Gérer et assurer la qualité de services de ressources dans un environnement multi-cloud

Jeremy Mechouche

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAS018

Thèse de doctorat

Gérer et assurer la qualité de services de ressources dans un environnement multi-cloud

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 09/10/2024, par

JEREMY MECHOUCHE

Composition du Jury :

Raja CHIKY Directrice des relations entreprises, 3iL Limoges, France	Rapportrice
Isabelle CHRISMENT Professeure, TELECOM Nancy, France	Rapportrice
Djamal ZEGHLACHE Professeur, TELECOM SudParis, France	Président
Sami YANGUI Maître de Conférences, INSA Toulouse, France	Examinateur
Walid GAALOUL Professeur, TELECOM SudParis, France	Directeur de thèse
Mohamed SELLAMI Maître de Conférences, TELECOM SudParis, France	Co-encadrant
Roua TOUIHRI Chercheuse associée, SAMOVAR, France	Co-encadrante

Table des matières

A la mémoire de :
Monette GUEDJ et Georges MERLOT

Remerciements :

Tout d'abord, j'aimerais remercier l'équipe de recherche autour de cette thèse. Merci Roua, d'avoir cru en ma capacité de finir cette thèse et de m'avoir dit non les deux fois où j'ai voulu arrêter. Merci Walid, de m'avoir donné la possibilité de suivre cette thèse, de m'avoir fait découvrir ce qu'était la recherche et appris à être rigoureux dans mes travaux. Merci Mohamed, de ne pas m'avoir lâché tout au long de ces 4 longues années, d'avoir lu, relu et re-relu tout ce que je pouvais écrire ou faire. L'aboutissement de ce projet de recherche est en grande partie grâce à toi. Merci à tous les trois et j'espère continuer à travailler avec vous dans de futurs projets !

Je remercie également la professeure Raja Chiky, la professeure Isabelle CHRISMENT, le professeur Djamal ZEGHLACHE et le docteur Sami YANGUI d'avoir accepté de participer à mon jury de thèse.

Merci à tous mes collègues que j'ai pu côtoyer pendant ces 5 dernières années, ceux de plus longue date qui m'ont accompagné et permis, malgré un certain nombre d'aléas, de continuer mes travaux doctoraux dans de bonnes conditions, ainsi que ceux plus récents avec qui j'ai la chance et le plaisir de travailler sur de nouveaux sujets de recherche. J'adresse un remerciement tout particulier à mes collègues du laboratoire avec qui j'ai pris plaisir à passer l'immense majorité de mon temps de rédaction et de préparation de ma soutenance.

Je remercie toutes les personnes que j'ai pu côtoyer et qui m'ont soutenues pendant ces 5 dernières années. Toutes ces relations construites ou solidifiées m'ont été indispensables pour aller au bout. Merci à tous ceux proches avec qui j'ai passé du temps, vous ne savez pas à quel point cela a été important pour moi. Merci à tous ceux plus loin avec qui le contact s'est même renforcé malgré la distance. En dépit d'une audience limitée, d'un submergeant nombre de deux, du podcast, sa diffusion quotidienne m'a permis de relâcher la pression de la thèse.

Pour finir, j'aimerais remercier ma famille. Tout d'abord, ma mère de m'avoir poussé à tenter l'expérience même si je n'avais aucune idée de ce dans quoi je me lançais. De mon père, de m'avoir toujours soutenu et poussé à aller le plus loin possible, et de ne pas lâcher, mais qui m'aurait tout autant soutenu de ne pas réussir à aller au bout. Merci à mes frères de m'avoir subi, suivi mais quand même accompagné dans ce que je faisais de plus ou moins loin. Merci à mes grands-parents, tantes, oncles, cousins, cousines et tous ceux que je considère comme faisant partie de ma famille de m'avoir supporté depuis toujours dans tout ce que je peux faire.

Merci à tous ! Pour finir, je pense que j'ai fait ce que tu m'as dit, papa, et je ne pourrais pas aller plus loin dans les études !

