initialement fournie à l'orchestrateur. Cette traduction est détaillée dans le chapitre 4. Elle repose sur un algorithme de compilation prudente de traces de conversation, qui spécifie les communications entre différents services web.

3.3.5 Boucle de sécurité

Lors de la spécification d'un problème d'orchestration, un modeleur peut spécifier des propriétés de sécurité devant être automatiquement validée pour l'orchestration obtenue. Alternativement, une fois une orchestration obtenue, le modeleur peut rajouter des propriétés de sécurité dans la spécification et essayer de les valider en utilisant les outils de validation de la plateforme AVANTSSAR. En effet l'orchestrateur ne traite pas la validation et il est possible que la composition qu'il génère ne satisfasse pas les propriétés de sécurité souhaitées. Dans ce cas, l'orchestrateur fournit une solution alternative au problème, s'il en existe une. Dans cette section nous décrivons cette boucle de sécurité.

Le noyau de l'orchestrateur, CL-Atse, qui produit une trace à partir de laquelle le médiateur est recréé en utilisant l'outil `Trace2ASLan`, est capable de sauvegarder sa position courante dans l'arbre de recherche. Quand une trace est trouvée, l'outil sauvegarde son état actuel dans un fichier. L'idée est alors de reprendre la recherche à partir de la position sauvegardée, quand la solution courante s'avère être vulnérable par rapport aux propriétés de sécurité souhaitées par le modeleur. De cette manière CL-Atse est capable de parcourir tout l'arbre de recherche, et

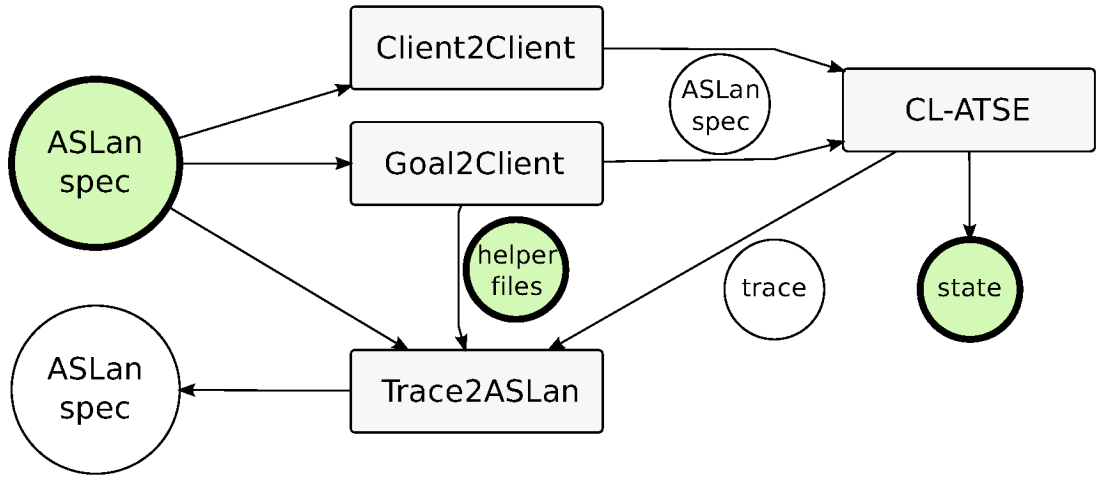


FIG. 3.8 – Données nécessaires pour lancer la recherche d'une nouvelle composition

produit éventuellement un ensemble de traces différentes. La seule restriction de l'approche est son incomplétude : CL-Atse retourne une composition quand il en existe une, mais le retour de toutes les traces possibles n'est pas garanti.

La figure 3.8 illustre les données nécessaires pour relancer la recherche d'une composition pour un problème de composition donné.

- la spécification ASLan du problème de composition ;
- l'état courant de l'outil CL-Atse ;
- dans le cas où un service objectif partiellement spécifié est fourni, les fichiers d'assistance pour `Trace2ASLan`, produit par l'outil `Goal2Client`.

Avant de commencer la génération de l'orchestration, l'outil alloue un identifiant unique (désigné par *sessionID*) pour identifier la tâche de composition. Cet identifiant est également utilisé pour nommer les fichiers qui contiennent des données mentionnées ci-dessus nécessaires pour restaurer la recherche de compositions alternatives. Le *sessionID* est aussi retourné par l'outil avec la solution trouvée. Ainsi le modeleur peut l'utiliser pour commander une solution alternative (voir la figure 3.9).

Comme les fichiers nécessaires à la reprise de la génération des compositions possibles peuvent être récupéré sur la base du *sessionID*, l'outil peut facilement continuer son exécution. La chaîne de composition est alors illustrée dans la figure 3.10 (les données récupérées par le *sessionID* sont surlignées).

Ainsi, nous pouvons ré-invoquer l'orchestrateur pour obtenir une composition alternative, dans le cas où la composition courante ne satisfait pas les propriétés de sécurité requises par le modeleur (ou pour n'importe quelle autre raison).

3.3.6 Un simple exemple de problème de composition et sa solution

Dans cette section nous présentons un exemple de problème de composition de services et nous décrivons sa résolution grâce aux outils de la plateforme AVANTSSAR. Supposons que nous voulons créer un service qui accepte deux chiffres et renvoie le carré de leur somme. Le service objectif que nous recherchons n'est pas censé effectuer les opérations de multiplication et d'addition, mais plutôt compter sur les services disponibles pour cela : un service additionneur (*Adder*) et un service multiplicateur (*Multiplier*).

D'abord nous devons spécifier les deux services disponibles.

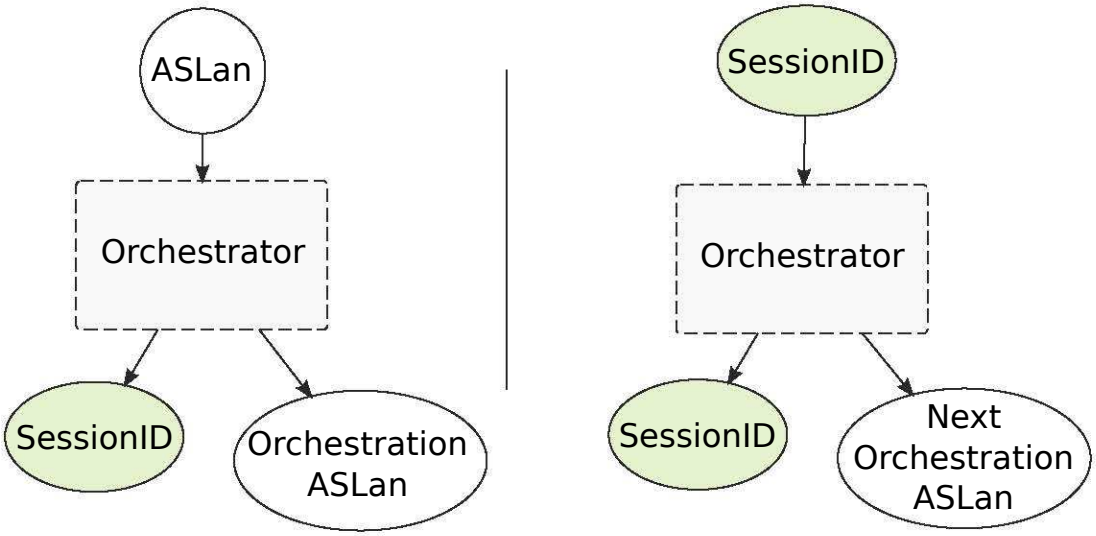


FIG. 3.9 – Invocation itérative de l'orchestrateur

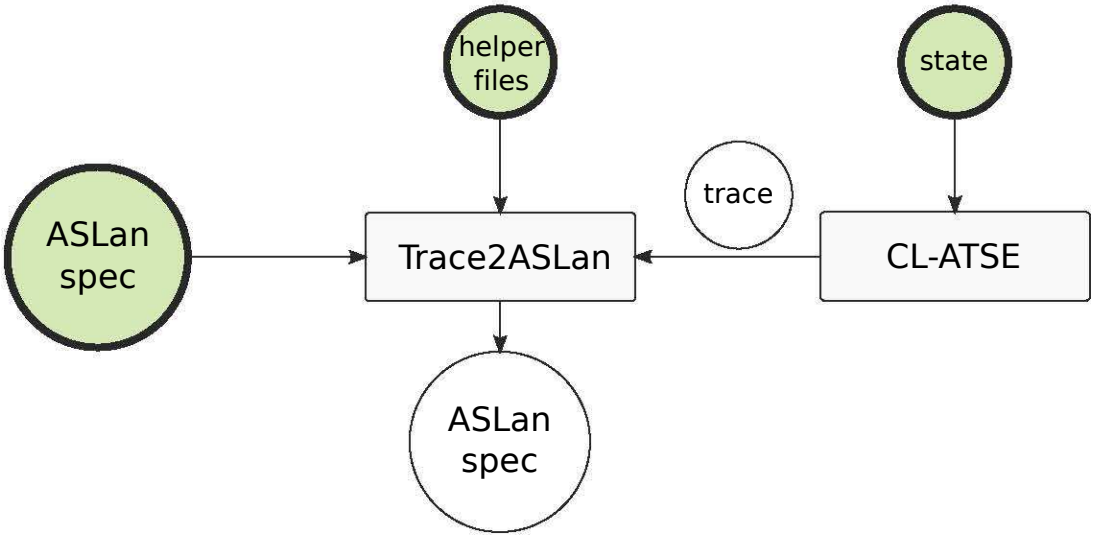


FIG. 3.10 – Reprise de la génération des compositions

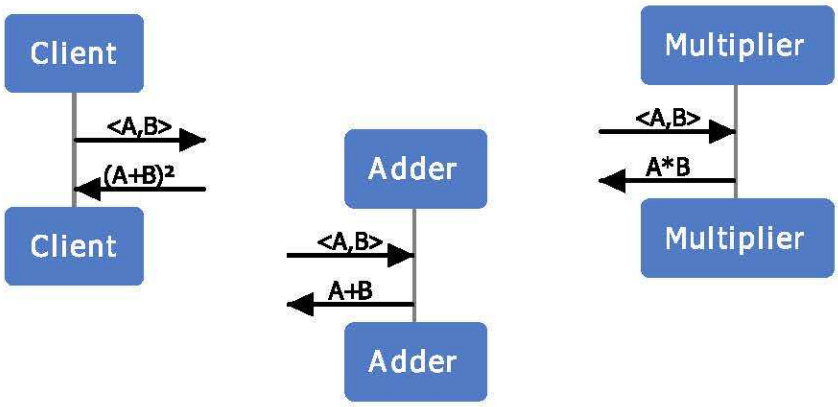


FIG. 3.11 – Services disponibles et service client

Communauté des services disponibles :

Adder : reçoit deux valeurs a et b, et renvoie leur somme $a + b$.

```
step step_0 (A,SID,I,J) :=
 state_Adder(A,1,dummy_msg,dummy_msg,SID).
 iknows(pair(I,J))
 =>
 state_Adder(A,2,I,J,SID).
 iknows(apply(plus,pair(I,J)))
```

Son état initial est le suivant :

```
state_Adder(a,1,dummy_msg,dummy_msg,3)
```

Multiplier : reçoit deux valeurs a et b, et renvoie leur produit $a * b$.

```
step step_1 (M,SID,I,J) :=
 state_Multiplier(M,1,dummy_msg,dummy_msg,SID).
 iknows(pair(I,J))
 =>
 state_Multiplier(M,2,I,J,SID).
 iknows(apply(times,pair(I,J)))
```

Son état initial est le suivant :

```
state_Multiplier(m,1,dummy_msg,dummy_msg,4)
```

Ensuite nous devons choisir entre les deux manières possibles de spécifier le problème : donner une spécification partielle du service objectif désiré (*Goal*) ou celle d'un service client (*Client*) dont les requêtes sont à satisfaire.

Formulation des problèmes de composition

Client : envoie (dispose de) a et b, et reçoit (souhaite) $(a + b) * (a + b)$.

Goal : reçoit a et b ; retourne $(a + b) * (a + b)$.

Spécification d'un service client à satisfaire

Si le modeleur choisit de spécifier le comportement du client pour définir un problème d'orchestration, la spécification du client est la suivante :

```
step step_2 (C,SID,I1,I2) :=
 state_OrchestrationClient(C,1,dummy_nonce, dummy_nonce,SID).
=[exists I2,I1]=>
 state_OrchestrationClient(C,2,I1,I2, SID).
 iknows(pair(I1,I2)).
 keepsecret(apply(times,pair(I1,I2)), secret_value_id)

step step_3 (C,I1,I2,SID) :=
 state_OrchestrationClient(C,2,I1,I2, SID).
 iknows(apply(times,pair(apply(plus,pair(I1,I2)),
    apply(plus,pair(I1,I2)))))
 =>
 state_OrchestrationClient(C,3,I1,I2, SID).
 iknows(end_orchestration)
```

Son état initial est alors :

```
 state_OrchestrationClient(c,1,dummy_nonce, dummy_nonce,5).
```

Le fait iknows(end_orchestration) dans la dernière transition du client marque son état final satisfaisant (voir l'explication dans le paragraphe correspondant de la section 3.3.1 et l'exemple 4). L'état d'attaque correspondant est le suivant :

```
attack_state orchestrationFinalState (ASGoal) :=
 iknows(end_orchestration)
```

Le fait keepsecret(apply(times,pair(I1,I2)),secret_value_id) est donnée comme exemple d'une propriété de sécurité à vérifier. Cette propriété ne fait pas partie des contraintes de composition, mais se réfère plutôt à un problème de validation à résoudre par le validateur AVANTSSAR, une fois une composition trouvée. L'état d'attaque correspondant est le suivant :

```
attack_state to_validate(MGoal) :=
 iknows(MGoal).
 keepsecret(MGoal,secret_value_id)
```

Conformément aux exigences données dans la section 3.3.1, nous devons aussi spécifier les connaissances initiales du service objectif :

```
 state_OrchestrationGoal(g)
```

Solution Le résultat est une spécification complète (exécutable) du service objectif qui est intégrée dans le fichier d'entrée, contenant les spécifications formelles du client et de la communauté de services disponibles[2] :

```
...
step step_OG_1 (OGoal,SID,G,Dummy_I1_n3,I1_n3,Dummy_I2_n3,I2_n3,
    Dummy_X4,Dummy_X7,Dummy_Eo) :=
 state_OGoal (OGoal,SID,1,G,Dummy_I1_n3,Dummy_I2_n3,
     Dummy_X4,Dummy_X7,Dummy_Eo).
 iknows(pair(I1_n3,I2_n3))
```

[2]Nous avons remplacé OrchestrationGoal par OGoal et End_orchestration par Eo pour un meilleur affichage

```
 =>
 state_OGoal (OGoal,SID,3,G,I1_n3,I2_n3, Dummy_X4,Dummy_X7,Dummy_Eo).
 iknows(pair(I1_n3,I2_n3))

step step_OG_2 (OGoal,SID,G,I1_n3,I2_n3, Dummy_X4,X4,Dummy_X7,Dummy_Eo)
   :=
 state_OGoal (OGoal,SID,3,G,I1_n3,I2_n3, Dummy_X4,Dummy_X7,Dummy_Eo).
 iknows(X4)
 =>
 state_OGoal (OGoal,SID,5,G,I1_n3,I2_n3, X4,Dummy_X7,Dummy_Eo).
 iknows(pair(X4,X4))

step step_OG_3 (OGoal,SID,G,I1_n3,I2_n3, X4,Dummy_X7,X7,Dummy_Eo) :=
 state_OGoal (OGoal,SID,5,G,I1_n3,I2_n3, X4,Dummy_X7,Dummy_Eo).
 iknows(X7)
 =>
 state_OGoal (OGoal,SID,7,G,I1_n3,I2_n3, X4,X7,Dummy_Eo).
 iknows(X7)

step step_OG_4(OGoal,SID,G,I1_n3,I2_n3, X4,X7,Dummy_Eo,Eo) :=
 state_OGoal (OGoal,SID,7,G,I1_n3,I2_n3, X4,X7,Dummy_Eo).
 iknows(Eo)
 =>
 state_OGoal (OGoal,SID,8,G,I1_n3,I2_n3,X4,X7,Eo)
...
```

Spécification partielle d'un service objectif à compléter

L'autre option est de spécifier partiellement un service objectif, c'est à dire spécifier le sous-ensemble connu de ses communications avec son client :

```
 step step_2 (GOAL,SID,I1,I2) :=
   state_OrchestrationGoal(GOAL,1, dummy_msg,dummy_msg,SID).
   iknows(pair(I1,I2))
 =>
   state_OrchestrationGoal(GOAL,2,I1,I2, SID).
   iknows(apply(times,pair(apply(plus,pair(I1,I2)),
      apply(plus,pair(I1,I2))))).
   keepsecret(apply(times,pair(I1,I2)),secret_value_id)
```

Ici, nous avons également ajouté une propriété de sécurité qui doit être vérifiée par le validateur. L'état d'attaque correspondant est le suivant :

```
 section goals:

 attack_state to_validate (MGoal,ASGoal) :=
 iknows(MGoal).
 keepsecret(MGoal,secret_value_id).
```

L'état initial du service objectif partiellement spécifié est le suivant :

```
  state_OrchestrationGoal(g,1,dummy_msg,dummy_msg,5)
```

Solution Nous illustrons ci-dessous (et en partie) la solution retournée par l'orchestrateur, contenant la spécification complétée du service objectif et une spécification de son client hypothétique[3] :

```
...
step step_OG_1(OGoal,SID,G,X_Int1,Dummy_msg,X_Int5,
   Dummy_I1,I1,Dummy_I2,I2,Dummy_X7,Dummy_X10) :=
 state_OGoal(OGoal,SID,1,G,X_Int1,Dummy_msg,X_Int5,
    Dummy_I1,Dummy_I2,Dummy_X7,Dummy_X10).
 iknows(pair(I1,I2))
 =>
 state_OGoal(OGoal,SID,3,G,X_Int1,Dummy_msg,X_Int5,I1,I2,
    Dummy_X7,Dummy_X10).
 iknows(pair(I1,I2))

step step_OG_2(OGoal,SID,G,X_Int1,Dummy_msg,X_Int5,I1,I2,
   Dummy_X7,X7,Dummy_X10) :=
 state_OGoal(OGoal,SID,3,G,X_Int1,Dummy_msg,X_Int5,I1,I2,
    Dummy_X7,Dummy_X10).
 iknows(X7)
 =>
 state_OGoal(OGoal,SID,5,G,X_Int1,Dummy_msg,X_Int5,I1,I2, X7,Dummy_X10).
 iknows(pair(X7,X7))

step step_OG_3(OGoal,SID,G,X_Int1,Dummy_msg,X_Int5,I1,I2,
   X7,Dummy_X10,X10) :=
 state_OGoal(OGoal,SID,5,G,X_Int1,Dummy_msg,X_Int5,I1,I2, X7,Dummy_X10).
 iknows(X10)
 =>
 state_OGoal(OGoal,SID,7,G,X_Int1,Dummy_msg,X_Int5,I1,I2, X7,X10).
 iknows(X10)

step step_0001(ClientAgent_0000,IID_0000) :=
 state_OrchestrationClient(ClientAgent_0000,IID_0000,1)
 =>
 state_OrchestrationClient(ClientAgent_0000,IID_0000,2).
 iknows(pair(var_Renamed_I1_0000_0000, var_Renamed_I2_0000_0000))

step step_0000(ClientAgent_0000,IID_0000) :=
 state_OrchestrationClient(ClientAgent_0000,IID_0000,2).
 iknows(apply(times,pair(apply(plus,pair(
    var_Renamed_I1_0000_0000,var_Renamed_I2_0000_0000)),
    apply(plus,pair(var_Renamed_I1_0000_0000,
    var_Renamed_I2_0000_0000)))))
 =>
 state_OrchestrationClient(ClientAgent_0000,IID_0000,3).
 iknows(finish_0000)
...
```

[3]Nous avons remplacé OrchestrationGoal par OGoal dans le résultat pour un meilleur affichage

3.4 Discussion

Nous justifions notre approche par le fait que les spécifications standards sont traduisibles dans notre modèle. Dans la suite nous décrivons d'abord les standards reconnus pour les services web, ensuite nous donnons les pistes permettant une traduction automatisée depuis les fichiers de spécifications standards vers le langage ASLan. Enfin nous concluons par un résumé de notre approche pour la composition de services web et nous proposons des améliorations prévues pour des travaux futurs.

3.4.1 Les standards pour les services web

L'interface d'un service Web est décrite d'une manière standard, à l'aide du langage *WSDL* [67]. Cette description est structurée en ports, chacun proposant une série d'opérations disponibles. Une opération est alors définie par les patrons de ses message d'entrée, de sortie conventionnelle et erronée. Ces patrons sont généralement décrits en utilisant le langage *XSD* [59] et reflètent les structures des messages *XML* correspondant. Des contraintes de sécurité peuvent alors être définie par-dessus la description de l'interface du service en utilisant les annotations spécialement définies dans *WS-SecurityPolicy* [54]. Ces annotations interviennent à différents niveaux dans le fichier *WSDL*, liant les niveaux dans lesquels elles apparaissent aux contraintes de sécurité qu'elles transportent. Ces niveaux vont de celui du message, où un exemple typique est le besoin de signer numériquement ou de chiffrer une partie des message reçu et/ou envoyés par une opération du service, jusqu'au niveau du service en entier, tel que : le besoin d'utiliser le protcole *SSL* pour interagir avec le service. L'utilisation de *XSD* pour la description des messages permet l'utilisation du langage *XPATH* [66] pour décrire les parties des messages concernées par les contraintes de sécurité, à l'aide de requêtes simples.

Nous mettons dans ce travail l'accent sur les services basés *SOAP* [33] (par opposition aux services dits RESTful). Ces services se basent sur le protocole de communciation *SOAP* qui encapsule les messages décrites dans leurs spécifications *WSDL*. Après une analyse automatisée, nous pouvons recueillir depuis les spécifications standards, les différents patrons des messages réellement échangés par les services correspondant (encapsulation *SOAP* comprise).

3.4.2 Des standards vers ASLan

Les interfaces des services peuvent être représentées par le langage ASLan. Comme décrit précédemment, ce langage supporte la représentation de messages reçus et émis, dans des étapes correspondant aux opération dans le monde des services, et soumis à des protections cryptographiques (les effets des annotations WS-SP). Dans ce travail nous nous sommes aussi intéressés à des services avec état (*Stateful*). A l'opposé des services *stateless*, ces processus sauvegardent leurs états internes et peuvent donc coordonner plusieurs traitements complexes. Dans les solutions de composition au sein de la SOA nous retrouvons cette notion de *processus métier*, décrite par exemple par le langage BPEL [52]. La théorie $\mathcal{E}_{XML}$ permet de représenter les messages échangés entre services, avec leur encapsulation SOAP et les traitements cryptographiques résultants des annotations WS-SP. La notion de strands permet de capturer la famille des processus séquentiels. Ces processus sont des médiateurs dans les orchestrations comme celles présentées dans ce chapitre.

3.5 Conclusion

Pour le domaine des services web sécurisés, nous avons proposé un outil pour la composition automatique basée sur leurs politiques de sécurité. Étant donné une communauté de services et un service objectif, nous réduisons le problème de l'existence d'un service médiateur permettant de simuler le service objectif en un problème de sécurité, où le médiateur intercepte, adapte et envoie des messages depuis et vers la communauté de services et un service client jusqu'à satisfaire les requêtes de ce dernier. Ainsi, dans le cadre de la composition de services, le médiateur joue le rôle jouée par un attaquant lors de la validation de protocoles.

En cas de succès l'outil produit une conversation entre la communauté de services disponibles, le client et le médiateur qui montre l'existence d'une orchestration satisfaisant le client. Cependant cette conversation ne fournit que les activités de communication du médiateur. Par exemple lorsque le médiateur doit adapter certains messages reçus avant de les renvoyer à un service disponible ou au client, ces étapes d'adaptation ne sont pas explicites dans la trace. Dans le chapitre suivant nous décrivons comment toutes ces activités d'adaptation sont extraites de manière automatique (et efficace) depuis la conversation.

Chapitre 4

Implémentations prudentes des services web

Sommaire

4.1 Introduction à l'implémentation des services web

L'orchestrateur AVANTSSAR produit des scénarios de composition sous la forme de traces de communication où l'intrus i représente le médiateur. A titre d'exemple, la trace de communication qui résout le problème de composition présenté dans la section 3.1.4 est illustrée dans la figure 4.1 (et où le médiateur est nommé *Goal*). Cependant cette trace qui illustre les communications entre la communauté de services disponibles, le client, et le médiateur est insuffisante pour implémenter ce dernier.

En effet il faut expliciter comment le médiateur construit les messages à partir de ses connaissances et quels sont les vérifications ou tests qu'il doit faire sur les messages reçus. Ce problème

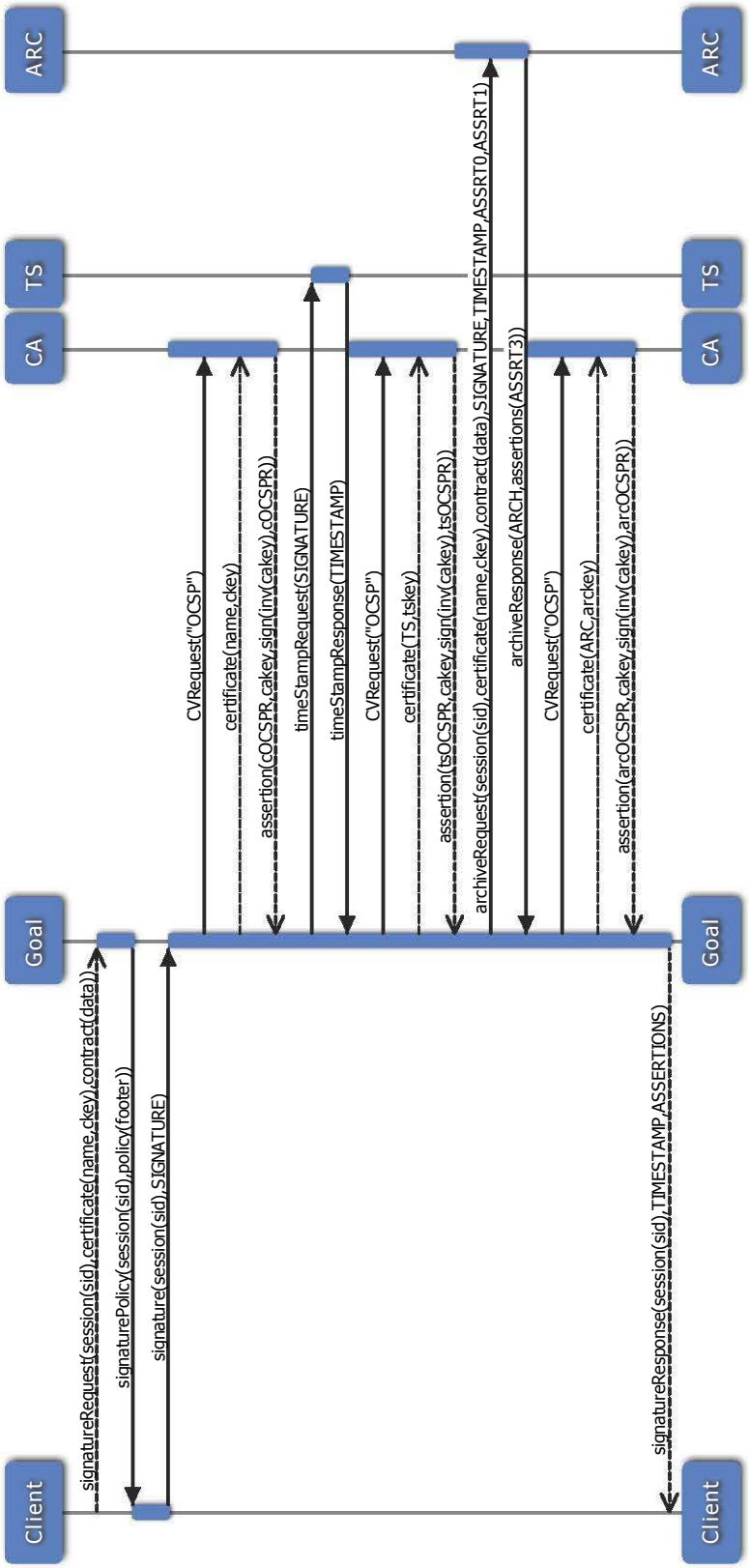


FIG. 4.1 – Solution du problème de composition de la section 3.1.4

se retrouve dans le domaine de la compilation des protocoles cryptographiques [31]. L'algorithme que nous proposons dans ce chapitre permet de construire un service médiateur exécutable

qui effectue le maximum de tests possibles sur les messages reçus. L'implantation de cet algorithme peut produire d'une part une spécification ASLan du médiateur qui peut être validée dans la plate-forme AVANTSSAR et d'autre part une servlet Java déployable dans des serveurs d'application.