Chapitre 1

Introduction générale

Sommaire

À mesure que le cloud computing se développe, de nouveaux besoins clients naissent, surpassant les capacités d'un unique fournisseur. Malgré le grand nombre d'avantages apportés par le cloud computing tel que, la capacité à supporter des pics d'activité ou optimiser les coûts en adaptant les architectures d'application grâce à l'élasticité des services, ces avantages restent limités quand tous les services sont consommés auprès d'un seul fournisseur. Cette situation pourrait engendrer des problèmes [77] tels que, la compromission de la disponibilité d'une application cloud en cas de défaillance totale du fournisseur, l'augmentation des prix du fournisseur ou l'impossibilité de bénéficier des innovations d'autres fournisseurs.

Pour ces raisons, les clients se sont naturellement intéressés à la consommation de services de plusieurs fournisseurs. Dans ce contexte, le concept de *multi-cloud* émerge, offrant une réponse adaptée à ces nouveaux besoins. En effet, le multi-cloud permet aux entreprises de profiter des avantages de plusieurs fournisseurs cloud, tout en conservant une certaine flexibilité et autonomie. Ce qui ouvre un nouveau champ

de possibilités pour les applications cloud. Ces nombreux avantages dans le multi-cloud [77, 79], sont par exemple, l'optimisation des coûts en utilisant les services avec le meilleur rapport qualité/prix ou l'augmentation de la résilience en assurant un plan de continuité/reprise d'activités réparties sur plusieurs fournisseurs. Selon le rapport *State-of-the-Cloud 2024* de Flexera [86], 89% des entreprises interrogées tirent parti du multi-cloud soit une augmentation de 2% par rapport à celui de 2023.

Cependant, bien que ces avantages soient conséquents, le multi-cloud soulève de nouveaux verrous inhérents à la multiplicité de fournisseurs et à l'hétérogénéité de leurs services. En effet, chaque fournisseur a ses propres types de services et ses propres modèles de description de ces derniers. Par exemple, la performance exprimée pour un service chez *AWS* ne sera pas exprimée avec les mêmes termes chez *Microsoft Azure* ou *Google Cloud Platform*. En outre, la gestion de la qualité de service, le maintien et le suivi des objectifs de niveau de service, sont rendues d'autant plus complexe en raison de la nature distribuée du contexte multi-cloud et de la dépendance qui existe entre les différents composants [88].

1.1 Contexte de recherche

Dans les environnements multi-cloud, la définition et le suivi des objectifs de niveau de service (*angl. Service Level Objective, SLO*) revêtent une importance cruciale afin de fournir un service de qualité aux utilisateurs finaux et de garantir que les services fournis par les différents fournisseurs correspondent aux attentes client. Les SLOs définissent les attentes en termes de performance, de disponibilité et d'autres critères de qualité pour chaque service cloud. Les SLOs composent les accords de niveaux de services (*angl. Service Level Agreement, SLA*) qui engagent le fournisseur sur un niveau de service à fournir. En raison de la diversité des fournisseurs, des infrastructures et du caractère distribué d'un environnement multi-cloud, la définition et le suivi des SLAs est complexe. Également, pour garantir le respect des SLOs dans le contexte cloud un élément essentiel est la reconfiguration dynamique des services sous forme de stratégies. Ces stratégies de reconfiguration déterminent comment les ressources sont allouées, ajustées ou libérées en fonction des variations de charge, des pannes et d'autres facteurs. Ces derniers sont particulièrement complexes à mettre en œuvre dans un environnement multi-cloud en raison de sa nature distribuée et hétérogène.

Aussi, à l'issue de la mise en œuvre des SLAs, il est essentiel de les vérifier pour assurer que les services fournis répondent aux attentes et aux exigences définies. La mise en œuvre de mécanismes de vérification permet non seulement de s'assurer que les SLAs sont respectés, mais aussi d'identifier précisément tout écart potentiel. Cette vérification devient particulièrement complexe dans un contexte de SLA multi-cloud en raison de la quantité et de l'hétérogénéité des données fournies par les différents fournisseurs de services cloud. Par conséquent, l'adoption d'une approche systématique et automatisée pour la validation et la vérification du bon déroulement des SLAs est cruciale pour appréhender efficacement la complexité du multi-cloud

et assurer un service cloud cohérent et fiable.