Terminologie adoptée : Une conversation est une suite ordonnée de triplets (A, B, M) notée $A \rightarrow B : M$, indiquant que le service A envoie au service B un message M. Dans le contexte des protocoles cryptographiques, cela correspond à une notation *Alice et Bob* $A\&B$. L'orchestrateur AVANTSSAR produit une conversation dont la projection sur un agent ou un service S décrit les activités de communication concernant ce service. Cette projection est construite à partir de la sous-suite de la conversation qui contient les triplets tels que soit A soit B est égal à S.

4.1.1 Travaux reliés

Bien que de nombreux travaux ont été consacrés à la vérification des protocoles cryptographiques et ce dans différents formalismes, seuls quelques-uns ont examiné le problème de l'extraction de définitions opérationnelles non ambiguës pour les rôles intervenant dans un protocole à partir de sa description. Des rôles opérationnels sont décrit comme des règles de réécriture sur les multi-ensembles dans CAPSL [35] et CASRUL [41], ou comme des processus séquentiels du *spi-calculus* avec *pattern-matching* [24]. Parallèlement, dans [45, 44] des rôles opérationnels sont extraits par projection de pairs (ou *end-point projection*). Un travail pionnier dans ce domaine est celui de Carlsen [26] qui a proposé une traduction des narrations décrivant des protocoles cryptographiques dans CKT5 [20], une logique modale de la communication, des connaissances et du temps.

Le compilation des narrations de protocoles vers des rôles a été étendue au-delà des primitives de chiffrement parfait pour prendre en compte des théories algébriques dans [31, 28, 47]. Un avantage de [47] est la prise en compte du déchiffrement implicite qui permettrait d'élaborer des procédures de décisions plus efficace du secret. On peut noter que, bien que ces travaux admettent des objectifs très similaires, tous leurs calculs de rôles opérationnels sont ad-hoc et manque d'uniformité. En particulier ils ont essentiellement ré-implémenté des techniques déjà connues.

4.1.2 Notre approche pour l'implémentation prudente de services web

Une autre motivation pour ce travail est la quantité existante de travaux sur la sécurité de protocoles qui utilisent des primitives cryptographiques très variées. Dans un tel contexte, des implémentations de protocoles sont données sans justification. En particulier, rien ne garantit *la prudence* de ces implémentations. Un premier résultat que nous présentons dans ce chapitre, est la formalisation des notions d'implémentation et d'implémentation prudente. La prudence est alors définie dans le sens qu'un agent jouant un protocole, doit tester et corréler les parties extractibles dans ses messages reçus.

Comme conséquence de ces formalisations, nous relions le problème de calcul d'implémentations prudentes à des procédures de décision classiques, à savoir les problèmes d'*atteignabilité* et d'*équivalence statique*. Étant donnée un système de déduction $\mathcal{D}$, nous prouvons en particulier qu'un algorithme solvant le problème d'atteignabilité pour $\mathcal{D}$ nous permet de calculer une implémentation pour un rôle r disposant des capacités de calcul décrites par $\mathcal{D}$. En outre nous prouvons qu'un algorithme résolvant le problème de l'équivalence statique peut être employé

pour calculer une implémentation prudente de r. Ainsi nous pouvons réutiliser des outils comme Yapa [15] pour compiler automatiquement des protocoles cryptographiques.

Nous procédons par *projection de pairs* (*End-Point Projection*) [25] de la trace de la conversation afin d'extraire le comportement local du médiateur. Comme cela a été souligné dans [44], la tâche n'est pas simple quand les messages échangés son sujet à des traitements cryptographiques. Le problème principal avec l'utilisation de la cryptographie est que la structure d'un message peut-être vue différemment par son émetteur et son récepteur. Pour illustrer cette propriété nous considérons la simple conversation de l'exemple 8, discuté dans la section suivante.

4.1.3 Exemple introductif pour l'implémentation prudente de services

Exemple 8 *Nous considérons un contexte où un service A qui détient initialement une clef de chiffrement symétrique k et message m et un service B sans connaissances initiales ont la conversation suivante :*

$A \text{ -> } B: \quad scrypt(k,m)$

Au premier abord on peut écrire les projections (ci-dessous) de la conversation pour A et B. Dans cette notation chaque service est paramétré par ses connaissances initiales et $?t$ et $!t$ représentent respectivement la réception et l'émission du message t.

$A(k,m) = !scrypt(k,m)$
$B() = ?scrypt(k,m)$

Le problème avec cette projection naïve de l'exemple 8 est que du point de vue de B le message reçu ne peut pas vraiment être comparée à son pattern fourni, $scrypt(k,m)$, car B ne détient pas k. Ainsi, vérifier une telle projection de B n'est pas vraiment significatif puisque ce comportement ne correspond pas à ce que pourrait être une véritable implémentation du service. Dans le pire des cas cela peut conduire à un modèle du service qui cacherait certaines failles durant la phase de vérification qui restent bien possibles dans son implémentation réelle.

Pour supprimer cette discordance entre ce qui est vérifié et ce qui pourrait être réellement implémenté, nous proposons de considérer un élément clef : du point de vue de chaque service intervenant dans la conversation les parties pertinentes dans un message émis ou reçu à une certaine étape i sont celles qui sont *atteignables* par le service à l'étape i. Nous définissons ici l'atteignabilité à une étape i d'un sous-terme t apparaissant dans la communication d'un service S, comme la capacité de S à synthétiser t, à partir de ses connaissances courantes qui englobent ses propres connaissances initiales et toutes celles qu'il aura atteint à partir de ses messages reçus jusqu'à l'étape i. Dans le cas d'une réception, ceci donne exactement les parties des messages qui doivent être corrélées avec leurs occurrences précédentes dans les messages reçus auparavant ou encore les tests de sécurité qui doivent être effectués sur le message reçu lui-même, par exemple, la vérification d'une signature électronique. Dans le cas d'une émission ceci rend explicites les parties nécessaires à la construction du message devant être envoyé.[4]

[4]Nous rappelons que par construction de la trace décrivant le médiateur, tous les messages devant être émis à une étape donnée, sont déjà atteignables par le médiateur à cette étape.

4.2 Représentation de Services par des Strands

4.2.1 Les strands

Nous formalisons les activités de communications d'un service à l'aide d'un *strand* [38], une notion standard dans la modélisation des protocoles cryptographiques.

Définition 1 *Un* strand s *est une séquence finie de messages portant chacun un label* ! *ou* ?. *Un message labellisé par* ! *(respectivement* ?*) est un message envoyé (respectivement reçu). Un strand est* positif *si et seulement si tous ses labels sont* ?. *La* longueur *d'un strand* $s = \overset{!}{?}m_1, \ldots, \overset{!}{?}m_n$ *est l'entier* n *et son* entrée *notée* $\text{input}(s)$ *est le strand* $(?r_1, \ldots, ?r_{n'})$ *où* $r_1, \ldots, r_{n'}$ *est la sous-séquence ordonnée des messages de* s *portant le label* ?.

On note s^i le préfixe $(\overset{!}{?}m_1, \ldots, \overset{!}{?}m_i)$ de s et s_i le message labellisé $\overset{?}{!}m_i$. On définit *la substitution* de s, notée σ_s, la substitution $\{x_i \mapsto m_i\}_{1 \leq i \leq n}$. On définit aussi *la substitution d'entrée* de s, notée σ_s^{input}, comme la restriction de σ_s à l'ensemble $\{x_i \mid s_i = ?m_i\}$. Pour modéliser les connaissances initiales $IK(s)$ du service correspondant au strand s, nous préfixons s par une réception $?t$ pour tout terme t apparaissant dans $IK(s)$. Un strand est dit *clos* si tous ses messages sont clos. Dans la suite nous supposons que pour tout strand s, $\top \in IK(s)$ et que nous considérons uniquement des strands clos et donnés en forme normale.

Définition 2 *Étant donné un strand* s, *un contexte* C *et un terme clos* t, *on dit que* C *s'évalue à* t *sur* s, *si et seulement si* $\text{Var}(C) \subseteq \text{Supp}(\sigma_s^{input})$ *et* $C\sigma_s^{input} =_{\mathcal{E}_{XML}} t$.

Dans la section suivante nous donnons une sémantique opérationnelle aux activités d'émission et de réception définies par un strand.

4.2.2 Sémantique opérationnelle d'un strand

Définition 3 *Un* système d'unification S *est un ensemble fini d'équations notées* $(u_i \overset{?}{=} v_i)_{i \in \{1,\ldots,n\}}$ *avec les termes* $u_i, v_i \in \mathrm{T}(\mathcal{F}, \mathcal{X})$. *Il est satisfait par une substitution* σ *si pour tout* $i \in \{1, \ldots, n\}$ *nous avons* $u_i\sigma =_{\mathcal{E}_{XML}} v_i\sigma$ *et on note ceci :* $\sigma \models S$.

Matrice active

Les *matrices actives* donnent une sémantique opérationnelle aux strands. C'est un modèle de processus simple dans lequel la construction des messages envoyés et la vérification des messages reçus sont spécifiées. La notation $?r_i$ (respectivement $!e_i$) se réfère à un message reçu et stocké dans la variable r_i (respectivement émis et stocké dans la variable e_i).

Définition 4 *Une* matrice active *est une séquence* $(T_i)_{1 \leq i \leq k}$, *où :*

$$T_i = \begin{cases} !e_i \text{ avec } e_i \overset{?}{=} C_i[r_1, \ldots, r_{i-1}] & \textit{(envoi)} \\ \textbf{ou} & \\ ?r_i \text{ avec } S_i(r_1, \ldots, r_i) & \textit{(réception)} \end{cases}$$

et où $C_i[r_1, \ldots, r_{i-1}]$ *est un contexte,* S_i *est un système d'unification ayant pour variables :* $\langle r_j \rangle_{1 \leq j < i}$. *Une variable* r_i *(respectivement* e_i*) est appelée* variable d'entrée *(respectivement* variable de sortie*) de la matrice active.*

Définition 5 *Soit $\varphi = (T_i)_{1\leq i\leq k}$ une matrice active comme dans la définition 4 et où les variables d'entrée sont $r_1, \ldots, r_n$. Soit $s = !M_1, \ldots, !M_n$ un strand positif, $\sigma_{\varphi,s}$ la substitution $\{r_i \mapsto M_i\}$ et S l'union des systèmes d'unification dans φ. L'*évaluation *de φ sur s est notée $\varphi \cdot s$ et est le strand $(m_i)_{1\leq i\leq k}$ où :*

$$m_i = \begin{cases} !C_i[m_1, \ldots, m_{i-1}] & \text{si } T_i \text{ est } !e_i \\ ?r_i\sigma_{\varphi,s} & \text{si } T_i \text{ est } ?r_i \end{cases}$$

On dit que φ accepte *s si $S\sigma_{\varphi,s}$ est satisfiable.*

Définition 6 *Une matrice active φ est une* implémentation *du strand s si φ accepte* $\text{input}(s)$ *et si $\varphi \cdot \text{input}(s) =_{\mathcal{E}} s$. Si un strand s admet une implémentation, s est dit* exécutable.

4.2.3 Compilation d'un strand vers son implémentation

Étant donné un strand s, une première obligation est la suivante : si jusqu'à une étape i dans laquelle un message m est envoyé tous les messages reçus auparavant sont ceux spécifiés dans s, alors m doit être égale modulo la théorie $\mathcal{E}_{XML}$ à la réponse définie dans s. Pour satisfaire cette condition il suffit de calculer pour tout message m_i, émis à une étape i, un contexte C_{m_i} qui s'évalue à m_i sur l'entrée du strand s^i.

Définition 7 *Étant donné un strand s de longueur n et un terme clos t, un* algorithme d'atteignabilité *$\mathcal{A}_r$ calcule un contexte $\mathcal{A}_r(s,t)$ qui s'évalue à t sur s, si un tel un contexte existe (on dit alors que t est atteignable à partir s) et $\perp$ dans le cas contraire. On note $RST_i(s)$ l'ensemble de tous les sous-termes de s atteignable à partir de s^i et $RST_i^{new}(s)$ l'ensemble $RST_i(s) \setminus RST_{i-1}(s)$ (avec la convention $RST_j(s) = \emptyset$ pour tout j négatif). On utilise aussi l'abréviation $RST(s)$ pour $RST_n(s)$.*

Ceci dit, calculer une matrice active n'est pas suffisant pour assurer une vérification aussi complète que possible des messages reçus. Nous formalisons cette remarque à l'aide d'une relation de raffinement sur les séquences de messages. Un strand *s raffine* un strand s' si toute égalité *observable* entre deux messages dans s' peut aussi être observée dans s en utilisant les mêmes tests. Plus formellement :

Définition 8 *Étant donné un strand s, on note P_s l'ensemble de tous les couples de contextes $\{C_1, C_2\}$ tels que $C_1 \cdot s =_{\mathcal{E}_{XML}} C_2 \cdot s$. On dit alors que s raffine un strand s' si et seulement si $P_{s'} \subseteq P_s$.*

Exemple 9 *Considérons les strands suivants :*

$$\begin{cases} s &= ?pair(a,b)?pair(a,b)!a \\ s' &= ?pair(a,b)?pair(a,c)!a \end{cases}$$

Comme toute égalité valide sur s' est aussi valide sur s nous concluons que s raffine s'.

Nous utilisons la relation de raffinement pour définir le sens dans lequel une implémentation peut tester de la manière la plus complète son entrée.

Définition 9 *Soit s un strand et φ une implémentation de s. On dit que φ est* prudente *si tout strand s' accepté par φ est un raffinement de s.*

Définition 10 *Étant donné un strand s, un système d'unification P_s^f est une base finie de P_s si pour tout strand s' : $\sigma_{s'}^{input} \models P_s^f$ si et seulement si s' est un raffinement de s.*

Par abus de langage, on dira aussi que P_s^f est une base finie de s.

Maintenant supposons qu'il existe : un algorithme $\mathcal{A}_b(s)$ qui prend en entrée un strand s et calcule une base finie P_s^f de s et un algorithme d'atteignabilité $\mathcal{A}_r(s,t)$ tel que celui introduit dans la définition 7. Ces deux algorithmes vont servir de boites noires pour notre algorithme de compilation décrit ci-dessous :

L'algorithme $\mathcal{A}_c$

Soit $s = (\frac{!}{?}m_1, \ldots, \frac{!}{?}m_n)$ un strand. Calculer la matrice active $\varphi_s = (T_i)_{1\leq i\leq n}$ avec, pour tout i tel que $1 \leq i \leq n$:

$$T_i = \begin{cases} !x_i \text{ avec } x_i \stackrel{?}{=} \mathcal{A}_r(s^{i-1}, m_i) & \text{si } s_i = !m_i \\ ?x_i \text{ avec } \mathcal{A}_b(s^i) & \text{si } s_i = ?m_i \end{cases}$$

et retourner la matrice active $\varphi_s = (T_i)_{1\leq i\leq n}$.

Par construction nous avons le corollaire suivant, que nous formulons à l'aide des mêmes notations, introduites plus haut :

Théorème 2 *Étant donné les algorithmes $\mathcal{A}_r$ et $\mathcal{A}_b$, et un strand s exécutable tel que $\mathcal{A}_r(s^{i+1}, m_i)$ ne retourne jamais $\perp$ pour tout $s_i = !m_i$, alors l'algorithme $\mathcal{A}_c$ calcule une implémentation prudente de s.*

4.3 Compilation des strands vers des implémentations prudentes

Nous présentons dans cette section des algorithmes résolvant le problème d'atteignabilité et calculant une base finie pour un strand donné.

4.3.1 Principe de l'algorithme

Afin de calculer une implémentation prudente d'un strand s il nous faut considérer tous les contextes qui s'évaluent au même terme t sur s. En principe, nous devons considérer l'ensemble des possibilités infinies pour t et par conséquent le calcul explicite de cet ensemble est impossible. Par ailleurs, lorsque t est fixé il y a encore un nombre infini de contextes à considérer même si nous limitons l'étude à ceux en forme normale, comme expliqué dans l'exemple 10 ci-dessous.

Exemple 10 *Considérons le strand $s = ?k?scrypt(k,k)$. Nous avons l'égalité $sdcrypt(x_2, x_1) \cdot s =_{\mathcal{E}_{XML}} x_1 \cdot s$ et donc nous pouvons calculer une séquence finie de contextes en forme normale s'évaluant à k sur s, en remplaçant d'une manière itérative x_1 dans $sdcrypt(x_2, x_1)$ par $sdcrypt(x_2, x_1)$: $sdcrypt(x_2, sdcrypt(x_2, \ldots)) \cdot s =_{\mathcal{E}_{XML}} x_1 \cdot s$.*

Le principe de la solution que nous proposons est de considérer *uniquement* les relations de la forme $t = f(t_1, \ldots, t_k)$ modulo $\mathcal{E}_{XML}$, vérifiées par les termes $t, t_1, .., t_k$ dans $RST(s)$ et où f est un symbole public. D'abord nous calculons le sur-ensemble de ces relations vérifiées par les termes dans $ST(s)$ (Algorithme 1). Ensuite nous en éliminons celles qui ne font pas intervenir des termes dans $RST(s)$. La dernière opération est exécutée en un temps linéaire comme suit : une relation $t = f(t_1, \ldots, t_k)$ calculée par l'algorithme 1 est utilisée pour inférer l'atteignabilité

du terme t, pourvu que les termes $t_1, \ldots, t_k$ soient atteignables. En effet si $C_1, \ldots, C_k$ sont des contextes d'extraction pour $t_1, \ldots, t_k$ alors $f(C_1, \ldots, C_k)$ est un contexte d'extraction pour t. L'ensemble $RST_i(s)$ est alors calculé comme suit. En supposant que $s_i = ?m_i$, nous commençons le calcul avec l'ensemble $R = RST_{i-1}(s) \cup \{m_i\}$. Tous les termes de cet ensemble sont trivialement atteignables à partir de s^i, puisque ceux dans $ST_{i-1}(s)$ sont atteignables à partir de s^{i-1} (et par conséquent à partir de s^i) et puisque m_i l'est aussi grâce au contexte x_i. Ensuite nous visitons toutes les relations $t = f(t_1, \ldots, t_k)$ où $\{t_1, \ldots, t_k\} \subseteq R$. Pour chaque relation similaire visitée, le terme t est alors atteignable. Ce résultat peut être alors utilisé pour découvrir de nouveaux termes atteignables dans $RST_i(s)$, ou de nouveaux contextes d'extraction pour des termes dont l'atteignabilité était déjà connue. Enfin nous extrayons de tous ces contextes calculés, l'ensemble des couples de contextes qui s'évaluent sur le même terme t de s et nous prouvons que le résultat ainsi trouvé est une base finie de s.

Nous remarquons que cette approche fournit un contexte d'extraction pour tous les messages envoyés dans s, s'ils sont tous atteignables à partir de s (s est alors exécutable). Ceci permet d'utiliser le théorème 2 pour dériver une implémentation prudente de s.

Dans la suite les relations $t = f(t_1, \ldots, t_k)$ définies plus haut sont représentées par des *séquents* qui sont vrais pour le strand s.

Définition 11 *Étant donné un strand s de longueur n nous définissons les séquents*

$$t_1, \ldots, t_k \vdash_f t$$

où $t \in ST(s)$, $t_1, \ldots, t_k$ est une séquence (éventuellement vide) d'éléments de $ST(s)$ et où f est soit un symbole public d'arité k soit une variable dans $\{x_1, \ldots, x_n\}$. Étant donné γ le séquent $t_1, \ldots, t_k \vdash_f t$, on appelle t la partie droite *de γ, f son* symbole *et la séquence $t_1, \ldots, t_k$ sa* partie gauche. *Nous les notons respectivement $rhs(\gamma)$, $symbol(\gamma)$ and $lhs(\gamma)$. Le séquent γ est* vrai *si*

a. soit f est un symbole public d'arité k et $t =_{\mathcal{E}_{XML}} f(t_1, \ldots, t_k)$;

b. soit $t_1, \ldots, t_k$ est la séquence vide, $f = x_i \in \mathrm{Supp}(\sigma_s^{input})$ et $x_i\sigma_s =_{\mathcal{E}_{XML}} t$.

On note dans la suite $S(s)$ l'ensemble de tous les séquents vrais de s et $R(s)$ le sous-ensemble de $S(s)$ contenant les séquents $t_1, \ldots, t_k \vdash_f t$, tels que $t, t_1, \ldots, t_k \in RST(s)$.

Soit s un strand de longueur n. Pour chaque étape $i \in \{1, \ldots, n\}$ et pour tout terme t dans $RST_i(s)$, soit $R_i(s,t)$ l'ensemble contenant le séquent $\vdash_{x_i} t$ si $s_i = ?t$ et tous les séquents $t_1, \ldots, t_k \vdash_f t$ tels que :

$$\begin{cases} \{t_1, \ldots, t_k\} \subseteq RST_i(s) \\ \{t_1, \ldots, t_k\} \cap RST_i^{new}(s) \neq \emptyset \end{cases}$$

Soit alors $R_i(s) = \bigcup_{t \in RST_i^{new}(s)} R_i(s,t)$ et $Y_{RST(s)} = \{y_t \mid t \in RST(s)\}$ un ensemble de variables[5] indexées par les termes de $RST(s)$. Par ailleurs, soit γ le séquent $t_1, \ldots, t_k \vdash_f t$ (respectivement $\vdash_{x_j} t$) dans $R_i(s,t)$, le *contexte* de γ noté $context(\gamma)$ est le terme $f(y_{t_1}, \ldots, y_{t_k})$ (respectivement x_j). En outre nous définissons, $\mathcal{C}_i(s,t) = context(R_i(s,t))$, $\mathcal{C}_i(s) = context(R_i(s))$ et $\mathcal{C}(s) = context(R(s))$.

Soit $<_{R(s)}$ un ordre total sur $R(s)$ et soit pour tout t dans $RST(s)$ le séquent :

$$\gamma_{min}(s,t) = min\{\gamma \in R(s) \mid t \in rhs(\gamma) \cup lhs(\gamma)\}$$

$\gamma_{min}(s,t)$ est le plus petit séquent de $R(s)$ dans lequel t apparait. Maintenant supposons [6] de plus que $<_{R(s)}$ satisfait les propriétés suivantes pour tout t dans $RST(s)$:

[5] nous supposons dans la suite que $X \cap Y_{RST(s)} = \emptyset$

[6] l'existence d'un tel ordre est prouvée dans la section 4.3.4

P1 : $t = rhs(\gamma_{min}(s,t))$;
P2 : $\gamma_{min}(s,t') <_{R(s)} \gamma_{min}(s,t)$ pour tout t' dans $lhs(\gamma_{min}(s,t))$.
P3 : $\vdash_{x_i} t <_{R(s)} \vdash_{x_j} t$ si et seulement si $i < j$
Soit pour tout t dans $RST(s)$, $C_{min}(s,t) = context(\gamma_{min}(s,t))$. Nous définissons pour tout i dans $\{1,\ldots,n\}$, le système d'unification suivant avec les variables $\{x_1,\ldots,x_i\} \cup \{y_t \mid t \in RST_i(s)\}$

$$\mathcal{U}_i(s) = \bigcup_{t \in RST_i(s)} \left\{ C_{min}(s,t) \stackrel{?}{=} C \mid C \in \mathcal{C}_i(s,t) \setminus \{C_{min}(s,t)\} \right\}$$

Enfin et dans la suite $\mathcal{U}_n(s)$ et aussi noté $\mathcal{U}(s)$ (avec n la longueur de s).

Théorème 3 *Soit s un strand de longueur n. Pour toute étape $1 \leq i \leq n$, soit $t_1,\ldots,t_{k(i)}$ une énumération des éléments de l'ensemble $RST_i^{new}(s)$, tels que*

$$C_{min}(s,t_1) <_{R(s)} \ldots <_{R(s)} C_{min}(s,t_{k(i)})$$

Et soit :
- $\tau_{s,i} = \{y_{t_1} \mapsto C_{min}(s,t_1)\} \circ \ldots \circ \left\{ y_{t_{k(i)}} \mapsto C_{min}(s,t_{k(i)}) \right\}$
- $\overline{\tau}_{s,i} = \tau_{s,1} \circ \ldots \circ \tau_{s,i}$

Pour toute étape i dans $\{1,\ldots,n\}$ nous avons :
1. *le contexte $C_{min}(s,t)\overline{\tau}_{s,i}$ s'évalue à t sur s^i pour tout t dans $RST_i(s)$;*
2. *$\mathcal{U}_i(s)\overline{\tau}_{s,i}$ est une base finie de s^i.*

La preuve complète du théorème 3 est dans le chapitre A. Son argument principal est la *localité* [42] de la théorie $\mathcal{E}_{XML}$. Ceci nous permet de résoudre le problème d'atteignabilité générale, en considérant uniquement sa restriction aux sous-termes d'un strand donné.

Dans la suite nous présentons les algorithmes qui calculent les systèmes d'unification $\{\mathcal{U}_i(s)\}_{1\leq i\leq n}$ et les applications $\{\overline{\tau}_{s,i}\}_{1\leq i\leq n}$ pour un strand s donné de longueur n. Ceci permet de calculer les bases finies de $\{s^i\}_{1\leq i\leq n}$, selon le théorème 3. En particulier nos algorithmes fournissent pour tout t dans $RST_i(s)$ le contexte $C_{min}(s,t)$. Ces contexte, avec $\{\overline{\tau}_{s,i}\}_{1\leq i\leq n}$, est un contexte d'extraction pour t à partir de s, et ce pour tout t dans $RST(s)$. Ainsi si tous les s_{i+1} labellisé par ! dans s sont atteignables à partir de s^i, nous pouvons alors fournir une implémentation prudente de s, d'après le théorème 2.

4.3.2 Algorithme linéaire de compilation

D'abord nous introduisons les structures de données pour représenter les termes, (y compris le cas spécial des contextes et donc les systèmes d'unification), les séquents et les strands. Ensuite, nous présentons les principes des algorithmes 1 et 2.

Tableaux et files

Nous utilisons des files FIFO et des tableaux pour contenir les objets représentant les termes et les séquents. Nous adoptons aussi une notation orientée objet. Étant donné un objet tableau A, $A.add(t)$ rajoute l'élément t au tableau et retourne non index, $A.nbelements()$ retourne le nombre d'éléments dans le tableau A et $A[i]$ retourne l'élément stocké à l'index i dans A si $i \leq A.nbelements()$. Étant donné un objet file Q, $Q.pop()$ consomme et retourne le premier élément dans Q, tandis que $Q.push(o)$ rajoute o à la fin de Q et $Q.nbelements()$ retourne le nombre d'éléments dans Q. Toutes les opérations décrite plus haut peuvent être implémentées en un temps constant. Enfin, pour un tableau ou une file O, $O.size()$ représente la somme de toutes les tailles des objets contenus dans O.

Représentation des termes

Un ensemble de termes S est stocké dans un tableau $\mathcal{A}$ d'*objets terme*, où chaque terme $t \in S$ est représenté par un objet terme ayant les champs suivants :

id : un entier identifiant t avec l'hypothèse $\mathcal{A}[i].id = i$ pour tout $1 \leq i \leq \mathcal{A}.nbelements()$;

symbol : un élément de $\mathcal{F}$ ou une constante qui représente le symbole de tête de t ;

dst : un tableau contenant les identifiants dans $\mathcal{A}$ des sous-termes maximaux de t ;

context : un entier identifiant le contexte $C_{min}(s,t)$ (les contextes sont décrits ci-dessous) ;

sequents : une file contenant les identifiants des séquents (décrits ci-dessous) où t apparait dans le côté gauche ;

inv : l'identifiant de $inv(t)$ dans $\mathcal{A}$.

Dans l'algorithme 1, un test de la forme $t = f(t_1, \ldots, t_n)$ équivaut à vérifier si $t.symbol = f$ et le cas échéant chaque t_i est assigné à $t.dst[i]$. On définit la taille d'un terme t comme étant la taille de l'objet terme qui le contient, c'est à dire la somme des tailles de ses champs énumérés ci-dessus.

Représentation des contextes et des systèmes d'unification

Un ensemble de contextes est stocké dans un tableau $\mathcal{C}$ d'*objets contextes*, où chaque contexte est représenté par un *objet contexte*. Un objet contexte est un cas particulier de l'objet terme, où uniquement les champs *symbol* et *dst* sont significatifs.

Une équation $C \stackrel{?}{=} C'$ est alors représentée par un couple d'entiers $(id_C, id_{C'})$, où id_C et $id_{C'}$ sont les identifiants des objets contextes représentant respectivement C et C' dans $\mathcal{C}$.

Un système d'unification est alors représenté par une file contenant toutes représentations des équations dans U.

Représentation des strands

Un strand $s = (\stackrel{?}{!} m_i)_{1 \leq i \leq n}$ est représenté par le couple $(\mathcal{A}, IO)$, où $\mathcal{A}$ est la représentation de $ST(s)$ et où IO est un tableau contenant les paires $(m_i.id, \stackrel{?}{!})_{1 \leq i \leq n}$ dans l'ordre. La taille de s est notée $|s|$ et elle vaut $\mathcal{A}.size() + IO.size()$.

Représentation des séquents

Un séquent γ est représenté par un *objet séquent* ayant les champs suivants :

id : un entier identifiant γ ;

rhs : un entier identifiant la partie droite de γ ;

symbol : un élement de $\mathcal{F}_p$ représentant le symbole de γ ;

lhs : un tableau d'entiers identifiant les termes dans la partie gauche de γ ;

ready : un entier représentant le nombre d'occurrences de termes dans le côté gauche de γ qui ne sont pas encore accessibles. Il vaut initialement l'arité du symbole de γ.

4.3.3 Calcul des sous-termes accessibles

Étant donné une représentation $(\mathcal{A}, IO)$ d'un strand s, notre objectif est de calculer un tableau $\mathcal{S}$ contenant les représentations des séquents dans $S(s)$ et de mettre à jour les champs *sequents* pour tous les éléments dans $\mathcal{A}$. La mise à jour est effectuée sur les tableaux globaux $\mathcal{A}$ et $\mathcal{S}$ par la méthode *register* :

method register($id_1, \ldots, id_n \vdash_f id$)
cr $\leftarrow \mathcal{S}$.add($id_1, \ldots, id_n \vdash_f id$)
for all $k \in \{1, \ldots, n\}$ **do** $\mathcal{A}[id_k]$.sequents.push(cr) **end for**
return cr
end method

Algorithm 1 Calcul de $S(s)$

1: $\mathcal{S} \leftarrow \emptyset$
2: **for all** t $\in \mathcal{A}$ **do**
3: **if** $t = scrypt(k, m)$ **then**
4: $\mathcal{S}$.register($k.id, m.id \vdash_{scrypt} t.id$)
5: $\mathcal{S}$.register($k.id, t.id \vdash_{sdcrypt} m.id$)
6: **else if** $t = crypt(k, m)$ **then**
7: $\mathcal{S}$.register($k.id, m.id \vdash_{crypt} t.id$)
8: $\mathcal{S}$.register($k.id, t.inv \vdash_{dcrypt} m.id$)
9: **else if** $t = sign(inv(k), m)$ **then**
10: $\mathcal{S}$.register($inv(k.id, m.id \vdash_{sign} t.id$)
11: $\mathcal{S}$.register($k.id, m.id, t.id \vdash_{verif} \top.id$)
12: **else if** $t' = inv(t)$ **then**
13: $\mathcal{S}$.register($t.id, t'.id \vdash_{invtest} \top.id$)
14: **else if** $t = node_a^n(t_1, \ldots, t_a)$ **then**
15: $\mathcal{S}$.register($t_1.id, \ldots, t_a.id \vdash_{node_a^n} t.id$)
16: **for all** $i \in \{1, \ldots, a\}$ **do**
17: $\mathcal{S}$.register($t.id \vdash_{child_{\frac{i}{a}}^n} t_i.id$)
18: **end for**
19: **end if**
20: **end for**
21: **return** $\mathcal{S}$

Principe de l'algorithme 1.

Étant donné un strand s et pour tout terme $t \in ST(s)$, nous effectuons une analyse au cas par cas sur sa structure pour calculer les séquents. Ensuite nous insérons ces séquents dans $\mathcal{S}$ à l'aide de la méthode *register*. Nous remarquons que chaque sous-terme t de s contribue à $S(s)$ par un nombre de séquents dépendant uniquement de son symbole de tête. Par conséquent la valeur de $\mathcal{S}.nbelements()$ peut être calculée à l'avance et est linéaire en la taille de l'entrée $(\mathcal{A}, IO)$. En fait, $\mathcal{S}$ ne contient pas encore les séquents de $S(s)$ ayant des parties gauches vides. Ces séquents sont ensuite rajoutés à $\mathcal{S}$ par l'algorithme 2.

Complexité de l'algorithme 1

La boucle la plus externe parcourt les sous-termes de s stockés dans $\mathcal{A}$. Chaque sous-terme t visité donne lieu à $t.dst.nbelements() + 2$ instructions exécutées en temps constant, pour collecter les valeurs de $t.id$, $t.symbol$ et toutes les valeurs de $t.dst[i].id$, où i varie entre 0 et $t.dst.nbelements()$. En outre si $t.symbol$ est dans l'ensemble $\{crypt, sign, inv\}$, nous devons aussi collecter $inv(t.dst[1]).id$, ce qui est effectué en un temps constant. Dans ce dernier cas, le nombre total d'instructions à effectuer en un temps constant est $t.dst.nbelements() + 3$. Ensuite, suivant le symbole de tête de t, un nombre fixe d'appels de la méthode *register* est effectué. Nous détaillons ci-dessous le coût en temps d'exécution selon $t.symbol$:

scrypt : Le coût est de $8 + 6t.dst.nbelements()$ et correspond à 2 appels de la méthode *register*. Pour chaque appel, une structure de séquent est d'abord construite en $2 + t.dst.nbelements()$ instructions, chacune exécutée en temps constant. Ensuite la structure est insérée dans $\mathcal{R}$ et le compteur cr est incrémenté en 2 instructions exécutées en temps constant. Enfin $2t.dst.nbelements()$ instructions collectent et mettent à jour $t.dst[i].sequents$ champs, totalisant $4 + 3t.dst.nbelements()$ instructions exécutées en temps constant.

inv : Le coût est de $4 + 3t.dst.nbelements()$ et correspond à 1 appel de la méthode *register*. D'abord une structure de séquent est construite en $2 + t.dst.nbelements()$ instructions exécutées en temps constant. Ensuite la structure est insérée dans $\mathcal{R}$ et le compteur cr est incrémenté en 2 instructions exécutées en temps constant. Enfin $2t.dst.nbelements()$ instructions collectent et mettent à jour $t.dst[0].sequents$ champs, totalisant $4 + 3t.dst.nbelements()$ instructions exécutées en temps constant.

crypt, sign : Le coût est $8 + 8t.dst.nbelements()$. La seule différence avec le cas *scrypt* est que chaque appel de la méthode *register*, le champs *sequents* est mis à jour pour $inv(t.dst[1])$ à la place de $t.dst[1]$. Ensuite $t.dst.nbelements()$ instructions supplémentaires exécutées en temps constant sont nécessaires pour collecter à chaque fois l'indice de $inv(t.dst[1])$ dans $\mathcal{A}$ depuis $t.dst[1]$.

node$_a^n$: Le coût est de $4 + 10t.dst.nbelements()$ et correspond à $1 + t.dst.nbelements()$ appels de la méthode *register*. En effectuant une analyse similaire, le premier appel coute $4 + 3t.dst.nbelements()$ instructions exécutées en un temps constant alors que chacun des $t.dst.nbelements()$ appels consécutifs de la méthode *register* (dans la boucle *for all* interne) coute 7 instructions exécutées en un temps constant.

Ainsi nous concluons que l'algorithme 1 traite chaque sous-terme t de s en un nombre d'instructions exécutées en un temps constant qui reste proportionnel à la taille de t. Ce qui permet de prouver la linéarité de l'algorithme par rapport à la taille de s.

4.3.4 Calcul des bases finies

Étant donné la représentation $(\mathcal{A}, IO)$ d'un strand s de longueur n, $\mathcal{S}$ et $S(s)$, nous calculons un tableau $\mathcal{C}$ représentant les contextes de $\mathcal{C}(s)$ et les tableaux $\mathcal{I}$ et $\mathcal{U}$ représentant une implémentation prudente de s. Ces tableaux vérifient les conditions ci-dessous :

1. si $s_i = !m_i$ alors $\mathcal{I}$[i] est l'index de l'objet contexte $C_{min}(s, m_i)\overline{\tau}_{s,i}$ dans $\mathcal{C}$[7] ;
2. si $s_i = ?m_i$ alors $\mathcal{U}$[i] est une file représentant le système d'unification $\mathcal{U}_i(s)\overline{\tau}_{s,i}$.

L'algorithme 2 utilise la procédure *register2* qui met à jour le tableau global $\mathcal{C}$.

method register2($f[id_1, \ldots, id_n]$)

[7] le minimum ici est pris au sens de l'ordre $<_Q$ introduit dans la section 4.3.4

cr $\leftarrow \mathcal{C}$.add($f[\mathcal{A}[id_1].context, \ldots, \mathcal{S}[id_n].context]$)
return cr
end method

Algorithm 2 Calcul des $\mathcal{U}_i(s)\overline{\tau}_{s,i}$

```
1: S ← Sortie de l'algorithm 1
2: C,Q,step ← ∅,∅,0
3: for all m_i ∈ IO do
4:    step++
5:    if m_i = (id_i, ?) then
6:       Q.push(S.add( ⊢_{x_i} id_i))
7:       while Q ≠ ∅ do
8:          seq ← Q.pop()
9:          t ← S[seq.rhs.id]
10:         ind = register2(seq.symbol[seq.lhs])
11:         if t.context = null then
12:            t.context ← ind
13:            while t.sequents ≠ ∅ do
14:               seq' ← S[t.sequents.pop()]
15:               seq'.ready−−
16:               if seq'.ready = 0 then
17:                  Q.push(seq')
18:               end if
19:            end while
20:         else
21:            U[step].push((t.context,ind))
22:         end if
23:      end while
24:   else if m_i = (id_i, !) then
25:      I[step] ← A[id_i].context
26:   end if
27: end for
28: return I,U,C
```

Principe de l'algorithme 2.

A partir du tableau global de séquents $\mathcal{S}$ calculé par l'algorithme 1, l'algorithme 2 calcule d'une manière itérative les sous-termes atteignables à partir de s pour toutes les étapes de réceptions de s. Si un message labellisé $s_i = !m_i$ est tel que m_i est atteignable à partir de s alors un contexte d'extraction de m_i à partir de s est stocké dans $\mathcal{I}$. D'où le fait que $\mathcal{I}$ permet de simuler un appel à un oracle $\mathcal{A}_r$ en prenant $\mathcal{A}_r(s^{i-1}, m_i) = \mathcal{I}[i]$ pour $s_i = !m_i$. De manière similaire, le tableau $\mathcal{U}$ stocke les contextes d'extraction pour les sous-termes atteignables à partir de s (à chaque étape) et peut être utilisé pour construire une base finie de s et tous ses préfixes en prenant $\mathcal{A}_b(s^i) = \mathcal{U}[i]$.

Correction de l'algorithme 2

La correction de l'algorithme 2 est basée sur le fait que l'ordre dans lequel il insère les contextes satisfait les propriétés P1–P3 imposées sur $<_{RST(s)}$. Nous introduisons quelques notations pour les besoins de la preuve. Dans cette preuve, nous numérotons l'exécution de chaque étape atomique dans l'algorithme 2 par un indice k. On note alors $l(k)$ la ligne de l'algorithme exécutée à l'étape k.

Affirmation 1 : Au début de chaque itération de la boucle **for** externe, la file $\mathcal{Q}$ est vide. En effet, ceci est vrai après la phase d'initialisation étant donnée l'assignation $\mathcal{Q} = \emptyset$, et cela reste vrai à la fin de chaque itération vu la condition de sortie de la boucle **while**.

Affirmation 2 : A la fin de chaque itération i de la boucle **for** externe, chaque terme t tel que $t.context \neq null$ est dans $RST_i(s)$. Pour prouver ceci nous faisons le raisonnement suivant :

1. A chaque étape k de l'algorithme nous définissons l'ensemble D_k des termes t tels que $t.context \neq null$;
2. Nous remarquons que $D_{k+1} = D_k \cup \{t\}$ si $l(k) = 12$, et que si $l(k) \neq 12$ alors nous avons $D_{k+1} = D_k$.
3. A chaque étape k nous ordonnons les termes dans D_k suivant l'ordre $<_k$ défini par $t <_k t'$ si, et seulement s'il existe $k' < k$ avec $t \in D_{k'}$ et $t' \notin D_{k'}$;
4. Comme $t.context$ n'est jamais assignée la valeur $null$, la séquence d'ensembles $(D_k)_{k \geq 0}$ croît d'une manière monotone. Ainsi, si $t <_k t'$, alors pour tout $k' > k$ nous avons toujours $t <_{k'} t'$.
5. Par 2, quand un terme t est ajouté à l'ensemble D_k nous avons $t' <_{k+1} t$ pour tout $t' \in D_k$ et donc à chaque étape k l'ordre $<_k$ est total sur D_k.
6. Une caractéristique de l'algorithme 2 (non prouvée ici) est qu'un contexte C est inséré dans la file $\mathcal{Q}$ si, et seulement si, la boucle while interne des lignes 13–19 a été exécutée pour tout t dans sa partie gauche.
7. Par analyse structurelle, si $l(k) \in \{13, \ldots, 19\}$ alors le terme t avec lequel la boucle while interne est executée est dans D_k.
8. Par conséquent, à chaque étape k, si le contexte $t_1, \ldots, t_k \vdash_f t$ est dans $\mathcal{Q}$ alors pour tout $i \in \{1, \ldots, n\}$ le terme t_i est dans D_k.
9. Une conséquence des points précédents est qu'à chaque étape k l'ordre $<_k$ est total sur D_k ; de plus il est tel que pour tout terme $t \in D_k$, $t.context$ est un contexte $t_1, \ldots, t_n \vdash_C t$ avec $t_i <_k t$ pour tout $i \in \{1, \ldots, n\}$.
10. Si on suppose qu'au début de la boucle **for** externe chaque terme $t \in D_k$ est dans $RST_{step}(s)$, alors par induction sur l'ordre $<_k$, si k est une étape dans l'exécution de la $step$ième boucle *for* externe, alors $t \in D_k$ implique $t \in RST_{step}(s)$.
11. Cet invariant de boucle prouve l'assertion.

Affirmation 3 : Soit $<_R$ l'ordre total sur les séquents extraits de la file $\mathcal{Q}$, durant l'exécution de l'algorithme. Pour toute paire $(t.context, ind)$ ajoutée à $\mathcal{U}$ par l'exécution de la ligne 21, nous avons $t.context <_R ind$.

La preuve consiste à remarquer que $t.context$ est initialement nul, et que la paire est rajoutée à $\mathcal{U}$ uniquement quand $t.context$ est mis à jour. D'où le contexte désigné par $t.context$ a du être extrait de la file avant celui désigné par ind.

Affirmation 4 : Soit $\gamma_{\min}(s,t)$ le contexte $t.context$. L'ordre $<_R$ vérifie bien les propriétés $P1 - -P3$.

Comme t est la partie droite du contexte $t.context$, la propriété P1 est valide par construction. La propriété P2 est quant à elle une conséquence de l'affirmation et des définitions de D_k et $<_R$.

Complexité de l'algorithme 2.

Étant donné un strand s chaque séquent γ dans $S(s)$ est au plus rajouté une fois dans la file Q (quand $\gamma.ready = 0$). En outre chaque fois un tel séquent est traité, l'algorithme parcourt tous les éléments dans $rhs(e).sequents$ et $lhs(e)$. Comme expliqué précédemment dans la section 4.3.3, le coût de chaque traitement est linéaire en la taille du strand s. En conclusion l'algorithme 2 a une complexité linéaire par rapport à la taille de son entrée.

4.4 Génération de code

4.4.1 Détails de la génération de la spécification ASLan

Étant donné un partenaire $p = (n, \mathcal{IK}, c)$ nous proposons de générer la spécification ASLan de son role $r(p)$. D'abord nous identifions par une variable fraîche X_i tout sous-terme t_i atteignable. Ces variables X_i représenteront les variables du fait $state\_n$. Ensuite, nous définissons l'état initial de $r(p)$ qui contiendra le fait $state\_n$ dans lequel toutes les variables X_j correspondant à des valeurs dans l'état initial du partenaire $\mathcal{IK}$ sont respectivement remplacées par ces valeurs. Finalement, nous calculons le système de transition décrivant les communications de $r(p)$. Chaque étape est alors traduite en une transition ASLan qui reflète l'activité de communication du partenaire et l'évolution de ses connaissances, à l'aide des variables X_i. Les transitions peuvent aussi être gardées par des égalités entre variables.

Nous décrivons ici la procédure de génération du ASLan. Pour chaque partenaire $p = (n, \mathcal{IK}, c)$ nous construisons un système de transitions spécifié en ASLan qui représente une implémentation prudente de p. D'abord, nous considérons une liste $A = \langle a_1, \ldots, a_n \rangle$ de tous les sous-termes de c atteignables par p et tels que pour tout $1 \leq i < j \leq n$, $reach(p, a_i) \leq reach(p, a_j)$. Nous associons une variable fraîche pour tout élément dans A à travers la bijection $\sigma^{-1} : a_i \mapsto X_i$. et définissons le fait $state\_n$ pour p comme suit :

$$type(a_1) * \ldots * type(a_n) \rightarrow fact$$

Pour chaque réception $?msg(par(pa), opt(op), pld(m))$ dans c nous générons une transition ASLan τ avec le fait $iknows(\sigma^{-1}(m))$ dans son CD, ainsi que toutes les égalités apparaissant dans le système d'unification de l'étape et faisant intervenir les variables X_j. Remarquons que $\sigma^{-1}(m)$ est bien définie, car tout message m reçu par p est intuitivement atteignable par p. Pour toute émission $!msg(par(pa), opt(op), pld(m))$ dans c, nous générons une transition ASLan τ avec le fait $iknows(\sigma^{-1}(m))$ dans son CD. Remarquons là aussi que si c est exécutable alors tout message m envoyé par le partenaire et atteignale et donc $\sigma^{-1}(m)$ est bien définie.

4.4.2 Du ASLan à la servlet Tomcat

Nous réalisons chaque partenaire par une servlet Java, décrite partiellement ci-dessous :

```
class Partner extends HttpServlet{
```

```
  String[] X; int step;
  void dispatch(String msg){
   if(accept(msg)){step++; sendRemaining();}
  }
}
```

Le tableau X est un conteneur pour les variables X_i qui définissent l'état du rôle considéré. Le champ step est un entier sur l'indice de l'étape courante. Le cœur de la servlet est la méthode dispatch, qui gère les messages reçus. Cette méthode soumet un message reçu msg à la méthode accept (non représentée) qui retourne une réponse positive uniquement si *(i)* le champ step correspond à une activité de réception, et si *(ii)* msg passe le parsing et les vérifications possibles de l'étape step. Le cas échéant, la méthode accept met à jour le tableau X avec les nouvelles valeurs fraichement calculées. La méthode dispatch incrémente alors le compteur step puis appelle la méthode sendRemaining, non représentée ici, qui est en charge de l'envoi de tous les messages devant être envoyés par le rôle jusqu'à la prochaine étape de réception, ou jusqu'à la fin du scénario de communication.

Le code de la méthode accept peut être facilement généré depuis la spécification ASLan en transformant chaque condition d'égalité attachée à une transition soit à l'instanciation d'une nouvelle variable, soit à la vérification d'une certaine relation entre les variables déjà évaluées. Par exemple le code pour la méthode accept à l'étape step=3 est décrit ci-dessous :

```
if(msg.equals(X[10])){X[21] = msg; return(true);}
```

4.4.3 Gestion des sessions multiples

Notons que la servlet présentée dans la section 4.4.2 n'est pas capable de gérer différentes sessions du rôle correspondant.

Nous proposons ici de relaxer cette restriction en modifiant la servlet générée pour maintenir une liste de paires $(X, session)$ représentant les états des différentes sessions. Nous définissons par la suite une *politique de dispatch* aidant la servlet à acheminer les messages aux bonnes sessions. Tel que discuté dans [43] une stratégie naïve peut être appliqué si l'on est assuré que tous les messages reçus par un partenaire ont une *valeur distinctive* (par exemple un nonce) qui est atteignable par le partenaire à l'étape de réception et qui définit sans ambiguïté la session ciblée. Compte tenu de cette hypothèse et de la prudence des implémentations que nous générons, on peut utiliser un algorithme naïf (mais encore correct) pour dispatcher les messages : essayer toutes les sessions pour consommer un message entrant. En supposant que chaque réception conduit à calculer une valeur distinctive et que l'implémentation effectue toutes les vérifications de corrélation possibles sur les réceptions, nous pouvons être sûrs qu'au plus une session est candidate pour consommer un message entrant. Nous appelons cette propriété *Unique Dispatch* (UD).

La propriété UD est satisfaite pour les partenaires $P(A)$ et $P(B)$, mais pas par $P(S)$. Nous en concluons que le partenaire implémentant le rôle du serveur doit être sans état, tandis que ceux implémentant les rôles A et B peuvent, et doivent pour éviter de mélanger les messages de différentes sessions, être implémentés par des services à états.

4.5 Conclusion

Nous avons présenté dans ce chapitre des algorithmes permettant de compiler une trace de conversation entre services vers les implémentations prudentes de ces services. L'implantation de

ces algorithmes peut produire d'une part les spécifications ASLan des services impliqués (pouvant être validées dans la plate-forme AVANTSSAR) et d'autre part des servlets Java déployables dans des serveurs d'application et correspondant à des distributions exécutables des services.

Le principal avantage de notre approche reste sa généricité. En effet nous pouvons réutiliser nos résultats pour des théories équationnelles, aussi variées puissent-elles être, pourvu qu'elles satisfont l'hypothèse de localité.

Une limite de notre approche est sa restriction aux services séquentiels, *i.e.* ne contenant pas d'itérations ou de branchements conditionnés. Nous pensons néanmoins qu'il est possible de relaxer cette restriction pour compiler des services plus généraux. Dans ce sens, une première direction serait d'intégrer des branchements conditionnés par des égalités ou des inégalités entre sous-messages échangés par un service. Notre approche peut parfaitement se généraliser à ce cas précis, en représentant les activités de communication du service par un graphe acyclique dirigé (rendant compte des branchements) et en calculant pour chaque étape dans le graphe l'ensemble des traitements locaux devant être effectués par le service. Nous pensons qu'il est aussi possible d'étendre notre approche pour traiter des services avec des traitements itératifs. Un problème à résoudre serait alors de distinguer les messages reçus (ou calculés depuis les messages reçus) et correspondants à différents pas de l'itération.

Chapitre 5

Conclusions

Sommaire

5.1 L'orchestrateur AVANTSSAR appliqué aux études de cas

Dans cette section nous décrivons le traitement de certaines études de cas du projet AVANTSSAR par la plate-forme de validation d'AVANTSSAR ainsi que la génération prudente des médiateurs correspondants. Les descriptions détaillées des études de cas considérées sont consignées dans l'annexe B.