Dans les prochaines sections, nous identifions les questions de recherches (Section 1.2). De ces questions découlent les objectifs de recherche (Section 1.3) qui sont motivés dans un exemple (Section 1.4), puis adressés par nos contributions (Section 1.5).

1.2 Problématiques

Dans cette thèse, nous nous intéressons donc à la gestion des SLAs dans le contexte multi-cloud. Plus précisément, nous cherchons à traiter les questions de recherche suivantes :

(RQ1) Comment modéliser les SLAs multi-cloud considérant la dynamicité des services ? De plus, ce paradigme multi-cloud est caractérisé par plusieurs particularités telles que l'hétérogénéité et son caractère distribué sur plusieurs fournisseurs cloud [77], il est obligatoire de considérer ces spécificités pour modéliser un SLA multi-cloud dynamique [33, 39, 55].

(RQ2) Comment vérifier et suivre la mise en œuvre des SLAs multi-cloud considérant la dynamicité des services ? Cela implique de vérifier que les SLAs multi-cloud dynamique sont définis de façon cohérente [70]. Ensuite, il convient de vérifier que les niveaux de services fournis correspondent à ceux définis [33].

1.3 Objectifs

Pour adresser les 2 questions de recherche identifiées ci-dessus, nous définissons dans cette thèse les 3 objectifs suivants à traiter.

1.3.1 Modélisation de SLA multi-cloud dynamique

Notre premier objectif **(O1)** est de proposer un modèle de description de SLA qui répond aux verrous d'hétérogénéité et de granularité liés à la gestion des niveaux de service dans un environnement multi-cloud considérant la dynamicité en relation avec **(RQ1)**. Les modèles de SLA existants tels que WSLA [51], SLAC [90], rSLA [62] ou ySLA [32] ne permettent pas de représenter toutes les caractéristiques d'un SLA multi-cloud dynamique. En effet, il est nécessaire pour représenter dans son intégralité un SLA multi-cloud de prendre en compte des objectifs globaux à l'application (*Global-SLA*) et des objectifs locaux au niveau des composants (*Sub-SLA*) formant un système multi-cloud. En plus de ces objectifs de différents niveaux, il est nécessaire pour construire un modèle de description précis et complet de spécifier les parties prenantes, telles que les clients et les fournisseurs de services cloud, en

prenant en compte les exigences variées de performance, de disponibilité, de sécurité et de coût. Nous cherchons également à prendre en compte les mécanismes de reconfiguration des services cloud, permettant d'adapter efficacement aux fluctuations de la charge de travail et d'assurer le respect continu des SLAs. L'introduction de stratégies de reconfiguration composées de plusieurs mécanismes de reconfiguration ajoute également une couche de complexité supplémentaire.

1.3.2 Contrôle de SLA multi-cloud dynamique

Bien que l'on dispose d'un modèle formel pour la description des SLAs dynamiques dans un environnement multi-cloud, il existe un risque d'incohérence induite par l'architecte lors de leur définition. De plus, la mise en œuvre effective des SLAs par les fournisseurs de services cloud peut ne pas correspondre aux spécifications initialement établies. Par conséquent, il est essentiel d'instaurer des mécanismes de contrôle, tant en amont (pré-mise en œuvre) qu'en aval (post-mise en œuvre), afin d'assurer la cohérence du SLA multi-cloud défini et l'adéquation entre sa définition et sa mise en œuvre. Notre second objectif **(O2)** répondant à **(RQ2)** porte sur le contrôle des SLAs multi-cloud dynamique et se décompose en 2 sous-objectifs : la validation pré-mise en œuvre **(O2.1)** et la vérification post-mise en œuvre**(O2.2)**.