L'orchestrateur d'AVANTSSAR a été appliqué à plusieurs études de cas : la signature mutuelle d'un contrat numérique (DCS), un portail d'enchères publiques (PB) et un processus d'immatriculation de voiture à distance (CRP). Ces études de cas ne peuvent pas être manipulées par d'autres outils, car les messages échangés par les les services sont trop complexes. En effet ces messages sont non atomiques et construits avec des primitives cryptographiques et nécessitent donc une certaine adaptation automatique. Par exemple, l'orchestrateur a généré automatiquement un serveur de sécurité dans l'étude de cas de la signature mutuelle d'un contrat et a été également en mesure de générer le comportement pour les appels d'offres publics. Dans l'étude de cas de l'immatriculation de voitures à distance, un processus proposé par Siemens, l'orchestrateur a été en mesure de faire face à des contraintes supplémentaires imposées par les stratégies d'autorisation des services disponibles, spécifiées par un ensemble de clauses de Horn.

5.1.1 Signature mutuelle d'un contrat électronique (DCS)

Dans cette étude de cas deux signataires désirent cosigner électroniquement un même contrat en accédant à un serveur de sécurité spécialisé. Pour mener à bout sa tâche, le serveur dispose une communauté de services effectuant des tâches comme l'horodatage, la vérification de certificats électroniques et l'archivage à long-terme.

Les signataires se manifestent séquentiellement pour signer le contrat auprès du serveur de sécurité. Le serveur leur indique la politique à respecter pour la signature électronique et collecte leurs signatures respectives avant de procéder à leur vérification, leur horodatage et enfin leur archivage.

Le scénario de composition que nous avions choisi était de régénérer l'entité centrale de l'étude de cas : le serveur de sécurité. Plus de détails sur cette étude de cas sont donnés en annexe dans la section B.1.

5.1.2 Gestion d'enchères publiques (PB)

Cette étude de cas décrit une application web permettant de gérer l'émission d'un appel d'offre publique, de gérer la collecte des participations et enfin de décider de l'issue de l'appel d'offre. L'étude de cas fait intervenir, un processeur de l'appel d'offre, un portail web, un gestionnaire des enchères, un responsable technique, un responsable financier ainsi que deux participants aux enchères.

Le scénario de composition retenu était celui de régénérer le processeur de l'appel d'offre, l'entité centrale dans l'étude de cas.

5.1.3 Immatriculation d'un véhicule en ligne (CRP)

Cette étude de cas décrit l'immatriculation en ligne d'un véhicule par un citoyen. Il fait intervenir des employés virtuels avec une certaine hiérarchisation de leurs rôles, des autorités de certifications ainsi que des services de stockage.

Comme pour les autres études de cas considérées, nous avons là aussi considéré régénération de l'entité centrale qui permet de satisfaire la requête du citoyen, en mettant à contribution la communauté de services existants.

5.1.4 Temps d'exécution

Nous présentons ici les temps de réponses des outils relatifs à deux séries d'expérimentation.

- Dans la première série nous consignons les temps nécessaires à l'orchestrateur pour résoudre le problème d'orchestration. Ces temps concernent le calcul du médiateur ainsi que la génération de sa spécification ASLan.
- Dans la deuxième, nous régénérons les spécifications ASLan et les distributions exécutables (servlets Java) de tous les services intervenant dans l'orchestration (y compris le client) à partir de la trace de conversation retournée par l'orchestrateur. Ceci nous permet, entre autres, de vérifier le bon fonctionnement des servlets générées.

Composition de services

La table 5.1 illustre les résultats obtenus par l'orchestrateur AVANTSSAR pour les études de cas décrites plus haut. CS est le nom de l'étude de cas, NP est le nombre services impliqués dans la composition et OT le temps nécessaire à l'orchestrateur pour retourner la spécification ASLan du le médiateur.

Génération de services exécutables

La table 5.2 illustre les résultats obtenus pour les études de cas citées plus haut et pour une version services du protocoles NSPK. CS est le nom de l'étude de cas, $SIZE$ est la somme

TAB. 5.1 – Temps d'exécutions de l'orchestrateur

CS	NP	OT
DCS	7	1 s
PB	8	2 s
CRP	8	2 s

TAB. 5.2 – Temps d'exécutions du générateur de services

CS	SIZE	AGT	SGT	AET	SET
NSPK	166	163 ms	1 s	708 ms	615 ms
DCS	340	659 ms	2 s	3 m	1 s
PB	596	4 s	2 s	18 m	1 s
CRP	790	3 s	3 s	4 m	2 s

de toutes les tailles DAG des partenaires dans l'étude de cas, *AGT* (resp. *SGT*) est le temps nécessaire de générer les spécifications ASLan (resp. les servlets Java) correspondant à l'étude de cas et finalement *AET* (resp. *SET*) est le temps nécessaire à l'exécution des spécifications ASLan (resp. le temps nécessaire à l'exécution des servlets Java).

Nous notons ici que *AET* ne dépend pas exclusivement de la taille de l'étude de cas. En effet CL-Atse devine d'abord toutes les entrelacements possibles de la communication entre les rôles. Ce n'est pas le cas pour les *SET* car les servlets suivent un seul et unique entrelacement déjà défini.

5.2 Rappel des contributions

Dans cette thèse, nous contribuons à la recherche en sécurité et fiabilité des systèmes informatiques, l'une des priorités scientifiques pour la recherche sur Internet. Les nouvelles technologies numériques soulèvent de nombreuses questions relatives à la sécurité : la confidentialité, la confiance, l'authentification et l'identification, la certification, la protection des données ainsi que la traçabilité. Le niveau de confiance que l'utilisateur a dans ces technologies, qui comprennent des composants logiciels conséquents, est la clé de leur acceptation. Dans ce sens une validation de ces technologies pour garantir leur fonctionnement correct est nécessaire à travers l'utilisation de méthodes de développement sécurisé (les langages formels, les mathématiques, les preuves automatisées ainsi que les logiciels de certification automatique des composants).

Nous pensons que la plateforme de validation AVANTSSAR peut être exploitée dans ce sens à la fois pour l'industrie au niveau des applications et à des fins éducatives au niveau logique. Par ailleurs, les langages et les outils qui composent la plate-forme peuvent être utilisés indépendamment dans d'autres projets de recherche nationaux et internationaux. Par exemple, il y a des travaux en cours pour l'utilisation des techniques d'orchestration et d'analyse de la sécurité de la plateforme de validation AVANTSSAR dans le projet Nessos (FP7 project NESSOS [49]).

Nous avons montré que :

- L'orchestrateur AVANTSSAR propose une nouvelle approche entièrement automatique pour sécuriser la composition des services. Cet outil contribue à réduire le temps de conception des services web, en facilitant la prise en compte de la complexité de leurs politiques de sécurité et de leurs flux de messages.
- Le compilateur de conversations que nous présentons dans ce travail fournit un code facilement déployable et exécutable pour les nouveaux services composés obtenus.

– L'outil de validation CL-Atse offre la possibilité de valider les aspects de sécurité de nombreux services (composés) avant qu'ils ne soient effectivement déployés et exécutés.

5.3 Travaux futurs

Nous pensons que CL-Atse et l'orchestrateur AVANTSSAR ont besoin d'être davantage développés et appliqués à des cas d'études plus complexes de l'industrie. Dans un premier lieu le travail présenté dans cette thèse peut compléter les travaux et les méthodes existants pour le raisonnement automatisé sur la sécurité, des services et des applications web, et être très utile aux entreprises et organisations travaillant sur leur développement. Au-delà nous considérons aussi la possibilité de résoudre des problèmes de composition plus complexes, où le comportement du médiateur n'est plus forcément séquentiel, en autorisant des branchements et des boucles itératives conditionnés. En effet l'orchestrateur permet actuellement de découvrir tous les comportements séquentiels du médiateur. Nous pensons qu'il reste possible de factoriser tous ces comportements pour décrire son comportement global avec une représentation de toutes les branches d'exécution.

Par ailleurs, pour une plus large adoption, les outils doivent être mis en conformité aux normes et standards des services web. Par conséquent, nous devons développer des traducteurs de normes depuis et vers le langage ASLan. Nous avons exploré des pistes dans ce sens, notamment en traduisant automatiquement des spécifications de processus BPEL séquentiels vers ASLan. Cette traduction transforme les schémas XSD décrivant les messages XML échangés par le processus BPEL, en des termes du premier ordre. Elle fait pour l'instant abstraction des propriétés de sécurité des services web utilisés par le processus BPEL (ou de celles exigées pour le processus BPEL lui-même). Dans le cas particulier où ces politiques de sécurité adresse le niveau message, nous pensons qu'il est possible de les prendre en compte automatiquement en analysant leurs impacts sur les messages échangés par le processus.

Annexe A

Preuve du théorème 3

Sommaire

Le théorème qui est prouvé dans ce chapitre caractérise la prudence de la spécification opérationnelle générée pour un service. Son argument principal est la *localité* [42] de la théorie $\mathcal{E}_{XML}$. Cette propriété nous permet de résoudre le problème d'atteignabilité générale, en considérant uniquement sa restriction aux sous-termes d'un strand donné. D'un côté nous obtenons alors une implémentation pour le service, si le service est exécutable. En effet si le terme t modélisant un message à émettre à une étape *step* par le service est atteignable par son strand à l'étape *step*, alors le théorème donne une caractérisation d'un contexte permettant effectivement de le construire ou de l'extraire depuis les connaissances courantes du service. D'un autre côté, cette implémentation est prudente, parce qu'elle procèdera à tous les tests possibles : à chaque fois qu'une valeur déjà calculée (une partie dans un message reçu, par exemple), peut être recalculée d'une manière différente. Comme plus haut la propriété de localité permet de trouver un générateur fini, de tous les tests possibles à effectuer par le service au fil des messages qu'il reçoit et des nouvelles valeurs qu'il peut en extraire.

Nous remettons le théorème dans son cadre : Soit s un strand de longueur n. Pour tout i dans $\{1, \ldots, n\}$, nous rappelons le système d'unification suivant avec les variables $\{x_1, \ldots, x_i\} \cup \{y_t \mid t \in RST_i(s)\}$

$$\mathcal{U}_i(s) = \bigcup_{t \in RST_i(s)} \left\{ C_{min}(s,t) \stackrel{?}{=} C \mid C \in \mathcal{C}_i(s,t) \setminus \{C_{min}(s,t)\} \right\}$$

Dans la suite $\mathcal{U}_n(s)$ et aussi noté $\mathcal{U}(s)$.

Theorem 3. *Soit s un strand de longueur n. Pour toute étape $1 \leq i \leq n$ soit $t_1, \ldots, t_{k(i)}$ l'énumeration des éléments dans $RST_i^{new}(s)$ telle que :*

$$C_{min}(s,t_1) <_{R(s)} \ldots <_{R(s)} C_{min}(s,t_{k(i)})$$

On définit :

- $\tau_{s,i} = \{y_{t_1} \mapsto C_{min}(s,t_1)\} \circ \ldots \circ \left\{ y_{t_{k(i)}} \mapsto C_{min}(s,t_{k(i)}) \right\}$
- $\overline{\tau}_{s,i} = \tau_{s,1} \circ \ldots \circ \tau_{s,i}$

Pour toute étape i dans $\{1, \ldots, n\}$ nous avons :

1. *le contexte* $C_{min}(s,t)\overline{\tau}_{s,i}$ *s'évalue à* t *sur* s^i *pour tout* t *dans* $RST_i(s)$;
2. $\mathcal{U}_i(s)\overline{\tau}_{s,i}$ *est une base finie de* s^i.

Dans la suite x_i est noté e_i quand $s_i = !m_i$ et r_i sinon. On note aussi X_E l'ensemble des variables $\{x_e \mid e \in E\}$.

A.1 Preuve du résultat 1.

Pour prouver le résultat 1. nous procédons par induction sur l'ensemble des termes atteignables à une étape donnée. L'intuition est que chaque sous-terme atteignable est calculé pour la première fois par une séquence d'opérations sur les sous-termes atteignables à une étape antérieure ou égale.

Nous remarquons que pour toute étape i telle que $s_i = !m_i$ nous avons $RST_i^{new}(s) = \emptyset$. Par conséquent la substitution $\tau_{s,i}$ est l'identité. Nous restreignons dons la preuve aux étapes de réception dans $\overline{s^i}$. Soit i une étape de réception dans $\overline{s^i}$. On note $i\uparrow$ la plus grande étape de réception antérieure à i dans $\overline{s^i}$. Pour tout m dans $\{1,\ldots,k(i)\}$ soit

$$\tau_{s,i}^m = \{x_{t_1} \mapsto C_{min}(s,t_1)\} \circ \ldots \circ \{x_{t_m} \mapsto C_{min}(s,t_m)\}$$

. Nous définissons alors les propriétés suivantes :

$P_i^1(m)$: $\mathrm{Var}(\mathrm{Img}(\overline{\tau}_{s,i\uparrow} \circ \tau_{s,i}^m)) \subseteq \{x_1,\ldots,x_i\}$;

$P_i^2(m)$: $C_{min}(s,t)\overline{\tau}_{s,i\uparrow} \circ \tau_{s,i}^m =_{\mathcal{E}_{XML}} t$ pour tout t dans $RST_{i\uparrow}(s) \cup \{t_1,\ldots,t_m\}$;

$P_i(m)$: $P_i^1(m)$ et $P_i^2(m)$ sont valides.

Soit alors P_i la propriété définie par : $P_i(m)$ est valide pour tout m dans $\{1,\ldots,k(i)\}$. On propose de prouver P_i par induction sur les étapes de réception i dans $\overline{s^i}$. Si i est la première étape de réception dans $\overline{s^i}$ nous devons prouver ce qui suit pour tout m dans $\{1,\ldots,k(i)\}$:

$P_i^1(m)$: $\mathrm{Var}(\mathrm{Img}(\tau_{s,i}^m)) \subseteq \{x_i\}$;

$P_i^2(m)$: $C_{min}(s,t)\tau_{s,i}^m =_{\mathcal{E}_{XML}} t$ pour tout t dans $\{t_1,\ldots,t_m\}$.

On procède par induction sur m. Si $m = 1$ alors $C_{min}(s,t_1)$ est le plus petit contexte $R_i(s)$ par rapport à $<_{R(s)}$ d'où $C_{min}(s,t_1) = x_i$ et $\tau_{s,i}^1 = \{x_{t_1} \mapsto x_i\}$, ce qui donne $C_{min}(s,t_1)\tau_{s,i}^1 = x_i$ et finalement $P_i(1)$. Supposons maitenant que $m > 1$ et que $P_i(m')$ soit vraie pour toutes les étapes de réception m' telles que $1 < m' < m \leq k(i)$ et prouvons $P_i(m)$. Nous remarquons que $C_{min}(s,t_m)$ ne peut pas être réduit à une variable. En effet, puisque $C_{min}(s,t_m) \in R_i(s)$, la seule possibilité est x_i ce qui aboutit à la contradiction $t_m = t_1$. Il existe donc un symbole public f et $t'_1,\ldots,t'_{arity(f)}$ tels que $\left\{t'_1,\ldots,t'_{arity(f)}\right\} \subseteq RST_i(s)$ et vérifiant $C_{min}(s,t_m) = f(x_{t'_1},\ldots,x_{t'_{arity(f)}})$. Nous avons alors $C_{min}(s,t_m)\tau_{s,i}^m = f(x_{t'_1}\tau_{s,i}^m,\ldots,x_{t'_{arity(f)}}\tau_{s,i}^m)$. Par définition de $<_{R(s)}$ nous pouvons écrire $x_{t'_1},\ldots,x_{t'_{arity(f)}} <_{R(s)} x_{t_m}$ et comme $\{x_{t_m} \mapsto C_{min}(s,t_m)\}$ ne change pas les $x_{t'_1},\ldots,x_{t'_{arity(f)}}$ et que $\tau_{s,i}^m = \tau_{s,i}^{m-1} \circ \{x_{t_m} \mapsto C_{min}(s,t_m)\}$. nous pouvons alors écrire $C_{min}(s,t_m)\tau_{s,i}^m = f(x_{t'_1}\tau_{s,i}^{m-1},\ldots,x_{t'_{arity(f)}}\tau_{s,i}^{m-1})$. Par l'hypothèse d'induction nous avons d'abord $\mathrm{Var}(x_{t'_k}\tau_{s,i}^{m-1}) \subseteq \{x_i\}$ pour tout $1 \leq k \leq arity(f)$ et donc nous pouvons écrire $\mathrm{Var}(f(x_{t'_1}\tau_{s,i}^{m-1},\ldots,x_{t'_{arity(f)}}\tau_{s,i}^{m-1})) \subseteq \{x_i\}$ et nous obtenons finalement $\mathrm{Var}(C_{min}(s,t_m)\tau_{s,i}^m) \subseteq \{x_i\}$ ce qui prouve $P_i^1(m)$. D'un autre côté, nous avons $1 \leq k \leq arity(f)$, si $j(k)$ est tel que $t'_k = t_{j(k)}$. Alors nous avons $j(k) < m$ et $x_{t'_k}\tau_{s,i}^{j(k)} = C_{min}(s,t'_k)\tau_{s,i}^{j(k)}$ puisque par définition $x_{t'_k}$ n'apparait jamais dans $C_{min}(s,t'_k)$. Par définition $\left\{x_{t_{j(k)+1}} \mapsto C_{min}(s,t_{j(k)+1})\right\} \circ \ldots \circ$

$\{x_{t_m} \mapsto C_{min}(s,t_m)\}$ ne change pas $x_{t'_k}\tau_{s,i}^{j(k)}$ et $C_{min}(s,t'_k)\tau_{s,i}^{j(k)}$ d'où $x_{t'_k}\tau_{s,i}^{m-1} = C_{min}(s,t'_k)\tau_{s,i}^{m-1}$. Finalement nous obtenons $C_{min}(s,t_m)\tau_{s,i}^{m} = f(C_{min}(s,t'_1)\tau_{s,i}^{m-1},\ldots,C_{min}(s,t'_{arity(f)})\tau_{s,i}^{m-1})$ et par application des deux contextes dans la dernière équation sur $\overline{s^i}$ nous avons

$$C_{min}(s,t_m)\tau_{s,i}^{m}\cdot\overline{s^i} =_{\mathcal{E}_{XML}} f(t'_1,\ldots,t'_{arity(f)}) =_{\mathcal{E}_{XML}} t_m$$

ce qui prouve $P_i^2(m)$. Avec $P_i^1(m)$ prouvée plus haut, nous avons P_i ce qui prouve le cas de base de l'induction sur les étapes de réception i. Maintenant, soit i une étape de réception différente de la première dans $\overline{s^i}$. Supposons que $P_{i'}$ est vraie pour tout $1 \leq i' < i \leq n$ et prouvons P_i. Par hypothèse $P_{i\uparrow}$ est vraie ce qui donne $P_{i\uparrow}(k(i\uparrow))$, *i.e.*

$P_{i\uparrow}^1(k(i\uparrow))$: $\mathrm{Var}(\mathrm{Img}(\overline{\tau}_{s,i\uparrow})) \subseteq \{x_1,\ldots,x_{i\uparrow}\}$;

$P_{i\uparrow}^2(k(i\uparrow))$: $C_{min}(s,t)\overline{\tau}_{s,i\uparrow} =_{\mathcal{E}_{XML}} t$ pour tout t dans $RST_{i\uparrow}(s)$.

Par application d'un raisonnement analogue au cas de base nous avons P_i ce qui termine la preuve. En particulier $P_i(k(i))$ est vraie ce qui correspond au résultat 1. du théorème 3.□

Dans la suite nous utilisons l'ordre induit par la relation de raffinement présentée dans la section 4.2.3, pour caractériser une base finie minimale, résumant toutes les conditions à vérifier à la réception d'un message.

A.2 Preuve du résultat 2.

Pour prouver le résultat 2. nous introduisons la *complétion*, une construction stable sur les strands. Étant donné s nous calculons une base finie de sa complétion et nous montrons comment on peut en dériver une pour s. L'intuition derrière est que la complétion conserve les sous-termes atteignables et les bases finies.

Définition 12 *Soit s un strand de longueur n et soit $m_{i_1},\ldots,m_{i_k}$ labellisés par ? dans s. Pour tout $1 \leq j \leq k$ et pour tout t dans $RST_{i_j}^{new}(s)$ nous considérons $C_{i_j}(t)$, un contexte d'extraction pour t de s^{i_j}. Soit s' un strand de longueur n' avec $i_1 \leq n'$, la complétion de s' par rapport à s notée $\overline{s'}^s$ est le strand obtenu à partir de s' en remplaçant s'_{i_l} pour tout $i_1 \leq i_l \leq n'$ par le strand $?s'_{i_l}?(C_{i_l}(t_1^l)\cdot s')\downarrow \ \ldots\ ?(C_{i_l}(t_{n^l}^l)\cdot s')\downarrow$ où $\{t_1^l,\ldots,t_{n^l}^l\} = RST_{i_l}^{new}(s)$.*

Dans la suite, $\overline{s^i}^s$ est aussi noté $\overline{s^i}$ pour tout $1 \leq i \leq n$.
Pour tout terme t dans $ST(s)$ nous définissons $step(s,t)$ (resp. $index(s,t)$) comme étant i s'il existe $1 \leq i \leq n$ tel que $t \in RST_i^{new}(s)$ (resp. $s_i = ?t$) et ∞ sinon. Par définition tout élément t dans $RST_i(s)$ est reçu au moins une fois dans $\overline{s^i}$ d'pù $index(\overline{s^i},t)$ est bien défini pour tout t dans $RST(s)$.

Théorème 4 *Soit s un strand de longueur n et $\overline{\upsilon}_{s,i}$ la substitution définie par*

$$\left\{y_t \mapsto x_{index(\overline{s^i},t)}\right\}_{t\in RST_i(s)} \cup \left\{x_j \mapsto x_{length(\overline{s^{j-1}})+1}\right\}_{1\leq j\leq i}$$

Pour toute étape i dans $\{1,\ldots,n\}$, $\mathcal{U}_i(s)\overline{\upsilon}_{s,i}$ est une base finie de $\overline{s^i}$.

A.2.1 Preuve du théorème 4

Par définition de la base finie ceci se ramène à prouver pour tout s' :

$$P_{\overline{s^i}} \subseteq P_{s'} \Longleftrightarrow \mathcal{U}_i(s)\overline{v}_{s,i} \subseteq P_{s'}$$

Par construction $\mathcal{U}_i(s)\overline{v}_{s,i} \subseteq P_{\overline{s^i}}$ et donc la partie $\Rightarrow$ est triviale. To prove the $\Leftarrow$ part we reason by contradiction : assume there exists a strand s' such that $\mathcal{U}_i(s)\overline{v}_{s,i} \subseteq P_{s'}$ and $P_{\overline{s^i}} \not\subseteq P_{s'}$. Let $E = \left\{\{C, C'\} \in P_{\overline{s^i}} \mid \{C, C'\} \notin P_{s'}\right\}$, we denote by $<_1$ the recursive path order defined over first-order terms on the signature $\mathcal{F}'_p = \mathcal{F}_p \cup \{x_1, \ldots, x_n\}$, by considering the total precedence $<_{\mathcal{F}'_p}$ defined as follow :

- $x_1 \prec_{XML} x_2 \prec_{XML} \ldots scrypt$;
- $scrypt \prec_{XML} crypt \prec_{XML} sign \prec_{XML} node_a^n$ pour tout $node_a^n$;
- $node_a^n \prec_{XML} node_{a'}^{n'}$ si et seulement si $a < a'$ ou $n \prec_{string} n'$;
- $node_a^n \prec_{XML} sdcrypt \prec_{XML} dcrypt \prec_{XML} verif \prec_{XML} child_{\frac{i}{a'}}^{n'}$ pour tout $node_a^n$ et $child_{\frac{i}{a'}}^{n'}$;
- $child_{\frac{1}{a}}^{n} \prec_{XML} \ldots \prec_{XML} child_{\frac{a}{a}}^{n}$;
- $child_{\frac{i}{a}}^{n} \prec_{XML} child_{\frac{i}{a'}}^{n'}$ si et seulement si $a < a'$ ou $n \prec_{string} n'$

Nous notons $<_2$ l'extension multi-ensemble $<_1$ sur les paires de termes sur la signature $\mathcal{F}'_p$. Par hypothèse, E n'est pas vide et $<_2$ est bien fondé, donc E admet un élément minimal par rapport à $<_2$ noté $\left\{C_1^0, C_2^0\right\}$ Dans la suite nous prouvons que $\left\{C_1^0, C_2^0\right\} \notin E$ pour obtenir une contradiction. Nous commençons par prouver les propriétés suivantes :

Lemme 1 *$C_1^0 = C_1^0\downarrow$ and $C_2^0 = C_2^0\downarrow$*

Pour simplifier la preuve nous supposons que tous les strands s considérés sont en forme normale, *i.e.* $s_i = s_i\downarrow$ pour chaque s_i.

Preuve du lemme 1

On raisonne par l'absurde. Supposons que $C_1^0 \neq C_1^0\downarrow$. Alors $\left\{C_1^0\downarrow, C_2^0\right\}$ est un élément de E et comme $C_1^0\downarrow <_1 C_1^0$ nous avons $\left\{C_1^0\downarrow, C_2^0\right\} <_2 \left\{C_1^0, C_2^0\right\}$ ce qui contredit la minimalité de $\left\{C_1^0, C_2^0\right\}$.□

Lemme 2 *Pour tout C, sous-terme strict de C_1^0 (symétriquement de C_2^0) si $C \neq x_i$ pour tout i dans $\{1, \ldots, n\}$ alors $C \cdot \overline{s^i} \not=_{\mathcal{E}_{XML}} x_i \cdot \overline{s^i}$ pour tout i dans $\{1, \ldots, n\}$.*

Preuve du lemme2

On raisonne par l'absurde. Supposons qu'il existe un sous-terme C de C_1^0 et i dans $\{1, \ldots, n\}$ tels que $C \cdot \overline{s^i} =_{\mathcal{E}_{XML}} x_i \cdot \overline{s^i}$ et $C \neq x_i$. Alors seul l'un de ces deux cas est vrai :

- Si $C \cdot s' =_{\mathcal{E}_{XML}} x_i \cdot s'$: Soit p_0 une position dans C_1^0 telle que, $C_1^0|_{p_0} = C$ et soit $C^0 = C_1^0[x_i]_{p_0}$. Par construction de C^0, $\left\{C^0, C_2^0\right\}$ est un élément de E. Par monotonie de $<_1$ et comme $x_i <_1 C$ nous avons $C_0 <_1 C_1^0$ et donc $\left\{C^0, C_2^0\right\} <_2 \left\{C_1^0, C_2^0\right\}$, ce qui contredit la minimalité de $\left\{C_1^0, C_2^0\right\}$.
- Si $C \cdot s' \not=_{\mathcal{E}_{XML}} x_i \cdot s'$: par hypothèse (C, x_i) est un élément de E et comme $C, x_i <_1 C_1^0$ nous avons $\left\{C, x_i\right\} <_2 \left\{C_1^0, C_2^0\right\}$ ce qui contredit la minimalité de $\left\{C_1^0, C_2^0\right\}$.

Nous prouvons le lemme 2 pour C_2^0 symétriquement.□

Lemme 3 *Pour tout C, C' sous-termes stricts de C_1^0 (symétriquement C_2^0), $C.s \not=_{\mathcal{E}_{XML}} C'.s$.*

Preuve du lemme3

On raisonne par l'absurde : supposons qu'il existe C, C', deux sous-termes stricts et distincts de C_1^0 et tels que : $C.s =_{\mathcal{E}_{XML}} C'.s$. Considérons alors l'ensemble

$$F = \{\{C, C'\} \text{ sous-termes stricts et distincts de } C_1^0,\ C.s =_{\mathcal{E}_{XML}} C'.s\}$$

Par hypothèse F n'est pas vide et comme $<_2$ est bien fondé, il existe un élément miniaml $\{C_0, C_0'\}$ dans F par rapport $<_2$. Alors seul l'un de ces deux cas est vrai :

- Si $C_0 \cdot s' \neq_{\mathcal{E}_{XML}} C_0' \cdot s'$: par hypothèse nous avons $\{C_0, C_0'\}$ est dans E et $\{C_0, C_0'\} <_2 \{C_1^0, C_2^0\}$. Nous obtenons alors une contradiction sur la minimalité de $\{C_1^0, C_2^0\}$.
- $C_0 \cdot s' =_{\mathcal{E}_{XML}} C_0' \cdot s'$: comme $<_1$ est total, soit $C_{min} = min\{C_0, C_0'\}$ et $C_{max} = max\{C_0, C_0'\}$ et soit p_0 une position dans C_1^0 telle que $C_1^0|_{p_0} = C_{max}$ et $C_1^{0'} = C_1^0[C_{min}]_{p_0}$. Par construction de $C_1^{0'}$ nous avons $\{C_1^{0'}, C_2^0\}$ est dans E. Par monotonie de $<_1$ nous avon saussi $C_1^{0'} <_1 C_1^0$ et donc $\{C_1^{0'}, C_2^0\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.

Nous prouvons le lemme 3 for C_2^0 symétriquement.□

Lemme 4 *Pour tout C_1, C_2 qui sont soit : respectivement sous-terme strict de C_1^0 et sous-terme de C_2^0 soit respectivement sous-terme de C_1^0 et sous-terme strict de C_2^0, si $C_1 \neq C_2$ alors $C_1 \cdot \overline{s^i} \neq_{\mathcal{E}_{XML}} C_2 \cdot \overline{s^i}$.*

Preuve du lemme 4

On raisonne par l'absurde : Supposons l'existence de C_1, C_2 satisfaisant l'hypothèse du lemme et tels que $C_1 \neq C_2$ et $C_1 \cdot \overline{s^i} =_{\mathcal{E}_{XML}} C_2 \cdot \overline{s^i}$. Soit G l'ensemble de paires de contexte $\{C_1, C_2\}$, où C_1, C_2 sont soit respectivement sous-terme strict de C_1^0 et sous-terme de C_2^0 soit respectivement sous-terme de C_1^0 et sous-terme strict de C_2^0 et tels que $C_1 \neq C_2$ et $C_1 \cdot \overline{s^i} =_{\mathcal{E}_{XML}} C_2 \cdot \overline{s^i}$. Par hypothèse G n'est pas vide et comme $<_2$ est bien fondé, il existe $(C_1^{0'}, C_2^{0'})$, un élément minimal de G par rapport à $<_2$. Seul l'un de ces deux cas est vrai :

- If $C_1^{0'} \cdot s' \neq_{\mathcal{E}_{XML}} C_2^{0'} \cdot s'$: Alors $(C_1^{0'}, C_2^{0'})$ est un élément de E et $(C_1^{0'}, C_2^{0'}) <_2 \{C_1^0, C_2^0\}$. On obtient donc une contradiction sur la minimalité de $\{C_1^0, C_2^0\}$.
- Si $C_1^{0'} \cdot s' =_{\mathcal{E}_{XML}} C_2^{0'} \cdot s'$: et comme $<_1$ est bien fondé, soit $C'_{min} = min(C_1^{0'}, C_2^{0'})$, $C'_{max} = max(C_1^{0'}, C_2^{0'})$ et C^0_{max}, C^0_{min} les contextes dans $\{C_1^0, C_2^0\}$ contenant respectivement C'_{max}, C'_{min}. Soit p_0 une position telle que $C^0_{max}|_{p_0} = C'_{max}$ et $C_0 = C^0_{max}[C'_{min}]_{p_0}$. Par construction, $\{C_0, C^0_{min}\}$ est un élément de E et $\{C_0, C^0_{min}\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.□

Lemme 5 *Pour tout C sous-terme strict de C_1^0 (symétriquement de C_2^0), $C \cdot \overline{s^i} = (C \cdot \overline{s^i})\downarrow$.*

Preuve du lemme 5

On raisonne par l'absurde. Supposons qu'il existe C un sous-terme strict de C_1^0 tel que $C \cdot \overline{s^i} \neq (C \cdot \overline{s^i})\downarrow$. Considérons l'ensemble

$$F = \{C, \text{ sous-terme strict de } C_1^0,\ C.s \neq (C.s)\downarrow\}$$

Par hypothèse F n'est pas vide et comme $<_1$ est bien fondé, il existe un élément minimal C_0 de F par rapport à $<_1$. Pour tout i dans $\{1, \ldots, n\}$, nous avons $C_0 \neq x_i$ car sinon $x_i \cdot \overline{s^i} = (x_i \cdot \overline{s^i})\downarrow$ ce qui contredit les hypothèses et donc il existe k contexts $C_0^1, \ldots, C_0^k$ et f un symbole publique

d'arité k, tel que $C_0 = f(C_0^1, \ldots, C_0^k)$. Pour tout j dans $\{1, \ldots, k\}$ nous avons $C_0^j \cdot \overline{s^i} = (C_0^j \cdot \overline{s^i}) \downarrow$, car sinon, comme $C_0^j <_1 C_0$ et comme C_0^j est un élément de F nous obtenons une contradiction avec la minimalité de C_0. D'où il existe j dans $\{1, \ldots, k\}$ tel que $(C_0 \cdot \overline{s^i}) \downarrow$ et un sous-terme de $C_0^j \cdot \overline{s^i}$. Alors seul l'un de ces deux cas est vrai :

- Il existe i danns $\{1, \ldots, n\}$ tel que $C_0^j = x_i$: Comme $(C_0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $C_0^j \cdot \overline{s^i}$ qui est à son tour sous-terme de $\overline{s^i}$, nous avons $(C_0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $\overline{s^i}$ et donc il existe l dans $\{1, \ldots, n\}$ tel que $(C_0 \cdot \overline{s^i}) \downarrow = x_l \cdot \overline{s^i}$. Seul l'un de ces sous-cas est vrai :
  - $(C_0 \cdot s') \downarrow \neq x_l \cdot s'$: alors $\{C_0, x_l\}$ est un élément de E et comme $C_0 <_1 C_1^0$ et $x_l <_1 C_1^0$ nous avons $\{C_0, x_l\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
  - $(C_0 \cdot s') \downarrow = x_l \cdot s'$: Soit p_0 une position telle que $C_1^0|_{p_0} = C_0$ et soit $C_1^{0\prime} = C_1^0[x_l]_{p_0}$. Par construction de $C_1^{0\prime}$, $\{C_1^{0\prime}, C_2^0\}$ est un élément de E et $\{C_1^{0\prime}, C_2^0\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
- Il existe m contextes $C_j^1, \ldots, C_j^m$ et g un symbole public d'arité m, tels que $C_0^j = g(C_j^1, \ldots, C_j^m)$: alors il existe q dans $\{1, \ldots, m\}$ tel que $C_j^q \cdot \overline{s^i} = (C_0 \cdot \overline{s^i}) \downarrow$. Alors seul l'un des ces sous-cas est vrai :
  - $(C_0 \cdot s') \downarrow \neq C_j^q \cdot s'$: alors $\{C_0, C_j^q\}$ est un élément de E et comme $C_0 <_1 C_1^0$ et $C_j^q <_1 C_1^0$ nous déduisons $\{C_0, C_j^q\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
  - $(C_0 \cdot s') \downarrow = C_j^q \cdot s'$: Soit p_0 une position telle que $C_1^0|_{p_0} = C_0$ et soit $C_1^{0\prime} = C_1^0[C_j^q]_{p_0}$. Par construction de $C_1^{0\prime}$, $\{C_1^{0\prime}, C_2^0\}$ est un élément E et $\{C_1^{0\prime}, C_2^0\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.

Nous prouvons le lemme 5 pour C_2^0 symétriquement.□

Supposons maintenant qu'il existe f_1, f_2 deux symboles publics d'arités respectives k_1, k_2, k_1 des contextes $C_{1,1}^0, \ldots, C_{1,k_1}^0$, k_2 et des contextes $C_{2,1}^0, \ldots, C_{2,k_2}^0$, tels que $C_1^0 = f_1(C_{1,1}^0, \ldots, C_{1,k_1}^0)$ et $C_2^0 = f_2(C_{2,1}^0, \ldots, C_{2,k_2}^0)$. On note t_1^0, t_2^0 respectivement les deux termes $C_1^0 \cdot \overline{s^i}, C_2^0 \cdot \overline{s^i}$ et pour tout i dans $\{1, 2\}$ et pour tout j dans $\{1, \ldots, k_i\}$, on note $t_{i,j}^0$ le terme $C_{i,j}^0 \cdot \overline{s^i}$. Nous avons alors : $t_1^0 = f_1(t_{1,1}^0, \ldots, t_{1,k_1}^0)$, $t_2^0 = f_2(t_{2,1}^0, \ldots, t_{2,k_2}^0)$ et en appliquant le lemme 5 nous avons pour tout i dans $\{1, 2\}$ et pour tout j dans $\{1, \ldots, k_i\}$, $t_{i,j}^0 = (t_{i,j}^0) \downarrow$. Alors seul l'un des ces sous-cas est vrai :

- $t_1^0 = (t_1^0) \downarrow$ et $t_2^0 = (t_2^0) \downarrow$: nous avons $t_1^0 =_{\mathcal{E}_{XML}} t_2^0$, et donc par unicité de la forme normale $f_1(t_{1,1}^0, \ldots, t_{1,k_1}^0) = f_2(t_{2,1}^0, \ldots, t_{2,k_2}^0)$ ce qui donne $f_1 = f_2$, $k_1 = k_2$ et pour tout i dans $\{1, \ldots, k_1\}$, $t_{1,i}^0 = t_{2,i}^0$. Ainsi par définition de $t_{1,i}^0, t_{2,i}^0$, nous avons $C_{1,i}^0 \cdot \overline{s^i} = C_{2,i}^0 \cdot \overline{s^i}$. Par le lemme 5, nous avons alors pour tout i dans $\{1, \ldots, k_1\}$, $C_{1,i}^0 = C_{2,i}^0$, et donc nous concluons que $C_1^0 = C_2^0$. Comme C_1^0, C_2^0 sont définies sur s', $C_1^0 \cdot s' = C_2^0 \cdot s'$ et donc $\{C_1^0, C_2^0\}$ n'est pas dans E ce qui contredit les hypothèses.
- $t_1^0 \neq (t_1^0) \downarrow$ et $t_2^0 \neq (t_2^0) \downarrow$: alors $C_2^0 <_1 C_1^0$ ou $C_1^0 <_1 C_2^0$. Si $C_1^0 <_1 C_2^0$ (l'autre cas $t_2^0 <_1 t_1^0$ se traite symétriquement). Alors il existe j dans $\{1, \ldots, k_1\}$ tel que $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $C_{1,j}^0 \cdot \overline{s^i}$. Seul l'un de ces cas est vrai :
  - il existe i dans $\{1, \ldots, n\}$ et tel que $C_{1,j}^0 = x_i$: comme $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $C_{1,j}^0 \cdot \overline{s^i}$ qui est aussi un sous-terme de $\overline{s^i}$, nous avons $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $\overline{s^i}$ et donc il existe l dans $\{1, \ldots, n\}$ tel que $(C_1^0 \cdot \overline{s^i}) \downarrow = x_l \cdot \overline{s^i}$. Seul l'un de ces sous-cas est vrai :
    - $(C_1^0 \cdot s') \downarrow \neq x_l \cdot s'$: Nous avons $\{C_1^0, x_l\}$ est un élément de E et $\{C_1^0, x_l\} <_2 \{C_1^0, C_2^0\}$

ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
  - $(C_1^0 \cdot s') \downarrow = x_l \cdot s'$: $\{x_l, C_2^0\}$ est un élément de E et $\{x_l, C_2^0\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
- Il existe m contextes $C_j^1, \ldots, C_j^m$ et g un symbole public d'arité m, tel que $C_0^j = g(C_j^1, \ldots, C_j^m)$: alors il existe q dans $\{1, \ldots, m\}$ tel que $C_j^q \cdot \overline{s^i} = (C_1^0 \cdot \overline{s^i}) \downarrow$. Seul l'un de ces sous-cas est vrai :
  - $(C_1^0 \cdot s') \downarrow \neq C_j^q \cdot s'$: $\left\{C_1^0, C_j^q\right\}$ est un élément de E et comme $C_j^q <_1 C_1^0 <_1 C_2^0$ nous avons $\left\{C_1^0, C_j^q\right\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
  - $(C_1^0 \cdot s') \downarrow = C_j^q \cdot s'$: alors $\left\{C_j^q, C_2^0\right\}$ est un élément de E et $\left\{C_j^q, C_2^0\right\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
- $t_1^0 \neq (t_1^0) \downarrow$ et $t_2^0 = (t_2^0) \downarrow$: (le cas $t_1^0 = (t_1^0) \downarrow$ et $t_2^0 \neq (t_2^0) \downarrow$ est traité symétriquement). Comme $t_1^0 \neq (t_1^0) \downarrow$, il existe j dans $\{1, \ldots, k\}$ tel que $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme $C_0^j \cdot \overline{s^i}$. Alors seul l'un de ces cas est vrai :
  - il existe i dans $\{1, \ldots, n\}$ tel que $C_0^j = x_i$: comme $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme $C_0^j \cdot \overline{s^i}$ qui est à son tour sous-terme de $\overline{s^i}$, nous avons $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme $\overline{s^i}$ et donc il existe l dans $\{1, \ldots, n\}$ tel que $(C_1^0 \cdot \overline{s^i}) \downarrow = x_l \cdot \overline{s^i}$. Alors seul l'un de ces sous-cas est vrai :
    - $(C_1^0 \cdot s') \downarrow \neq x_l \cdot s'$: $\{C_1^0, x_l\}$ est un élément de E et $\{C_1^0, x_l\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
    - $(C_1^0 \cdot s') \downarrow = x_l \cdot s'$: $\{x_l, C_2^0\}$ est un élément de E et $\{x_l, C_2^0\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
  - Il existe m contextes $C_j^1, \ldots, C_j^m$ et g un symbole public d'arité m, tel que $C_0^j = g(C_j^1, \ldots, C_j^m)$: alors il existe q dans $\{1, \ldots, m\}$ tel que $C_j^q \cdot \overline{s^i} = (C_1^0 \cdot \overline{s^i}) \downarrow$. Alors seul l'un de ces sous-cas est vrai :
    - $(C_1^0 \cdot s') \downarrow \neq C_j^q \cdot s'$: $\left\{C_1^0, C_j^q\right\}$ est un élément de E et comme $C_j^q \cdot \overline{s^i} = C_2^0 \cdot \overline{s^i}$, par application du lemme 4 nous avons $C_j^q = C_2^0$ et donc $\left\{C_1^0, C_j^q\right\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.
    - $(C_1^0 \cdot s') \downarrow = C_j^q \cdot s'$: alors $\left\{C_j^q, C_2^0\right\}$ est un élément de E et $\left\{C_j^q, C_2^0\right\} <_2 \{C_1^0, C_2^0\}$ ce qui contredit la minimalité de $\{C_1^0, C_2^0\}$.

  Supposons maintenant qu'il existe i dans $\{1, \ldots, n\}$, f_1 un sybmole public d'arité k_1 et k_1 contextes $C_{1,1}^0, \ldots, C_{1,k_1}^0$, tels que $C_1^0 = f_1(C_{1,1}^0, \ldots, C_{1,k_1}^0)$ et $C_2^0 = x_i$. On note t_1^0, t_2^0 respectivement les deux termes $C_1^0 \cdot \overline{s^i}, C_2^0 \cdot \overline{s^i}$ et pour tout j in $\{1, \ldots, k_1\}$, on note $t_{1,j}^0$ le terme $C_{1,j}^0 \cdot \overline{s^i}$. Nous avons alors $t_2 = (t_2) \downarrow$ et $t_1^0 = f_1(t_{1,1}^0, \ldots, t_{1,k_1}^0)$ et grâce au lemme 5 nous avons pour tout j dans $\{1, \ldots, k_1\}$, $t_{1,j}^0 = (t_{1,j}^0) \downarrow$. Alors seul l'un de ces cas est vrai :
  - $t_1^0 = (t_1^0) \downarrow$: comme t_2^0 est un sous-terme de $\overline{s^i}$ et $t_2^0 = t_1^0 = f_1(t_{1,1}^0, \ldots, t_{1,k_1}^0)$, alors pour tout j dans $\{1, \ldots, k_1\}$, $t_{1,j}^0$ est un sous-terme de $\overline{s^i}$ et donc il existe i_j dans $\{1, \ldots, n\}$, tel que $t_{1,j}^1 =_{\mathcal{E}_{XML}} x_{i_j} \cdot \overline{s^i}$, ce qui par définition de $t_{1,j}^1$ donne $C_{1,j}^0 \cdot \overline{s^i} =_{\mathcal{E}_{XML}} x_{i_j} \cdot \overline{s^i}$. Par application du lemme 2 nous avons alors $C_{1,j}^0 = x_{i_j}$, pour tout j dans $\{1, \ldots, k_1\}$ et donc $C_1^0 = f_1(x_{i_1} \ldots, x_{\tilde{i_{k_1}}})$. Comme $C_2^0 = x_i$, nous avons $\{C_1^0, C_2^0\}$ in Q_s et en utilisant les hypothèses nous avons $C_1^0 \cdot s' =_{\mathcal{E}_{XML}} C_2^0 \cdot s'$. D'où $\{C_1^0, C_2^0\}$ n'est pas un élément de E ce qui aboutit à une contradiction.
  - $t_1^0 \neq (t_1^0) \downarrow$: alors il existe j dans $\{1, \ldots, k\}$ tel que $(C_1^0 \cdot \overline{s^i}) \downarrow$ est un sous-terme de $C_0^j \cdot \overline{s^i}$. Seul l'un de ces cas est vrai :

- Il existe i' dans $\{1,\ldots,n\}$ tel que $C_0^j = x_{i'}$: comme $(C_1^0 \cdot \overline{s^i})\downarrow$ est un sous-terme de $C_0^j \cdot \overline{s^i}$ qui à son tour est un sous-terme de $\overline{s^i}$ alors $(C_1^0 \cdot \overline{s^i})\downarrow$ est un sous-terme de $\overline{s^i}$ et donc il existe l dans $\{1,\ldots,n\}$ tel que $(C_1^0 \cdot \overline{s^i})\downarrow = x_l \cdot \overline{s^i}$. Comme $x_l \cdot \overline{s^i} = x_i \cdot \overline{s^i}$ et par construction de $\overline{s^i}$ nous avons $l, i \leq n$ et donc par définition de s' nous avons aussi $x_l \cdot s' = x_i \cdot s'$ ce qui donne $C_1^0 \cdot s' =_{\mathcal{E}_{XML}} C_2^0 \cdot s'$ et donc $\{C_1^0, C_2^0\}$ n'est pas un élément de E ce qui contredit les hypothèses.
- Il existe m contextes $C_j^1,\ldots,C_j^m$ et g un symbole public d'arité m, tels que $C_0^j = g(C_j^1,\ldots,C_j^m)$: donc il existe q dans $\{1,\ldots,m\}$ tel que $C_j^q \cdot \overline{s^i} = (C_1^0 \cdot \overline{s^i})\downarrow$. Comme t_1^0 est un sous-terme de $\overline{s^i}$ alors il existe i' dans $\{1,\ldots,n\}$ tel que $C_j^q \cdot \overline{s^i} = x_{i'} \cdot \overline{s^i}$ et donc par application du lemme 2 nous avons $C_j^q = x_{i'}$, ce qui donne $x_{i'} \cdot \overline{s^i} = x_i \cdot \overline{s^i}$. Par construction de $\overline{s^i}$ nous avons $i', i \leq n$ et par définition de s' nous avons aussi $x_{i'} \cdot s' = x_i \cdot s'$ ce qui donne $C_1^0 \cdot s' =_{\mathcal{E}_{XML}} C_2^0 \cdot s'$ et donc $\{C_1^0, C_2^0\}$ n'est pas un élément de E ce qui contredit les hypothèses.

Nous concluons dans tous les cas que E est l'ensemble vide.□

Pour achever la preuve du résultat 2. nous posons enfin le dernier lemme :

Lemme 6 *Soit s un strand de longueur n. Pour toutes les étapes $1 \leq i \leq n$ nous avons $\mathcal{U}_i(s)\overline{\tau}_{s,i}$ est unr base finie de s^i si et seulement si $\mathcal{U}_i(s)\overline{v}_{s,i}$ est une base finie de $\overline{s^i}$.*

Le lemme 6 peut-être prouvé en remarquant que par construction de $\overline{s^i}$: le test $y_t\overline{v}_{s,i} \stackrel{?}{=} f(y_{t_1}\overline{v}_{s,i},\ldots,y_{t_k}\overline{v}_{s,i})$ est valide sur $\overline{s^i}$ si et seulement si le test $y_t\overline{\tau}_{s,i} \stackrel{?}{=} f(y_{t_1}\overline{\tau}_{s,i},\ldots,y_{t_k}\overline{\tau}_{s,i})$ est valide pour s^i.□

Annexe B

Description détaillée des études de cas industrielles traités

[8]

Sommaire

B.1 Digital Contract Signing

B.1.1 Scenario Purpose

The Digital Contract Signing case study represents a contract signing procedure carried out by two partners. These two participants have secure access to a trusted third party web site, a business portal, in order to digitally sign a contract.

First, the business portal application generates an electronic document corresponding to the terms of agreement between the two signers. This electronic contract is added to a digital case, called the proof record that will contain at the end of the signing procedure the fully signed contract. Then, the first signer accesses the business portal and views the contract. If he agrees

[8] Extraits de [10]

to the terms of the contract, he signs and sends it back to the business portal. The latter checks the generated signature and stores the signed contract if the check succeeded.

The second party connects to the web site, checks the status of the existing signature and then co-signs the contract after viewing it. In a similar way, the business portal checks this second signature. If the check succeeds then the contract is archived for long-term conservation after being timestamped.

The processing of proof elements like timestamping or archiving is delegated to another service, called the security server. To provide its services, the security server relies on three trusted parties : the PKI, the TimeStamper and the Archiver. They are responsible respectively for publishing certificate revocation lists (CRL), timestamping messages and archiving proof records.

B.1.2 High-level description

In this section, we describe the scenario of the digital contract signing (DCS) case study. We use the descriptions of Deliverable D5.1 [4] and D4.1 [8]. We detail these descriptions while focusing in particular on the structure of the different messages exchanged. As stated in previous deliverables, the scenario is composed of two phases. The first phase consists in the achievement of the signature procedure by the signature of the one-signed contract. The second one aims at completing the signature procedure by the second signer. These two phases involve the following participants :

- $Signer1$
  This participant asks to sign the contract that he had already agreed on with the second signer (according to the assumption made before starting the first phase). To do this, he sends a signature request to the Business Portal (participant $BusinessPortal$). When he gets back the contract and its parameters, he views it, signs it, and sends it to the Business Portal. He receives in return an acknowledgement of his signature.
- $Signer2$
  This participant plays the role of the second signer. He participates in the second phase of the contract signing scenario, that is, after the signature of the first signer and the verification that follows. The behaviour of this participant is similar to the first signer except that here, the second signer has to wait for a signed contract.
- $BusinessPortal$
  This participant represents the core of the digital signature procedure that allows two signers to digitally sign an already agreed contract. Indeed, the business portal is responsible for the generation of an electronic contract based on the information collected from two signers. It must also coordinate the signing procedure between the two signers while delegating the management of the proof record and the management of digital proof elements (such as verification of signatures, timestamping or archiving of proof records) to the security server. We assume that messages between this participant and other participants (the security server and the two signers) are exchanged over secure channels.
- $SecurityServer$
  This participant has two main roles in the signing procedure. First, it locally keeps a proof record related to the signing contract. At the end of the signing procedure, this record contains the contract and the two signatures. Second, the security server can be called on for different other tasks especially related to the management of digital proof elements like timestamping or archiving. These tasks are delegated to other participants, detailed below. We assume that communications between the security server and each of these participants

are assured over a secure channel.

To manage digital proofs like timestamping or verifying certificates, the security server delegates some tasks to a certain number of participants :

- *TimeStamper*
  The main service offered by this participant is to put a time-stamp on any input message sent by the security server. For this, the time stamper adds the current time to the received message, signs the result and sends it back to the security server. We assume that communications between the two participants, i.e., the time stamper and the security server is done over a secure channel.
- *PKI*
  The main goal of the public key infrastructure (*PKI*) is to publish the certificate revocation list(*CRL*). This is done either periodically or when the security server asks for a new *CRL*. This list will be used later by the security server for the verification of the incoming signatures. These two participants communicate over a secure channel.
- *Archiver*
  The role of this participant is to store records sent by the security server. We also assume that communications between these two participants is done over a secure channel. The archiver does not check a received message, except for testing that it has been not yet stored.

As stated above, there are two phases in the scenario of the DCS case study. The first phase is related to a client who first signs the contract. The second one considers a second signer who checks the first signature and then co-signs. For both phases, we consider the descriptions previously given in Deliverable D5.1 [4] and focus more particularly on the structure (shape) of messages exchanged between the participants involved. Throughout this section, we use K_S to denote a public key of the agent S.

First Phase of DCS

Steps of this phase follow the diagram sequence illustrated in Figure B.1. We assume that before this phase the two participants `Signer 1` and `Signer 2` have already agreed on the contract and that the `Business Portal` has already created this contract. This contract contains informations provided by the two signers before the first phase. Thus, the contract is only missing the signatures of the two parties.