1.3.2.1 Validation pré-mise en œuvre

Notre second objectif **(O2.1)** est de proposer un processus de validation de la cohérence préalable à la mise en œuvre des SLAs multi-cloud et des stratégies de reconfiguration. Ces derniers étant définis par des architectes cloud, cette validation préalable vise à assurer que les SLAs définis et les stratégies de reconfiguration associées sont conformes et réalisables avant leur mise en œuvre concrète. Cette validation reste un processus complexe basé sur les interactions entre les différents SLOs et les stratégies de reconfiguration [28, 78]. En développant ce processus de validation, nous visons à identifier les incohérences potentielles ou les incompatibilités qui pourraient survenir, minimisant ainsi les risques de violations des SLAs préalablement à la mise en œuvre des SLAs.

1.3.2.2 Vérification post-mise en œuvre

Notre troisième objectif **(O2.2)** porte sur la vérification post-mise en œuvre des SLAs multi-cloud. Une fois un SLA multi-cloud défini et validé, ce dernier est mis en œuvre par les différents fournisseurs de services cloud. Il est nécessaire de s'assurer que ces derniers respectent leurs engagements définis dans le SLA. L'objectif est d'évaluer rétrospectivement la conformité entre ce qui s'est réellement passé et ce qui était défini. Une éventuelle divergence peut être le signe d'une violation du SLA multi-cloud défini. La problématique se trouve dans la nécessité de mettre en place un mécanisme de vérification qui valide la conformité entre les services attendus et les services fournis dans le contexte multi-cloud.

1.4 Exemple de motivation

Pour motiver et illustrer nos travaux, nous utilisons un exemple décrivant une application multi-cloud respectant les principes des modèles d'application cloud composite [1]. Cette application est basée sur trois composants : **Interface Utilisateur** (*UI*), **Authentification** (*Auth*) et **Stockage** (*Stor*). Ces composants sont fournis par trois fournisseurs de service cloud différents : CSP_1, CSP_2 et CSP_3.

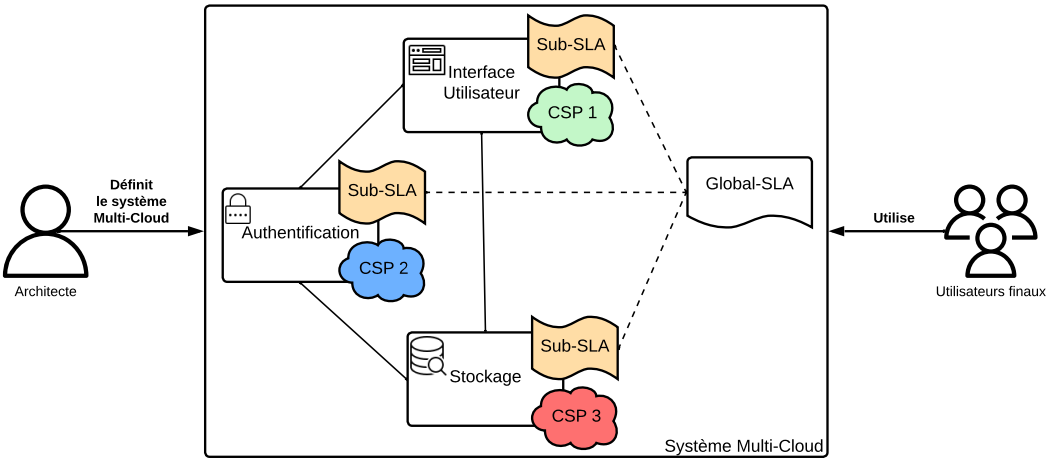


FIGURE 1.1 – Système multi-cloud d'exemple de motivation

On considère qu'un architecte cloud est en charge du déploiement de ces ressources requises et doit respecter des contraintes de disponibilité, de temps de réponse et de coût. La charge de travail des utilisateurs et le contexte cloud sont par essence fluctuants. Par conséquent, afin de gérer les pics d'activités planifiées et non planifiées, l'architecte cloud définit trois stratégies de reconfiguration différentes associées aux composants des applications. Ces stratégies sont désignées comme étant pour les besoins **normaux**, **élevés** et **faibles**. Un global-SLA indique les exigences globales de niveau de service pré-établies des utilisateurs finaux pour un temps de réponse inférieur à 5 ms, une disponibilité supérieure à $99,7\%$ et un coût par heure inférieur à 10€/h.