As illustrated in Figure B.1, there are three steps in the first phase :

1. Creation of a proof record.
   This first step is triggered by the business portal (by sending the message *M1*) who asks the security server to create a digital case, called proof record and to deposit the contract into this newly created case. This proof record will later gather the contract and the signatures of the two signers. It is stored on the security server's file system. The business portal receives as a response the message *M2* saying that the proof record of the contract in question has been created and that the contract was added. The message *M2* contains the contract. The structure of messages *M1* and *M2* is given in the following table :

#	**Type**	**Structure**
M1	CreateRecordRequest	ID.Contract.S1.K_{S1}.S2.K_{S2}
M2	CreateRecordResponse	ID.Contract

2. Preparation of the signature.
   This step is reached when the first signer requests (via *M3*) the contract from the business

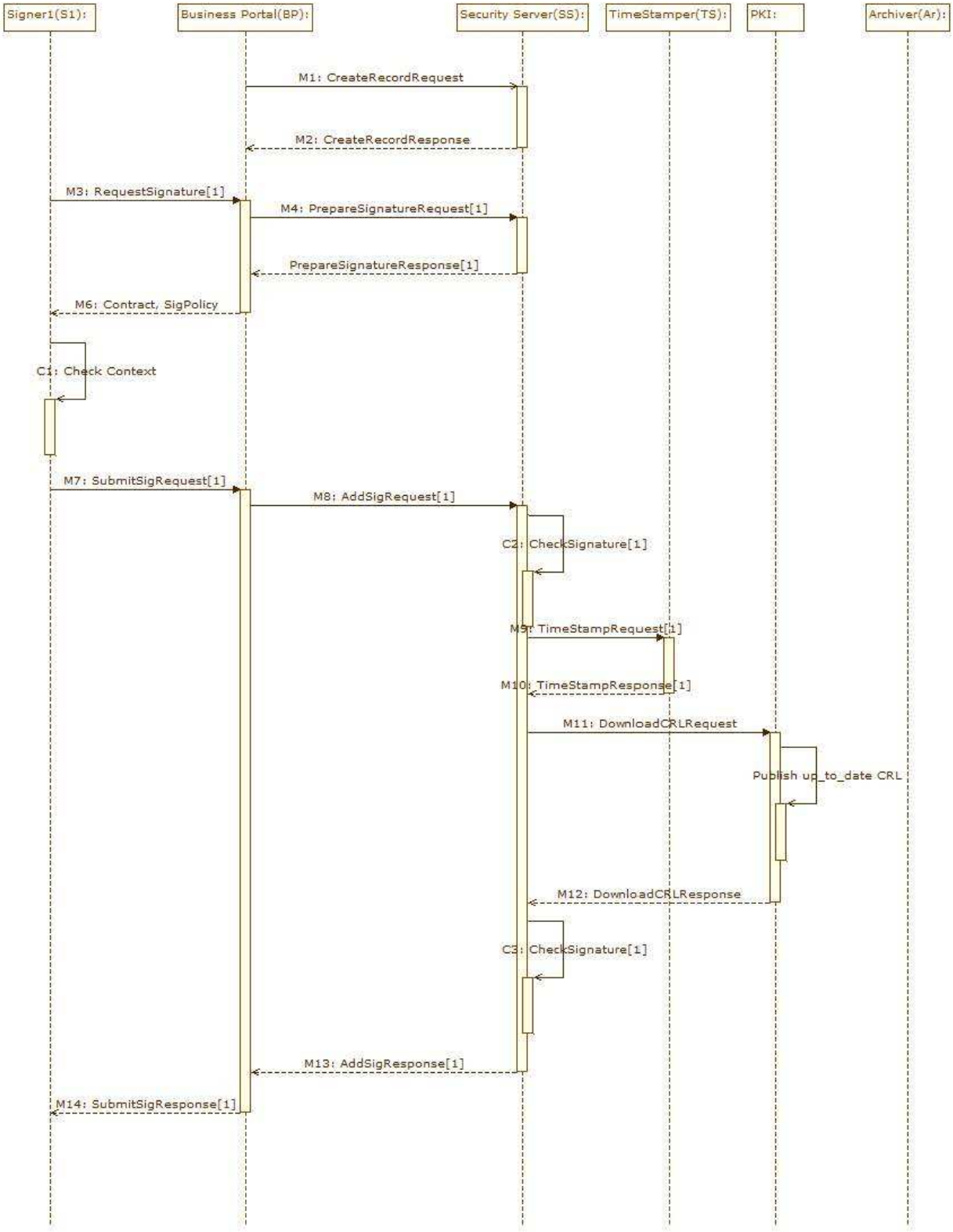


FIG. B.1 – First Phase of DCS

portal as well as all the rules and parameters applicable for generating the signature. These rules are defined in a signature policy managed by the security server. Upon receiving this request, the business portal requests (via *M4*) the contract signature preparation from the security server. This preparation consists in extracting a set of parameters from the signature policy. When receiving the signature policy (via *M5*), the business portal answers the first signer (via *M6*) by sending both the contract and the signature policy.

#	**Type**	**Structure**
M3	RequestSignature	ID.S1.K_{S1}
M4	PrepareSignatureRequest	ID.S1.K_{S1}
M5	PrepareSignatureResponse	ID.S1.K_{S1}.SignaturePolicy1
M6	SignaturePolicy	ID.S1.K_{S1}.Contract.SignaturePolicy1

3. Submission of the signature.
   The aim of this step is to submit the signature of the first signer in order to add it to the proof record. This step is divided into the following sub-steps :

   (a) The first signer views the contract and makes sure that it corresponds to the terms agreed upon (in *C1*). He then signs the contract using his private key and submits (via *M7*) this signature to the business portal.

   (b) The business portal forwards (via *M8*) the signature received to the security server in order to add it to the corresponding proof record.

   (c) The security server checks the signature (via *C2*) of the first signer.

   (d) The security server sends a timestamping request (via *M9*) to the time stamper to seal the signature's reception date. He gets in return (in *M10*) the timestamping response that contains the timestamped contract.

   (e) The security server sends a request (via *M11*) to the PKI to retrieve an up-to-date CRL ; The security server has an internal CRL cache which is refreshed periodically. Thus, when receiving (via *M12*) the new CRL from the PKI, the security server downloads the new CRL and loads it into its cache.

   (f) The security server checks (in *C3*) again the signature of the first signer. With the new CRL, the security server is able to confirm that the certificate of the first signer was not revoked when he created the first signature.

   (g) If all checks succeed, the security server sends a response (via *M13*) to the business portal saying that the signature is correct and containing the timestamped contract, signed by the security server.

   (h) The business portal forwards (via *M14*) the received message to the first signer.

   The structure of exchanged messages is given in the following table :

#	**Type**	**Structure**
M7	SubmitSignatureRequest	ID.signed(S1,Contract.S1.K_{S1}.SignaturePolicy1)
M8	AddSignatureRequest	ID.signed(S1,Contract.S1.K_{S1}.SignaturePolicy1)
M9	TimeStampRequest	ID.signed(S1,Contract.S1.K_{S1}.SignaturePolicy1)
M10	TimeStampResponse	ID.timestamped(TS,Time,signed(S1,Contract. S1.K_{S1}.SignaturePolicy1))
M11	DownloadCRLRequest	ID
M12	DownloadCRLResponse	ID.CRL
M13	AddSignatureResponse	ID.sigok(signed(SS,timestamped(TS,Time,signed(S1,Contract.S1.K_{S1}.SignaturePolicy1))),S1,K_{S1})
M14	SubmitSignatureResponse	ID.sigok(signed(SS,timestamped(TS,Time,signed(S1,Contract.S1.K_{S1}.SignaturePolicy1))),S1,K_{S1})

In this table, signed(A,M) denotes the signature of message M by agent A, timestamped(A,T,M) models the timestamping of message M by agent A using time T, sigok(M,A,K) denotes the fact that the signature of agent A having the public key K has successfully been checked and that M is the version of the signed contract which is timestamped and signed again by the security server.

Second Phase of DCS

The second phase of the scenario of the DCS case study corresponds to the signature of the contract by the second signer. As illustrated in Figure B.1, the second phase consists of three steps, where the two first steps work exactly in a similar way as for the first phase.

1. Preparation of the signature.
2. Submission of the signature.
3. Archiving the proof record.
   The proof record contains fully verified signatures : the contract and the two signatures. The security server sends it (via *M15*) to the archiver which guarantees that the signed contract can be restored in case it is removed from the Security Server's file system intentionally or accidentally. The archiver sends back (via *M16*) an acknowledgement saying that the proof record was successfully been archived. The structure of exchanged messages *M15* and *M16* is given in the following table :

#	**Type**	**Structure**
M15	ArchiveRecordRequest	ID.Msg_to_archive
M16	ArchiveRecordResponse	ID.arch_ok(Msg_to_archive,Ar)

In this table arch_ok(M,A) denotes the fact that the message M was already archived by agent A.

B.1.3 Composition scénario

The orchestration problem we posed is to generate a Mediator that emulates SS : satisfy BP's requests while relying on the community of available services (namely TS, PKI and ARC). Whereas the BP is the most natural candidate for a service to be orchestrated, we chose to generate SS in order to demonstrate the Orchestrator capabilities, since this entity possesses the most complex behavior of all in DCS.

The models for the Orchestrator was directly designed in ASLan independently from ones intended for the validation. The limitations assumed for these models are as follows : we assume

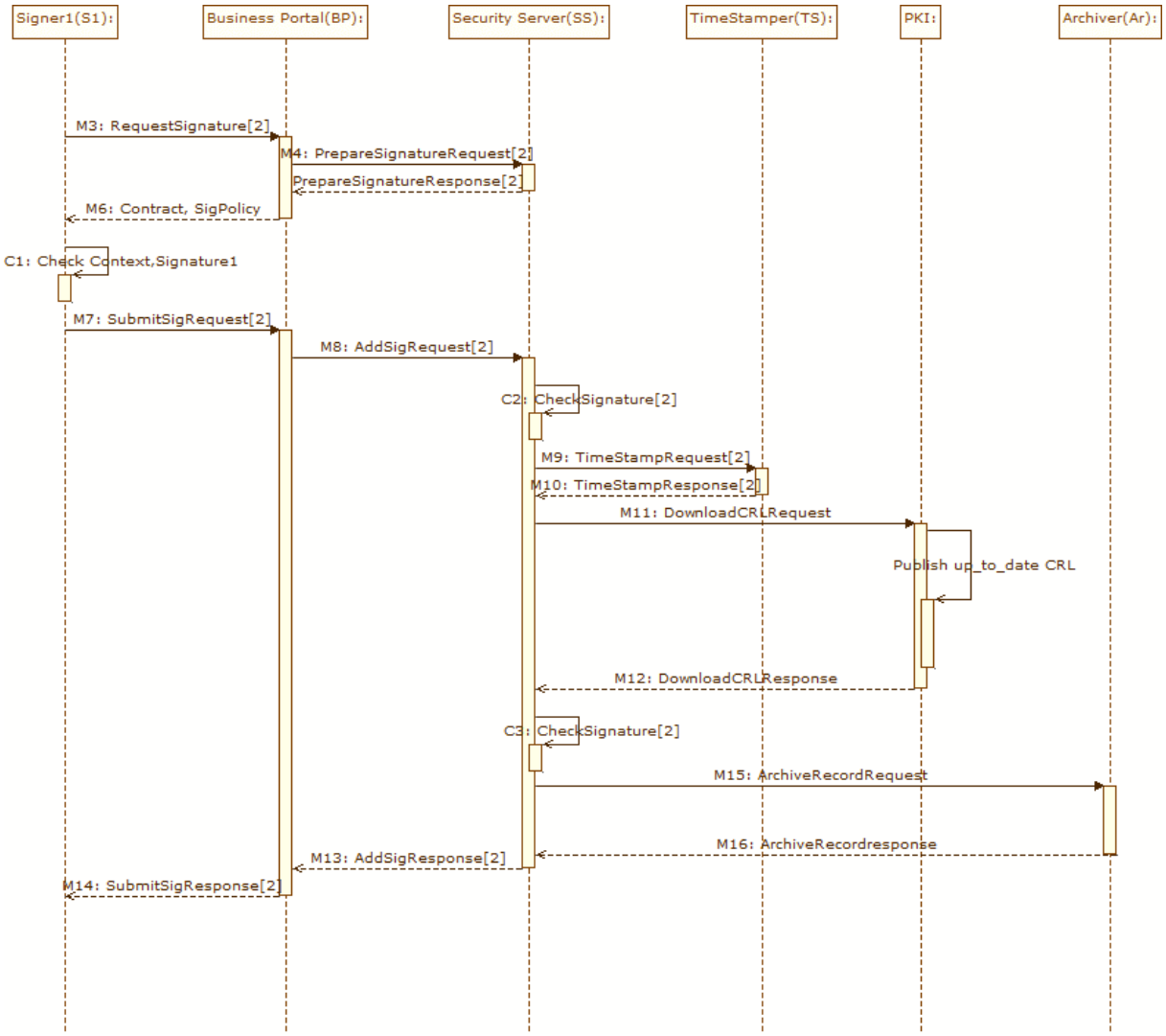


FIG. B.2 – Second Phase of DCS

the existence of a unique BP that represent signers and the number of represented signers is limited to two. Even in these settings the resulting model (that includes the generated Mediator) contains more than 40 transitions and the validation phase does not finish in a reasonable time, apparently because the specification satisfies the specified security goals.

We list the features used in DCS orchestration :

- *Input style*
  The orchestration problem was presented in two ways : *(i)* by giving BP as a client of SS, and *(ii)* by giving a partial specification of the SS which had be completed. In both cases the Orchestrator successfully found a possible orchestration in few seconds.
- *Security policies*
  The models take into account individual security policies of available services (e.g. every message received by ARC must be encrypted with his public key, etc) and thus the Mediator produces policy-compliant requests.
- *Rely-guarantee method*

The Mediator (SS) must produce some assertions (e.g. about the validity of a certificate) on which BP relies to continue his execution. Assertions are claims made by some issuer and stating some property for the parameters they transport. In the Web Service standards similar objects are represented by SAML [51] assertions, which we simply model here using first-order terms. The presence of an assertion in some received message by BP represents an additional constraint to the orchestration problem since SS will have to provide it. For example, to produce an assertion about the validity of a signer's certificate, SS has to contact an internal service : the Assertions Provider (AP) which permits to provide a good assertion only if a positive answer about the validity is given by a trusted third-party (here PKI). AP plays a role similar to the *trust engine* in the *rely-guarantee* method introduced in [39].

We emphasize here the expressiveness of assertions for the considered orchestration problem, since they can describe for example the need to use only certain schema for the timestamps, or only PKI's offering the *Online Certificate Status Protocol (OCSP)* versus those using the classical *Certificate Revocation List (CRL)*. This can be easily done by tuning the AP service behavior to match the expectations.

B.1.4 Security requirements and their classification.

. In this section, we recall the set of security requirements that a DCS scenario must fulfill. We consider properties defined in Deliverable [4] and specify them in ASLan++.

(S1) **Secrecy properties :** We have specified two variants of secrecy properties. The first one is related to the secrecy of archiving records. It states that each message sent to the archiver in order to store it should stay secret and must not be known by the intruder. Note that in the specification of the case study, all messages sent to the archiver are encrypted with a shared key between the latter and the security server.

We have modeled this property in ASLan++ by the following formula :

```
secrecy_archive :
  forall Message.
  [](secret_archive(Message)
  => (!iknows(Message))
  );
```

We have modified the security server entity in our case study specification by adding the fact secret_archive, stating that a message has to be checked for secrecy. In fact, when the security server sends a message Message to the archiver in order to store it, a new fact secret_archive(Message) is added to check that Message stay secret.

We have also specified a second secrecy property saying that the contract should stay secret. It should not be known by the intruder. This can be formalized as follows :

```
secrecy_contract :
  !iknows(contract_without_sigs);
```

In this formula, contract_without_sigs corresponds in the specification to the original contract passed as a parameter to the business portal.

(S2) **Authentication :** One of the most fundamental required properties in document exchange case studies and especially in the DCS case study, is authentication. To formalize the authentication property, we define two facts :

– witness_auth(agent,agent,message)

– request_auth(agent,agent,message)

The authentication property can then be formalized as follows :

```
authentication :
  forall A B M.
  [] (request_auth(A,B,M)
  => <->(witness_auth(A,B,M))
  );
```

This formula states that for each state where A requests B for M, sometime in the past B has sent the message M to A and thus the fact witness_auth(A,B,M) has occurred in the past.

For the DCS case study, we distinguish two kinds of authentication depending on the participants involved :

– authentication related to the end-users.
  Properties cited here correspond to the relation between the business portal and the end users, i.e. the two signers.
  – First, both signers must authenticate themselves when connecting to the business portal in order to get the contract. We have formalized this property by adding the following facts in our specification :

```
 1- witness_auth(Actor,BP,signature_req(Actor,
pk(Actor)))

 2- request_auth(S1,Actor,signature_req(S1,
pk(S1)))
```

    The first fact is added after having sent this request to the business portal. The second one is added at the reception of the corresponding message by the business portal. The same treatment holds for the authentication between the second signer and the business portal.
  – Second, signers must be able to authenticate the origin of the data they received. Indeed, both the first signer and the second one have to be sure that the contract that they will sign comes from the business portal. For this, we have added the following facts :

```
 1- witness_auth(Actor,S1,signature_rsp(S1,
pk(S1),Contract,SignaturePolicy1))

 2- request_auth(BP,Actor,signature_rsp(Actor,
pk(Actor),Contract,SignaturePolicy1))
```

    The first fact is added in the specification of the business portal, after sending the response of the latter. This response contains the contract. The second fact is added in the specification of the first signer at the reception of the corresponding response from the business portal. We treat authentication between the second signer and the business portal likewise.
– authentication related to the back-ends.
  We state here properties related to back end participants that are the security server, the

time stamper, the archiver and the PKI. These back-end services do not need and should not need to know the identity of the end-users. They provide services to the business portal who delegates his tasks to the security server.
Thus, we focus on the relation between the security server and the other back ends. We have specified three security requirements :

- the time stamper must be sure that the message that he will timestamp effectively comes from the security server. This is formalized by the following facts :

```
 1- witness_auth(Actor,TS,scrypt(K_SS_TS,
signed(S,Contract.S.K_S.SignaturePolicy)))

 2- request_auth(SS,Actor,scrypt(K_SS_TS,
Msg_to_timestamp))
```

  The first fact is added in the specification of the security server after sending the message to timestamp. The second fact is put in the specification of the timestamper when the latter receives the message to timestamp.
- the archiver authenticates the security server on the message that it will store. This is modeled as follows :

```
 1- witness_auth(Actor,Ar,scrypt(K_SS_Ar,
Proof_Record))

 2- request_auth(SS,Actor,scrypt(K_SS_Ar,
Msg_to_archive))
```

  The first fact is added in the specification of the security server after sending the message to archive. The second fact is put in the specification of the archiver when the latter receives the message to archive.
- the security server has to be sure that the certificate revocation list CRL he receives comes from the PKI. This is specified by adding the following two facts :

```
 1- witness_auth(Actor,SS,
scrypt(K_SS_PKI,CRL_Msg))

 2- request_auth(PKI,Actor,
scrypt(K_SS_PKI,CRL_Msg))
```

  The first fact is added in the specification of the PKI after sending the CRL. The second one is added in the specification of the security server when the latter receives the CRL.

(S3) **Integrity :** The two signers should not be able to modify the contract before signing. The security server then has to check that the uploaded signed contract is the same as the initial contract submitted to the first signer. The same requirement holds for the second signer.

```
integrity:
  forall C_orig C_sig1 C_sig2
 Signer1 Signer2 SPolicy1 SPolicy2.
  [] ((store_record (C_orig.C_sig1.C_sig2)
      & seal_record (C_orig.C_sig1.C_sig2))
  =>
```

```
((C_sig1 = signed (Signer1,C_orig.Signer1.
  pk(Signer1).SPolicy1))
& (C_sig2 = signed (Signer2,C_orig.Signer2.
  pk(Signer2).SPolicy2))
));
```

This specification formalizes the fact that the security server must ensure that the contract that has been uploaded by the two signers and then signed is the same as the one generated by the portal and submitted for signature. It states that if the security server seals the proof record and stores it (via the archiver), then this record contains the contract, the signed version of the contract by the first signer, and the signed version of the same contract by the second signer. This proves that no signer has modified the original contract.

Note that the specification of the security server entity is modified by adding seal_record(Proof_record) and store_record(Proof_record) as facts. The first fact is added when the security server receives the contract signed by the second signer. The second fact is added when the security server receives an acknowledgement from the archiver saying that the proof record was successfully stored.

(S4) **Non-repudiation :** This property aims at preventing a party from disclaiming its actions to invalidate the signing procedure. The digital contract signing(DCS) scenario has to satisfy non-repudiation for both origin and content. Indeed, signers must not be able to refute either the validity of their signatures or the content of the signed contract. Thus, the security server has to have enough proof elements to check a signature even after the signing procedure.

To formalize this property, we first introduce the following facts specifically related to the non-repudiation property.

- witness_non_repudiation(A,B,M).
  This fact states that A has created the message M for B.
- request_non_repudiation(A,B,M).
  This fact says that the agent A confirms having received the message M from B.

The non-repudiation property can be then formalized by means of the following formula :

```
non_repudiation:
  forall S BP SS Contract.
  [](request_non_repudiation(SS,S,signed(S,Contract))
  => <-> (
  (witness_non_repudiation(S,BP,signed(S,Contract)))
  & (witness_non_repudiation(BP,S,Contract))
  ));
```

This formula asserts that for every state in which the security server SS confirms having received from S the contract signed by the latter, sometime in the past the signer S has received the contract from the business portal (BP) and he has sent the signed version to BP.

In our case study, we have to verify the non-repudiation property for the two signers. For this, we have added the following facts :

```
 1- witness_non_repudiation(Actor,S1,Contract.S1.pk(S1).
SignaturePolicy1);
```

```
 2- witness_non_repudiation(Actor,S2,Contract.S2.pk(S2).
SignaturePolicy2);

 3- witness_non_repudiation(Actor,BP,
signed(Actor,Contract.Actor.pk(Actor).
SignaturePolicy2));

 4- witness_non_repudiation(Actor,BP,
signed(Actor,Contract.Actor.pk(Actor).
SignaturePolicy2));

 5- request_non_repudiation(Actor,S,
signed(S,Contract.S.K_S.SignaturePolicy))
```

Facts 1 and 2 are added in the specification of the business portal after sending the contract to the first signer and the second signer, respectively.

The fact 3 is placed in the specification of the first signer when he sends his signature to the business portal. Likewise, the fact 4 is placed in the specification of the second signer when he sends his signature to the business portal.

The fact 5 is added in the specification of the security server when he receives the signed contract.

(S5) **Proof of origin :** As stated in Deliverable [4] , this property states that the contract is submitted to the second signer for signature only when the security server has already fully validated the first signature.

To formalize this property, we have used the four following facts :

- ready_to_sign(agent,contract).
  This fact expresses that the agent agent can submit his signature of the contract contract.
- alreadySign(agent,contract).
  This fact states that the agent agent has already signed the contract contract. This supposes that the security server has validated the signature of the signer agent.
- isFirstSigner(agent).
  This fact says that agent is the first signer.
- isSecondSigner(agent).
  This fact states that agent is the second signer.

The property of proof of origin can then be defined as follows :

```
proof_of_origin:
  forall S1 S2 Contract.
  [] (ready_to_sign(S2,Contract) & isSecondSigner(S2)
  => <->(alreadySign(S1,Contract) & isFirstSigner(S1))
  );
```

This formula says that if the second signer has to submit his signature, then sometime in the past the first signer has already signed the contract.

The security concernes that we cover here are authorization (by means of 1), accountability (by 4), workflow security (by 5), application data protection (by 3), and communication security (by 2).

B.2 Public Bidding

B.2.1 Scenario Purpose

The Document Exchange Procedures cases aim at covering a large scale of dematerialized procedures. There is a growing demand for electronic services replacing paper-based business, either on the Internet or within enterprises. One concern of such service portals is to guarantee the legal value of the electronic documents produced. The security requirements imposed on such platforms are highly critical since the probative value of digitally signed documents relies on the conditions under which they have been produced and validated.

The Public Bidding process illustrates such a secure document exchange, and aims at providing a web application platform to manage an online call for tender, and also Bidders' proposal submissions.

B.2.2 High-level description

In this section, we describe the scenario of the Public Bidding case study, specified in AS-Lan++ in efappendix-pb. Based on the description given in Deliverable D5.1 [4] (Problem Cases, and their Trust and Security Requirements), we detail these steps while focusing on the structure of exchanged messages. As stated in the previous deliverables, the scenario is composed of four sub-processes : `Publication`, `Submission`, `Evaluation` and `Decision` phases. These phases involve the following participants :

- `The Bidding Manager` (`BM`).
  This participant is responsible for submitting the call for tender and making the final decision about the winner among the eligible `Bidders`.
- `The Technical Committee` (`TC`).
  The goal of this participant is to evaluate the technical proposals submitted by the `Bidders`.
- `Bidders`.
  The aim of a `Bidder` is to propose an offer during the submission phase and to check the result of the bidding during the decision phase.
- `The Bidding Portal` (`BP`).
  This participant has a central role in the process. He acts as an intermediate between the other roles, keeping unknown their identities. The management of digital proof elements (signing, checking signatures, timestamping, etc) is delegated to a dedicated security service provider, `The Security Server`.
- `The Security Server` (`SS`).
  The main role of this participant is to discharge `The Bidding Portal` from tasks related to the management of digital proof elements. Among its responsibilities are the signing of messages on behalf of `The Bidding Portal` and the checking of signatures given by the latter. He appeals `The Time Stamper` for timestamping messages and to `The Archiver` for the storage of messages.
- `The Time Stamper` (`TS`).
  This participant's goal is to timestamp messages sent by `The Security Server`. To do so, he adds the time to the received message and signs the resulting message.
- `The Archiver` (`Ar`).
  Its only role is to store messages sent by `The Security Server`.

The following paragraphs describe the tasks that compose the Public Bidding business process. After uploading the call for tender to the Bidding Portal, during the `The Publication phase`, the Bidders are able to send a tender during `The Submission phase`. Then, during `The Evaluation`

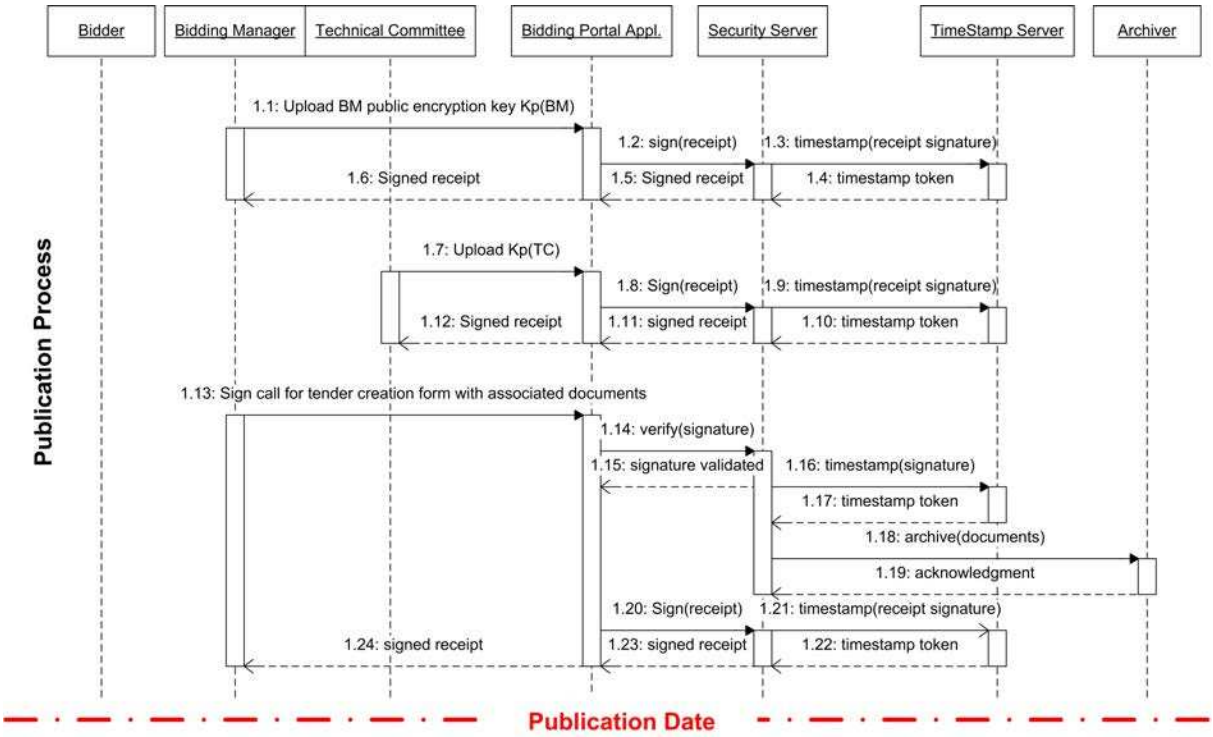


FIG. B.3 – Public Bidding - Publication sub-process.

and `Decision phases`, the documents produced by the Bidders are studied, in order to nominate the winner.

Publication sub-process. The Bidding process starts with the publication phase, which aims at publishing the call for tender, including the creation of the encryption keys. As illustrated in effig :PB-publication, the publication phase is composed of three major steps :

1. Upload of the Bidding Manager's key.
   The Bidding Manager uploads his public key to the Bidding Portal as the key to be used by potential Bidders for ciphering submitted documents. This step comprises these sub-steps :
   - Sending the public key of the Bidding Manager.
   - Generating a receipt.
     We detail this sub-step since a similar procedure is used in most other steps of the process.
     The timestamped receipt of a message (here the Bidding Manager's public key) is targeted to the original sender (in our case the Bidding Manager), as a proof of receipt of the original message. Here, the Bidding Portal generates a receipt of the message (the public key) he received from the Bidding Manager. We model the receipt of a message as a hash of this message : indeed, there is no need to have a proof of a receipt containing the original message, as the original owner (in our case the Bidding Manager) is able to hash the original message for verification.
     The Bidding Portal then sends the receipt to the Security Server in order to sign it on his behalf. The Security Server signs the received message and sends the result to the Time Stamper in order to timestamp it. The latter timestamps the message and signs it before sending back the resulting message to the Security Server.
     The Security Server sends the signed message back to the Bidding Portal. The structure

Message	Structure
M-1.2	scrypt(symmetricKey(SS,BP), timestamp_request.receipt(Puk_BM))
M-1.3	scrypt(symmetricKey(SS,TS),signed(SS,receipt(Puk_BM)))
M-1.4	scrypt(symmetricKey(SS,TS),timestamped(TS, ?Time, signed(SS,receipt(Puk_BM))))
M-1.5	scrypt(symmetricKey(SS,BP),timestamped(TS, ?Time, signed(SS,receipt(Puk_BM))))

TAB. B.1 – Messages for the generation of a receipt

of messages as terms is detailed in efgen-rec (the numbering of messages corresponds to effig :PB-publication). "scrypt" and "signed" model respectively encryption with a symmetric key and signing. "SS", "BP" and "TS" represent respectively the Security Server, the Bidding Portal and the Time Stamper.

In efgen-rec, `timestamp_request` is a flag denoting the kind of task the Bidding Portal asks for. Here, the Bidding Portal asks the Security Server to sign the message and to timestamp it. `Puk_BM` denotes the public key of the Bidding Manager.
We model timestamping via the function `timestamped(A,T,M)` that expresses the signature by the agent A of the message resulting on the concatenation of `hash(M)` with T. (Note that message `M` is not readable from `timestamped(A,T,M)`, only its hash).

- Receiving the signed receipt of the uploaded key.

2. Upload of the Technical Committee's key.
   The Technical Committee uploads his public key to the Bidding Portal as the key to be used for the encryption of technical proposals of the Bidders.
3. Upload of the Bidding Manager's tender.
   This step comprises these sub-steps :
   - The Bidding Manager creates a new call for tender and sends it (signed) to the Bidding Portal. Thus, the message 1.13 has the form of `signed(BM,upload_tender.Tender)` where `upload_ tender` is a flag that shows the kind of the message. `Tender` is the call for tender of the Bidding Manager and `signed(A,M)` denotes the signed message `M` (with agent `A`'s signature).
   - Storing the call for tender. The Security Server checks the signature. If the check succeeds, the Bidding Portal receives a message indicating that the signature is valid and that the process can go on. In parallel, the Security Server delegates the task of timestamping to the Time Stamper. He then delegates the task of storage of the timestamped message to the Archiver.
   - Generating a receipt.

Submission sub-process. During this phase, the Bidders can download the call for tender, and submit their own offer.

As illustrated in effig :PB-submission, this phase is reduced to one step, which can be divided into the following sub-steps. The Submission sub-process is bounded to a timer, as the Bidders cannot give their tender after a deadline.

1. Ask for call for tender.
   First, a Bidder B sends a `submission_request(B)` to the Bidding Portal. After receiving it,

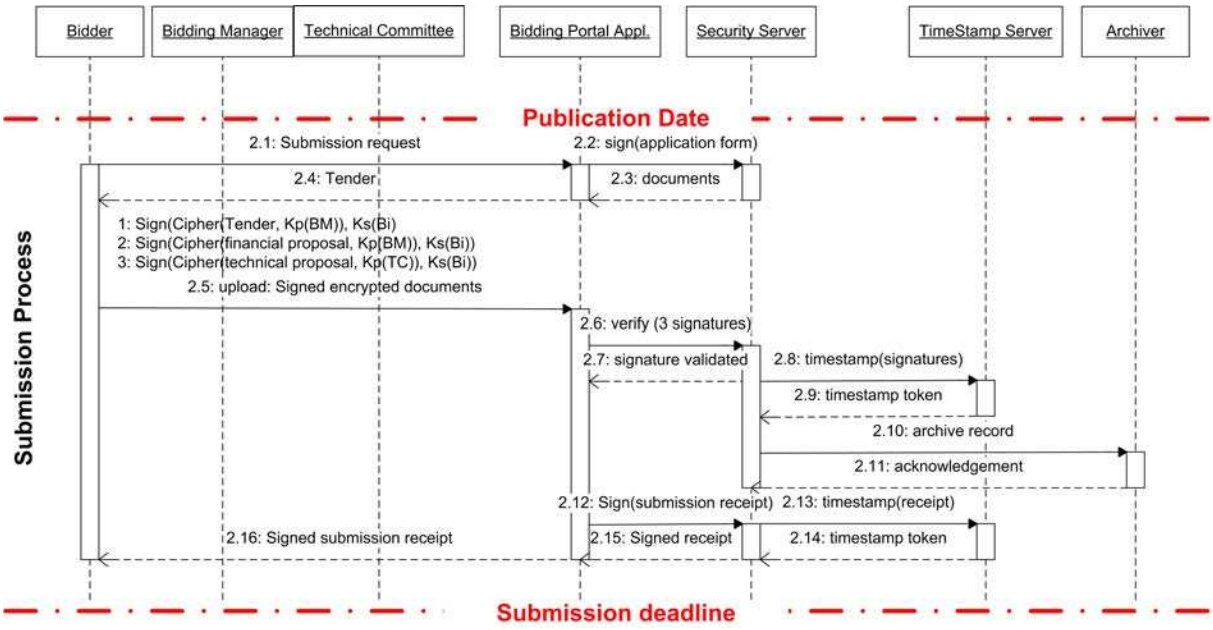


FIG. B.4 – Public Bidding - Submission sub-process

the portal sends the call for tender, with the public keys of the Bidding Manager and of the Technical Committee to the Security Server. This is message 2.2, $\texttt{signature\_request.Tender.publicKey(BM)}$ encrypted using k_{SS-BP}, the symmetric key shared between the Security Server and the Bidding Portal. The header lets the Security Server know that he has to sign it. The Security Server sends back to the Bidding Portal a resulting message of the form : $\texttt{signed(SS, Tender.publicKey(BM).publicKey(TC))}$ encrypted using k_{SS-BP}.
Then, the Bidding Portal sends the message back to the Bidder. This message contains the public keys of the Bidding Manager and of the Technical Committee and is signed by the Security Server.

2. Upload of signed documents.
   This sub-step is quite similar to the upload of the tender, except for the signature verification, where, here, the Security Server needs to validate the signatures for three documents :
   - the tender, encrypted with the Bidding Manager's key $\texttt{signed(Bidder, crypt(pk(BM), Tender))}$.
   - the financial proposal, encrypted with the Bidding Manager's key $\texttt{signed(Bidder, crypt(pk(BM), FP))}$.
   - the technical proposal, encrypted with the Technical Committee's key $\texttt{signed(Bidder, crypt(pk(TC), TP))}$.

   We also have an archiving step and then a receipt generation for each tender submitted by Bidders.

Evaluation sub-process. In this phase, the Bidding Manager receives the list of all Bidders, and sends back to the Bidding Portal the list of eligible Bidders. Then, the Technical Committee creates a report from the technical proposals of the eligible Bidders.

As shown in effig :PB-evaluation, this sub-process is divided into two steps :

1. The creation of eligible Bidders list.
   First, the Bidding Manager retrieves the server-signed list of Bidders, with their encrypted

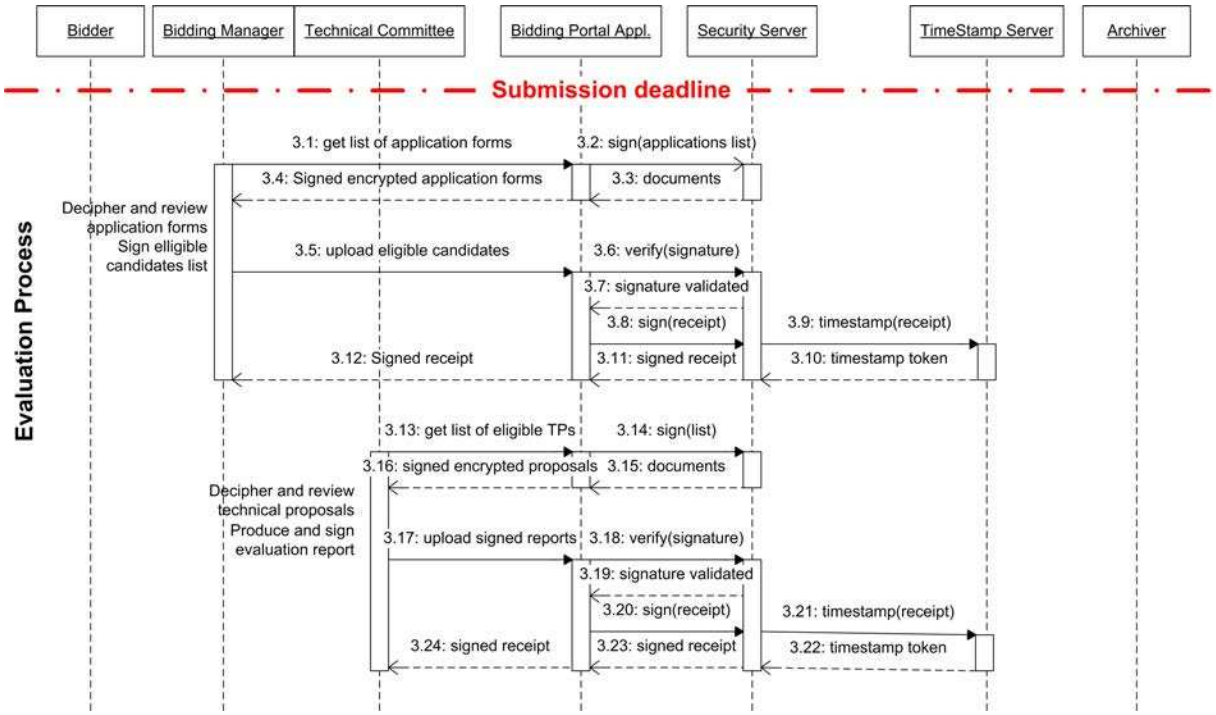


FIG. B.5 – Public Bidding - Evaluation sub-process.

tenders, as they were uploaded during the call for tender.
Then, he sends to the Bidding Portal the list of eligible Bidders. The portal sends this list to the Security Server, encrypted with the server-portal symmetric key, so that the server validates the signature. This is done by adding the flag `signature_validation_request` to the head of the message. The Security Server then sends to the Bidding Manager the message `valid_signature.Eligible_list`.
Then, a receipt (the hash of the `Eligible_list`) is generated and timestamped.

2. The creation of technical reports.
   Here, the Technical Committee creates a report for each technical report received from eligible Bidders, with the same sub-steps as the creation of eligible Bidders list.

Decision sub-process. In this last phase, the Bidding Manager retrieves the technical reports and the financial proposals from the Bidding Portal, and determines the winner. After that, each Bidder is able to see, from the Bidding Portal, who the winner is.

As we can see in effig :PB-decision, here there are two majors steps :

1. Choosing the best Bidder.
   The Decision sub-step has exactly the same pattern as the Submission phase : that is why we will not give a detailed description of it.
   - The Bidding Manager downloads the financial proposals and the technical reports from the Bidding Portal. The latter sends them to the Security Server, with a `signature_request` flag. After signing is done, the documents are sent back to the Bidding Portal, then to the Bidding Manager.
   - The Bidding Manager uploads the result, which will be archived following the archiving process described before.
   - A receipt is generated.

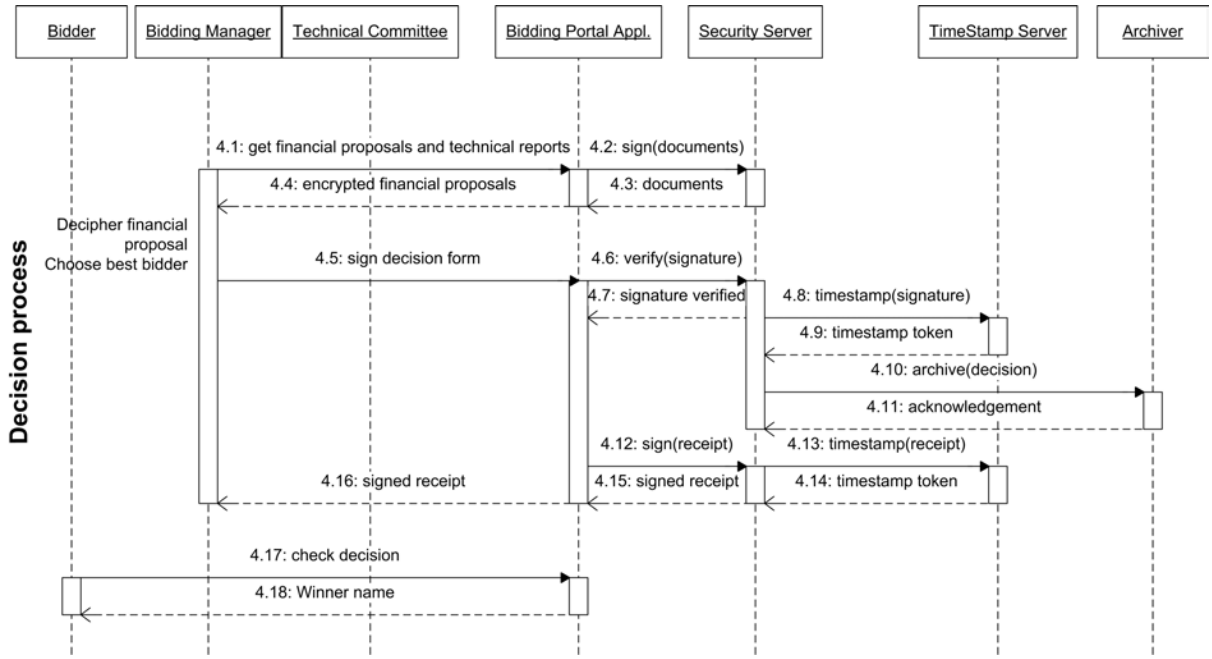


FIG. B.6 – Public Bidding - Decision sub-process

2. Checking the decision.
   Finally, each Bidder, who wants to know the result, sends a `winner_ request` to the Bidding Portal. The latter sends back the name of the winner.

B.2.3 Composition scénario

The Orchestrator was evaluated on the Public Bidding example by stating as goal the outcome of a successful bid. The starting point was scene 4, which is a variation of scene 2 with some manual loop optimizations, as described further below. The Bidder process was removed from the model and the successful orchestration showed that the Orchestrator can generate a Bidder process that uses the other services provided in the model (notably the Bid Portal) to achieve the desired outcome. The Orchestrator thus effectively manages to synthesize the Bidder process.

For the resulting orchestration, the properties specified for the original model were verified : authenticity, integrity, non-repudiation, secrecy, and executability, thus checking the correctness of the generated orchestration.

In order to achieve both orchestration and validation, several targeted optimizations were performed on the ASLan output of the ASLan++ translator. In particular, the model was rewritten such that all paths through the main message reception loops would be distinct. This makes all transitions reachable from the initial states on paths which need not repeat any transition, which significantly lowers the state space exploration complexity for the backends. In particular, for CL-AtSe, the default option *--nb 1* can be employed instead of *--nb 2*, and time decreases 100 times, from more than half an hour to less than 20 seconds.

Overall, this case study showcased a successful combination of orchestration and validation, albeit on a simplified model, and pointed out optimizations that can help in handling more complex models. Some of these optimizations could be performed by enhancing the step lumping option of the translator, while others require rewriting the model with validation performance in mind.

B.2.4 Security requirements and their classification

The Public Bidding scenario must fulfill a number of security requirements, expressed in Deliverable D5.1 [4], and also described in Deliverable 2.1 [3]. Here we give the list of these requirements :

(S1) **Separation of roles** : Within the process, a Bidder needs to create and send a `technical proposal` and a `financial proposal`, which will be evaluated respectively by the Technical Committee and the Bid Manager. These two entities should note the Bidders' documents according to their contents. Actually, it is impossible to make sure that the bid will be fully fair. For example, a Bidder can use out-of-band mechanisms to give information about their identity, or money, to the decision makers. Such situations are not in the scope of the AVANTSSAR project ; we will come on fairness issues later. Here, we aim at preventing a peculiar situation which could lead to a unfair evaluation : if the Bid Manager or someone from the Technical Committee submits its own offer, as a Bidder, he then would be able to overvalue its offer. Thus, we should prevent the Technical Committee and the Bid Manager from submitting Bidders' documents to the Bidding Portal. In other words, the Technical Committee and the Bid Manager must not have the authorization to send an `application form`, containing the `technical proposal` and the `financial proposal`.

(S2) **Secrecy on Bidders** : In a Public Bidding scenario, a Bidder is not allowed to know any information about his opponents, neither their name nor their submitted record. Thus, we have to verify that all information concerning the Bidders is kept secret and is only known by the targeted participants.

(S3) **Non − repudiation** : A common requirement in document exchange procedures in general is non-repudiation, aiming at preventing a party from disclaiming its actions to invalidate the process. That is the purpose of the `Proof Record` created by the Archiver : recording enough documents to avoid such repudiation. This security requirement aims at verifying that for each document an actor `B` "owns", some time in the past an actor `A` created that message for `B`.

(S4) **Data integrity** : A common requirement in such a process is to ensure the integrity of documents. In the scenario, the Bidders create and send several documents to the Bidding Portal : the `tender`, the `financial proposal` and the `technical proposal`. Thus we must ensure that no one can remove, add or change one or several of these documents.

(S5) **Authentication** : Although the authentication process is not a part of the Public Bidding scenario described in Deliverable D5.1 [4], we can assume the certificate distribution is done before the bidding process starts. Thus, we have to verify that each end-user must be authenticated to the Bidding Portal (when the Bidding Portal receives a message from an end-user), and vice versa.

(S6) **Fairness** : This scenario comes with fairness issues, expressing the need that no Bidder could be removed on purpose or inadvertently. Thus, when the Bidding Manager receives the list of the participating Bidders, he must not omit from the list sent to the Bidding Portal a Bidder who is eligible. Also, the Technical Committee cannot drop a `technical proposal` when evaluating it. For example, this second case can be expressed in the following way : for each eligible Bidder, a report on his `technical proposal` is eventually created. In the same fashion more fairness requirements can be specified, e.g. "All bidders must have a proof of receipt of their `tender`".

Thus the covered security concerns are : authorization policies, accountability, application data protection and communication security.

B.3 Car Registration Process

B.3.1 Scenario purpose

The purpose of the Car Registration Protocol is to model a typical e-government scenario, specifically within the class of government-to-citizen (G2C) services. This corresponds to the initiative of several EU countries to create and maintain certified Citizen Portals that will support a secure communication interface within the Internet. Every citizen should have a secure, personalized Internet access point, enabling communication with government offices and service providers in an easily usable and secure way. From this portal, citizens may access a great variety of services with different authentication, authorization, and protection requirements. The portal is responsible for providing the corresponding security mechanisms and protecting the privacy of the citizen.

The main focus of the CRP case study is the interplay between workflow and access control policies. The goals of this case study are the orchestration of the workflow while abstracting away the policies, and the validation of the resulting workflow taking the access control policies into account.

B.3.2 High-level description

Excerpts of the CRP have been modeled in ASLan in Deliverable 2.1 [3] and a preliminary model in ASLan++ was included in Deliverable 2.2 [7], to illustrate the design and use of some language features. Subsequently, an ASLan model, derived by translation and manual adaptation from an ASLan++ model was used both for orchestration (showing that a valid usage scenario by a client exists), and for validation of several security properties. In the current deliverable we provide a complete and refined ASLan++ model with additional features which can be validated after a completely automated translation to ASLan.

The case study involves several agents, one of which is the citizen who submits a car registration request. The citizen is viewed as the orchestration client and the goal of the orchestration phase is to satisfy his registration request. The formalization process was done is two steps :

1. First we have created a complete model representing the scenario we would like to obtain as the result of orchestration. This model includes the citizen agent. The purpose of this step is to test the modeling capabilities of the ASLan++ language and the behavior of the back-end tools on the resulting ASLan translation.
2. Second, we have transformed the citizen into an orchestration client (as specified in the section about the orchestrator from Deliverable 4.1 [8]) and considered the resulting orchestration problem. The purpose of this step was to test the orchestrator, which should be able to generate an orchestration goal so that a successful completion of a car registration request is performed.

We present the complete model used for validation, since the model for orchestration is obtained by changing parts of the former. When presenting the orchestration, we will highlight only the adaptations that were performed on the initial model in order to be accepted by the orchestrator.

The complete model for this case study can be analyzed at two distinct levels :
- the functional level of the workflow, where the participating agents exchange messages in order to achieve the common goal of processing car registration requests ;
- the meta level of access control policies, where the agents exchange information about who has the right to perform various operations.

The distinction is possible because we model the access control policies through Horn clauses. This choice relies on the fact that deductions through Horn clauses are performed instantaneously, without the need to exchange any messages, and thus represent a natural way to express policies. The link between the two levels is made by a *Certification Authority*agent, who emits signed certificates for all agents that are allowed to perform certain operations. These certificates are sent by the *Certification Authority*as messages at the functional level. Once an agent receives a certificate, he makes the knowledge available at the meta level and from there on the reasoning about access control policies is done at the meta level through Horn clauses.

First we describe the functional level : the behavior of each agent involved and the message exchange protocol between the agents. Then we describe the meta level : the Horn clauses that make possible for agents to know who has the right to perform operations and who does not have it.

The Functional Level There are six roles that can be played by agents : *Certification Authority*, *Citizen*, *Office Head*, *Employee*, *Registration Office*and *Central Repository*. For a general description of each role please refer to Deliverable 5.1 [4].

B.7 shows an abstract representation of the messages that are exchanged by the various roles.

The *Certification Authority*interacts with the employees of a *Registration Office*. The *Certification Authority*sends certificates according to the following rules :

- Every *Office Head*receives a certificate that states its position in the office (message 1) ;
- Every *Employee*receives a certificate that states its position in the office (message 2) ;

The *Citizen*interacts with the *Central Repository*and the *Registration Office*, as follows :

1. The *Citizen*requests an empty form from the *Central Repository*(message 4)
2. The *Central Repository*sends an empty form to the *Citizen*(message 5)
3. The *Citizen*fills and signs the form (step 6)
4. The *Citizen*submits the filled and signed form to the *Registration Office*(message 7)
5. The *Registration Office*sends in return a registration number that can be used to track the submitted form (message 8)
6. After its request is processed, the *Citizen*receives from the *Registration Office*a decision (message 19), denoting whether its request was *accepted* or *refused*.

The *Office Head*has predominantly a static role in the model. Besides receiving his certificate from the *Certification Authority*(message 1) and sharing it with trusted *Employee*s (message 3), he does not explicitly interact with anyone at the functional level. Rather he emits knowledge at the meta level about which *Employee*s are allowed to store documents in the *Central Repository*. The *Employee*s are modeled as belonging to one of two categories : trusted and untrusted. Only the trusted *Employee*s are granted rights to store documents in the *Central Repository*. We will describe how this is done when we present the meta level of the model.

The *Employee*does the actual job of processing car registration requests and deciding if they should be accepted or refused. The *Employee*interacts with the *Certification Authority*, the *Registration Office*and the *Central Repository*. Note that it does not directly interact with the *Citizen*. At any time the *Employee*can do one of the following :

- receive a certificate from the *Certification Authority*(message 2), in which case the *Employee*will make the received information available at the meta level ;
- interact with the *Registration Office*with the purpose of processing car registration requests, by following these steps :

1. the *Employee* asks the *Registration Office* for a car registration request that needs to be processed (message 10) ;
2. take an action, depending on the answer received from the *Registration Office*
   - if the *Registration Office* does not authorize the access (this can happen if the *Employee* is not an employee of that specific *Registration Office*) then the *Employee* does nothing ;
   - if the *Registration Office* says that there are no pending car registration requests, then again the *Employee* does nothing ;
   - if the *Registration Office* sends back a car registration request for processing (message 12), then :
     - if the request is valid then it is sent to the *Central Repository* to be stored (message 16), and :
       - if the *Central Repository* authorizes the storage (message 17), then the *Employee* informs the *Registration Office* that the car registration request was accepted (message 18) ;
       - if the *Central Repository* does not authorize the storage, then the *Employee* sends back the car registration request to the *Registration Office* (message 14) so that it will be later processed by another *Employee* who can store documents in the *Central Repository* ;
     - if the request is invalid, then the *Employee* marks it as refused and notifies the *Registration Office* (message 18).

The *Registration Office* interacts with the *Citizen* and the *Employee*. At each moment the *Registration Office* can do one of the following :

- receive from a *Citizen* a car registration request (message 7), in which case :
  1. assign a unique registration number to the request
  2. send the registration number back to the *Citizen* (message 8)
  3. store the request in a local database for later processing (step 9)
- receive from an *Employee* a request for work (message 10), in which case :
  - if the *Employee* is authorized to request work (he is an employee of this specific *Registration Office*) then
    - if there is no pending car registration request to be processed, inform the *Employee* about this ;
    - if there are pending car registration requests to be processed, pick one randomly (step 11) and send it back to the *Employee* (message 12) ;
  - if the *Employee* is not authorized to request work, send an appropriate denial message back to him.
- receive from an *Employee* a request to store back a car registration request which could not be fully processed (message 14), in which case :
  - if the *Employee* is authorized for the operation (he is an employee of this specific *Registration Office*) then store the car registration request in the local database (step 15), so that it can later be processed by another *Employee* ;
  - if the *Employee* is not authorized for the operation, send an appropriate denial message back to him.
- receive from an *Employee* a final decision about a processed car registration request (message 18), in which case :
  - if the *Employee* is authorized for the operation (he is an employee of that specific *Registration Office*) then notify the *Citizen* who sent the car registration request about the

final decision (message 19) ;
- if the *Employee* is not authorized for the operation, send an appropriate denial message back to him.

The *Central Repository* interacts with the *Citizen* and the *Employee*. At any time the *Central Repository* can do one of the following :

- receive from the *Citizen* a request for an empty form (message 4), and respond by sending back an empty form (message 5) ;
- receive from an *Employee* a request to store a document (message 16), in which case :
  - if the *Employee* is authorized to store documents, then store the document and inform the *Employee* about the success of the operation (message 17) ;
  - if the *Employee* is not authorized to store documents, then send him back a denial message and don't store the document.

The Meta Level The purpose of the meta level is to make available to the *Central Repository* the knowledge about who has the right to store documents there and who has not.

Writing documents to the *Central Repository* requires a special permission, which can be given by the *Office Head*. The *Certification Authority* decides who has the role of *Employee* or *Office Head*. This is expressed by signed certificates which are sent by the *Certification Authority* to everyone who has one of the two roles. In addition, every agent who has the role of *Employee* will also receive a certificate about the *Office Head* of his *Registration Office*.

B.8 shows schematically the flow of information about access control policies. It can be seen how from the two sources (*Certification Authority* and *Office Head*) the information flows to the meta level where (through Horn clauses) all necessary deductions are performed. From the meta level the information is made available to the *Central Repository*, but in a distilled form, i.e., only the knowledge about who has the right to store documents into the *Central Repository*. Note that although B.8 is a sequence diagram, only the dotted arrows represent actual messages exchanged between agents. The solid arrows represent information flow at the meta level and deductions made through Horn clauses.

B.3.3 Composition scenario

In order to perform orchestration on the Car Registration specification, the *Citizen* entity is transformed into an *Orchestration Client* so that the orchestrator synthesizes an *Orchestration Goal* that coordinates the rest of the entities towards a successful processing of a car registration request. The transformation is performed automatically by the ASLan++ connector, starting from the same ASLan++ model that is used for verification.

The orchestration is performed successfully and the result can be sent to the validation platform for verifying the security properties. CL-AtSe is able to verify the orchestrated model that uses static access policies, while on the one that uses dynamic access control policies it does not terminate in the one hour limit imposed during the assessment tests. OFMC and SATMC do not terminate in the one hour limit on both versions of the model, with static and with dynamic access control policies. The reason for the too long running time of the backends is that the orchestrated model is much more complex than the original model, with many extra transitions added for the *Orchestration Goal*.

B.3.4 Security requirements and their classification

We started from the security requirements defined in the CRP specification in Deliverable 5.1 [4], and then we extended the set of security requirements as the model took shape, obtaining the following :

(S1) **Data consistency :** Documents stored in the repository are consistent, i.e., their signatures are correct.

(S2) **Proper logging of data** A *Citizen*never receives an affirmative answer for its request without its request being stored in the *Central Repository*.

(S3) **Secure workflow binding :** A *Citizen*never receives an answer intended for the request of another *Citizen*(or for another request of the same *Citizen*).

(S4) **Correct workflow sequence :** The workflow steps are completed in the appropriate order, i.e., a *Citizen*receives a registration number only after submitting a document, and then receives an answer only after receiving a registration number, etc.

(S5) **Non-repudiation :** A *Registration Office*is unable to repudiate that it issued decisions for car registration requests.

(S6) **Privacy of data :** The information submitted by a *Citizen*is known only the by authorized employees of the *Registration Office*. It can never be accessed by unauthorized employees and especially by persons who are not employed in the *Registration Office*.

(S7) **Proper authorization for changing data :** Only persons who are authorized to store documents into the *Central Repository*should be allowed to do that.

(S8) **Proper authorization for task distribution :** Only employees of a *Registration Office*should be allowed to request work from that *Registration Office*.

(S9) **Liveness :** A car registration request submitted by a *Citizen*should eventually get processed (no registration request is infinitely postponed).

(S10) **Consistent trust relations :** Based on the trust relations, the *Central Repository*always knows the correct access rights of employees.

Thus the covered security concerns are : authorization policies, accountability, trust management, workflow security, privacy, application data protection, and communication security.

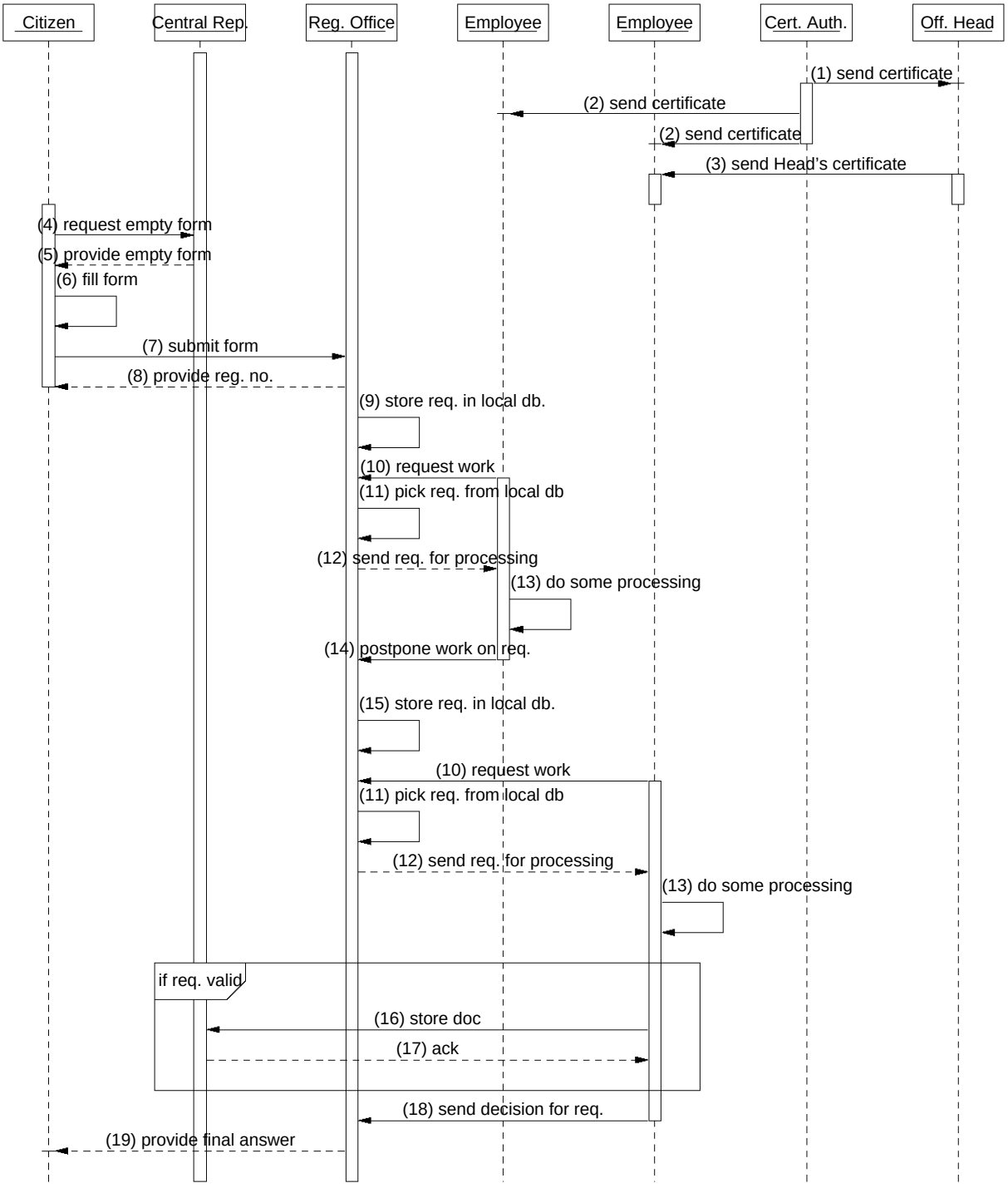


FIG. B.7 – Sequence diagram for the Car Registration Protocol case study.

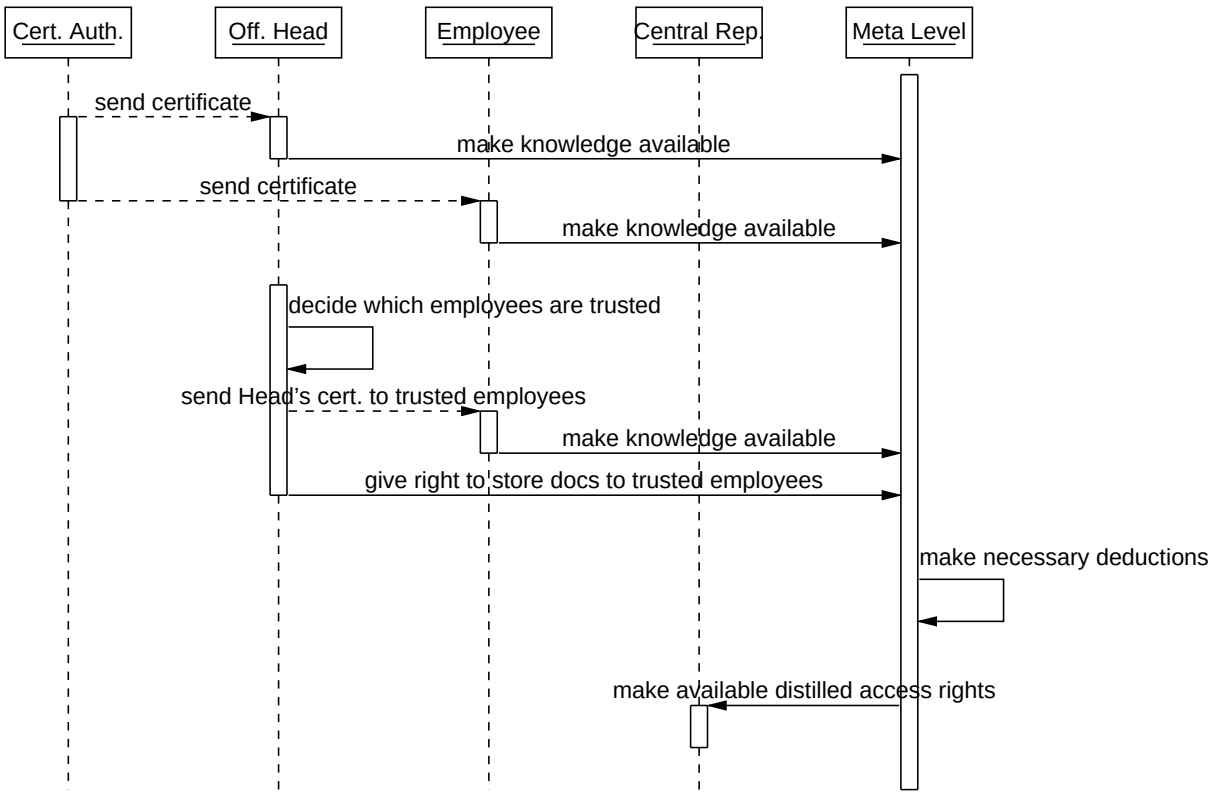


FIG. B.8 – Flow of information regarding access control in the Car Registration Protocol.

Bibliographie

[1] Automated Validation of Trust and Security of Service-Oriented Architectures, AVANTSSAR project. `http://www.avantssar.eu`, 2008–2010.

[2] Gavin Lowe August. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56 :131–133, 1995.

[3] AVANTSSAR. Deliverable 2.1 : Requirements for modelling and ASLan v.1. `www.avantssar.eu`, 2008.

[4] AVANTSSAR. Deliverable 5.1 : Problem cases and their trust and security requirements. `www.avantssar.eu`, 2008.

[5] AVANTSSAR. Deliverable 6.2.1 : State-of-the-art on specification languages for service-oriented architectures. Available at `http://www.avantssar.eu`, 2008.

[6] AVANTSSAR. Deliverable 6.2.2 : Industrial language requirements. Available at `http://www.avantssar.eu`, 2008.

[7] AVANTSSAR. Deliverable 2.2 : ASLan v.2 with static service and policy composition. Available at `http://www.avantssar.eu`, 2009.

[8] AVANTSSAR. Deliverable 4.1 : AVANTSSAR Validation Platform v.1. Available at `http://www.avantssar.eu`, 2009.

[9] AVANTSSAR. Deliverable 2.3 : ASLan final version with dynamic service and policy composition. `http://www.avantssar.eu/pdf/deliverables/avantssar-d2-3.pdf`, 2010.

[10] AVANTSSAR. Deliverable 5.4 : Assessment of the AVANTSSAR Validation Platform. Available at `http://www.avantssar.eu`, 2010.

[11] AVANTSSAR. The AVANTSSAR Validation Platform. Available at `http://www.avantssar.eu`, 2010.

[12] AVANTSSAR. Deliverable 2.3 (update) : ASLan++ specification and tutorial. `www.avantssar.eu`, 2011.

[13] AVISPA. Deliverable 2.3 : The Intermediate Format. `www.avispa-project.org`, 2003.

[14] Philippe Balbiani, Fahima Cheikh, and Guillaume Feuillade. Composition of interactive web services based on controller synthesis. *2nd International Workshop on Web Service Composition and Adaptation (WSCA 08)*, 0(0) :521–528, 2008.

[15] M. Baudet, V. Cortier, and S. Delaune. Yapa : A generic tool for computing intruder knowledge. In *RTA'09, 20th International Conference on Rewriting Techniques and Applications*, 2009.

[16] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based semantic Web Services with Messaging. In *Proc. 31st Int. Conf. Very Large Data Bases, VLDB 2005*, pages 613–624, 2005.

[17] D. Beringer, H. Kuno, and M. Lemon. Using WSCL in a UDDI Registry 1.02. UDDI Working Draft Technical Note Document, May 5, 2001. 2001.

[18] Antonia Bertolino, Paola Inverardi, Patrizio Pelliccione, and Massimo Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *ESEC/FSE '09 : Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 141–150, New York, NY, USA, 2009. ACM.

[19] Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, pages 124–140, Washington, DC, USA, 2009. IEEE Computer Society.

[20] Pierre Bieber. A logic of communication in hostile environment. In *Computer Security Foundations Workshop (III)*, pages 14–22, 1990.

[21] Antonio Bucchiarone and Stefania Gnesi. A survey on services composition languages and models. In *Proceedings of International Workshop on Web Services Modeling and Testing*, WS-MaTe2006, pages 51–63, 2006.

[22] Antonio Bucchiarone and Stefania Gnesi. Web service composition approaches : From industrial standards to formal methods. In *Internet and Web Applications and Services*, ICIW'07, page 15, 2007.

[23] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification : a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.

[24] Carlos Caleiro, Luca Viganò, and David Basin. On the semantics of alice&bob specifications of security protocols. *Theoretical Computer Science*, 367(1-2) :88–122, 2006.

[25] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *Programming Languages and Systems*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-71316-6_2.

[26] U. Carlsen. Generating formal cryptographic protocol specifications. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 137 –146, May 1994.

[27] Y. Chevalier, M.A. Mekki, and M. Rusinowitch. Orchestration under security constraints. In *Sixth International Workshop on Formal Aspects in Security and Trust (FAST2009) Eindhoven, the Netherlands, November 5-6, 2009*, 2009.

[28] Yannick Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques.* PhD thesis, Université Henri Poincaré, Nancy, Décembre 2003.

[29] Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Jacopo Mantovani, Sebastian Mödersheim, and Laurent Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Automated Software Engineering. Proceedings of the Workshop on Specification and Automated Processing of Security Requirements, SAPS'04*, pages 193–205. Austrian Computer Society, Austria, September 2004.

[30] Yannick Chevalier, Mohammed Anis Mekki, and Michaël Rusinowitch. Automatic composition of services with security policies. In *Proceedings of the 2008 IEEE Congress on Services - Part I*, SERVICES '08, pages 529–537, Washington, DC, USA, 2008. IEEE Computer Society.

[31] Yannick Chevalier and Michaël Rusinowitch. Compiling and securing cryptographic protocols. *Inf. Process. Lett.*, 110(3) :116–122, 2010.

[32] OMG Consortium. Business Process Modeling Notation (BPMN). `http://www.omg.org/spec/BPMN/1.1/PDF`, 2006.

[33] The World Wide Web Consortium. Simple Object Access Protocol 1.2. `http://www.w3.org/TR/soap12-part1`, Apr 2007.

[34] F McCabe E. Newcomer M. Champion C. Ferris D. Booth, H. Haas and D. Orchard. Web Services Architecture. 2007. `http://www.w3.org/TR/ws-arch/`.

[35] Grit Denker, Jonathan Millen, and Harald Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000. `http://www.csl.sri.com/~millen/capsl/`.

[36] Maurizio Lenzerini Massimo Mecella Diego Calvanese, Giuseppe De Giacomo and Fabio Patrizi. Automatic service composition and synthesis : the roman model. In *IEEE Data Eng. Bull.*, volume 3, pages 18–22, 2008.

[37] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[38] F.J.T. Fabrega, J.C. Herzog, and J.D. Guttman. Strand spaces : why is a security protocol correct ? In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 160–171, May 1998.

[39] Joshua D. Guttman, F. Javier Thayer, Jay A. Carlson, Jonathan C. Herzog, John D. Ramsdell, and Brian T. Sniffen. Trust management in strand spaces : A rely-guarantee method. In *In Proc. of the European Symposium on Programming (ESOP '04), LNCS*, pages 325–339. Springer-Verlag, 2004.

[40] Richard Hull. Web services composition : A story of models, automata, and logics. In *Proceedings of the 2005 IEEE International Conference on Services Computing - Volume 01*, pages .18–xix, Washington, DC, USA, 2005. IEEE Computer Society.

[41] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[42] David A. McAllester. Automatic recognition of tractability in inference relations. *Journal of the ACM*, 40 :284–303, 1993.

[43] J. McCarthy and S. Krishnamurth. Trusted multiplexing of cryptographic protocols. In *Sixth International Workshop on Formal Aspects in Security and Trust (FAST2009) Eindhoven, the Netherlands, November 5-6, 2009*, 2009.

[44] Jay A. McCarthy and Shriram Krishnamurthi. Cryptographic protocol explication and end-point projection. In *Proceedings of ESORICS'08*, LNCS 5283, pages 533–547. Springer, 2008.

[45] Jay A. McCarthy, Shriram Krishnamurthi, Joshua D. Guttman, and John D. Ramsdell. Compiling cryptographic protocols for deployment on the web. In *WWW '07 : Proceedings of the 16th international conference on World Wide Web*, pages 687–696, New York, NY, USA, 2007. ACM.

[46] Sebastian Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares 2009*, pages 433–440. IEEE Computer Society Press, 2009. DOI :`http://doi.ieeecomputersociety.org/10.1109/ARES.2009.95`. An extended version is available as

Technical Report no. RZ3709, IBM Zurich Research Lab, 2008, `domino.research.ibm.com/library/cyberdig.nsf`.

[47] Sebastian Mödersheim. Algebraic properties in alice and bob notation. In *ARES*, pages 433–440. IEEE Computer Society, 2009.

[48] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21 :993–999, December 1978.

[49] NESSOS : Network of Excellence on Engineering Secure Future Internet Software Services and Systems. `www.nessos-project.eu`, 2010.

[50] OASIS. ebXML standards. `http://www.ebxml.org/`.

[51] OASIS. Security Assertion Markup Language (SAML) v2.0. Available at `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security`, April 2005.

[52] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf`, 11 April, 2007.

[53] Oasis Consortium. Web Services Business Process Execution Language Version 2.0. `http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel`, 23 January, 2006.

[54] Oasis Technical Comittee on Secure Exchange. Ws-securitypolicy 1.2. `http://doc.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cd-02.pdf`, 2007.

[55] Fabio Patrizi. An introduction to simulation-based techniques for automated service composition. In *Proceedings of the Fourth European Young Researchers Workshop on Service Oriented Computing*, volume 2 of EPTCS of *YR-SOC 2009*, pages 37–49, 2009.

[56] Chris Peltz. Web services orchestration. In *HP white paper*, 2003.

[57] Chris Peltz. Web services orchestration and choreography. *Computer*, 36 :46–52, 2003.

[58] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of CSFW'01*, pages 174–190. IEEE Computer Society Press, 2001.

[59] The World Wide Web Consortium. XML Schema Definition (XSD). `http://www.w3.org/XML/Schema`, March 2005.

[60] Mohamed Anis Mekki Michaël Rusinowitch Tigran Avanesov, Yannick Chevalier. Web services verification and prudent implementation. In *4th SETOP International Workshop on Autonomous and Spontaneous Security, SETOP 2011, Leuven, Belgium, 15-16 September 2011*, 2011.

[61] Mohamed Anis Mekki Michaël Rusinowitch Tigran Avanesov, Yannick Chevalier and Mathieu Turuani. Distributed orchestration of web services under security constraints. In *4th SETOP International Workshop on Autonomous and Spontaneous Security, SETOP 2011, Leuven, Belgium, 15-16 September 2011*, 2011.

[62] Mathieu Turuani. The CL-Atse Protocol Analyser. In *Term Rewriting and Applications (Proceedings of RTA'06)*, LNCS 4098, pages 277–286, 2006.

[63] Mathieu Turuani. The CL-Atse Protocol Analyser. In *Term Rewriting and Applications (Proceedings of RTA'06)*, LNCS 4098, pages 277–286, 2006.

[64] UDDI. Introduction to UDDI : Important Features and Functional Concepts. `http://uddi.org/pubs/uddi-tech-wp.pdf`, 2004.

[65] SAP Claus von Riegen, I. Trickovic, L. Clément, A. Hately, and T. Bellwood. Using BPEL4WS in a UDDI registry. `http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.pdf`, 2004.

[66] W3C Consortium. XML Path Language (XPath) 2.0 (Second Edition). http://www.w3.org/TR/xpath20/, 14 December, 2010.

[67] World Wide Web Consortium. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 15 March, 2001.

[68] Michaël Rusinowitch Yannick Chevalier, Mohamed Anis Mekki. Orchestration under security constraints. In Stefan Hallerstede Michael Leuschel Frank de Boer, Marcello Bonsangue and Eric Madelaine, editors, *Formal Methods for Components and Objects, International Symposium, FMCO 2010, Austria, 29 November -1 December 2010*, volume 6957 of *Lecture Notes in Computer Science*. Springer, 2010.

Résumé

La composition automatique de services web est une tâche difficile. De nombreux travaux ont considérés des modèles simplifiés d'automates qui font abstraction de la structure des messages échangés par les services. Pour le domaine des services sécurisés (utilisant la signature numérique ou l'horodatage), nous proposons une nouvelle approche pour automatiser la composition des services basés sur leurs politiques de sécurité. Étant donnés, une communauté de services et un service objectif, nous réduisons le problème de la synthèse de l'objectif à partir des services dans la communauté à un problème de sécurité, où un intrus que nous appelons médiateur doit intercepter et rediriger les messages depuis et vers la communauté de services et un service client jusqu'à atteindre un état satisfaisant pour le dernier. Nous avons implémenté notre algorithme dans la plateforme de validation du projet AVANTSSAR et nous avons testé l'outil correspondant sur plusieurs études de cas. Ensuite, nous présentons un outil qui compile les traces obtenues décrivant l'exécution d'un médiateur vers le code exécutable correspondant. Pour cela nous calculons d'abord une spécification exécutable aussi prudente que possible de son rôle dans l'orchestration. Cette spécification est exprimé dans la langue ASLan, un langage formel conçu pour la modélisation des services Web liés à des politiques de sécurité. Ensuite, nous pouvons vérifier avec des outils automatiques que la spécification ASLan obtenue vérifie certaines propriétés requises de sécurité telles que le secret et l'authentification. Si aucune faille n'est détectée, nous compilons la spécification ASLan vers une servlet Java qui peut être utilisé par le médiateur pour contrôler l'orchestration.

Mots-clés: Services Web, Composition, Orchestration, Politiques de sécurité, Propriétés de sécurité, Protocoles Cryptographiques.

Abstract

Automatic composition of web services is a challenging task. Many works have considered simplified automata models that abstract away from the structure of messages exchanged by the services. For the domain of secured services (using e.g. digital signing or timestamping) we propose a novel approach to automated composition of services based on their security policies. Given a community of services and a goal service, we reduce the problem of composing the goal from services in the community to a security problem where an intruder we call mediator should intercept and redirect messages from the service community and a client service till reaching a satisfying state. We have implemented the algorithm in AVANTSSAR Platform [11] and applied the tool to several case studies. Then we present a tool that compiles the obtained trace describing the execution of a the mediator into its corresponding runnable code. For that we first compute an executable specification as prudent as possible of her role in the orchestration. This specification is expressed in ASLan language, a formal language designed for modeling Web Services tied with security policies. Then we can check with automatic tools that this ASLan specification verifies some required security properties such as secrecy and authentication. If no flaw is found, we compile the specification into a Java servlet that can be used by the mediatior to lead the orchestration.

Keywords: Web Services, Composition, Orchestration, Security Policies, Security Properties, Cryptographic Protocols.