Aussi, à chaque composant de cette application est associé un sub-SLA définissant les objectifs de niveau de service locaux associés. Pour cet exemple, nous considérons uniquement le sub-SLA *Interface Utilisateur* pour simplifier les choses. Ce sub-SLA associé au composant d'interface utilisateur est composé de trois SLOs : (*i*) un temps de réponse inférieur à 4 ms, (*ii*) un coût par heure inférieur à 2€/h et (*iii*) un taux de disponibilité de 99,9%. Pour maintenir ces SLOs, l'architecte tirera parti des mécanismes d'élasticité du cloud pour implémenter les stratégies de reconfiguration et ainsi s'assurer que la configuration de l'application s'adaptera à la charge utilisateur. Les trois stratégies de reconfiguration pour l'interface utilisateur (UI) sont définies comme suit : $normalNeeds$ avec 2 machines virtuelles, $highNeeds$ avec 4 machines virtuelles, $lowNeeds$ avec 1 machine virtuelle et $final$ avec 0 machine virtuelle. Pour mettre en œuvre de telles stratégies de

reconfiguration et évaluer leur conformité avec le global-SLA de l'application multi-cloud, il est nécessaire de : (*i*) représenter formellement les exigences de l'application multi-cloud sous forme d'un SLA multi-cloud exprimant l'accord entre les différentes parties impliquées, (*ii*) s'assurer que la stratégie de reconfiguration associée à un sub-SLA respecte le global-SLA et (*iii*) vérifier que le service fourni correspond au SLA convenu.

En effet, ces stratégies de reconfiguration sont définies par un architecte et sont donc sujettes aux erreurs, c'est-à-dire qu'elles peuvent éventuellement ne pas être conformes aux exigences de l'utilisateur final. Ces stratégies de reconfiguration incorrectes peuvent résulter d'une mauvaise définition des ressources ou d'un changement de charge non pris en compte pour une application multi-cloud ou l'un de ses composants. De telles stratégies incorrectes peuvent entraîner une violation du SLA, ce qui peut avoir des conséquences négatives pour un fournisseur d'application cloud, telles que des dommages à la réputation ou des pénalités.

On note deux niveaux d'incohérence potentiels. Premièrement entre les global et les sub-SLAs, par exemple, un objectif de disponibilité global défini à 99.999% et un objectif local défini à 98% entraînera potentiellement une violation. Deuxièmement, entre les SLOs et les stratégies de reconfiguration, par exemple, une stratégie définie sur un mauvais événement ne déclenchera pas la stratégie ce qui entraînera également une potentielle violation.

Afin d'éviter de tels problèmes, le comportement de l'application et de ses composants (décrits par la stratégie de reconfiguration) doivent être pris en compte dans la modélisation du SLA **(O1)**. La définition de ces objectifs et des stratégies de reconfiguration doit être vérifiée, en amont, pour éviter la mise en œuvre d'objectifs et de stratégies de reconfiguration incohérente **(O2.1)** et, en aval, pour assurer la conformité des objectifs et des stratégies de reconfiguration mis en œuvre **(O2.2)**.

1.5 Contributions

Pour atteindre les objectifs ci-dessus tout en traitant les questions de recherche, nous introduisons trois approches complémentaires permettant la modélisation et la vérification des SLAs multi-cloud dynamique. Ces trois contributions sont illustrées dans la Figure 1.2 et détaillées dans les sections 1.5.1, 1.5.2 et 1.5.3.

1.5.1 Modèle de description de SLA multi-cloud intégrant des stratégies de reconfiguration

Notre première contribution, répondant à **O1**, porte sur la proposition d'un modèle de description de SLA multi-cloud. Ce modèle vise à répondre aux verrous liés à la gestion de l'hétérogénéité, de la granularité et de la dynamicité des services dans un environnement multi-cloud. En développant ce modèle de description, nous cherchons à fournir un modèle formel et précis qui permet de spécifier les engage-

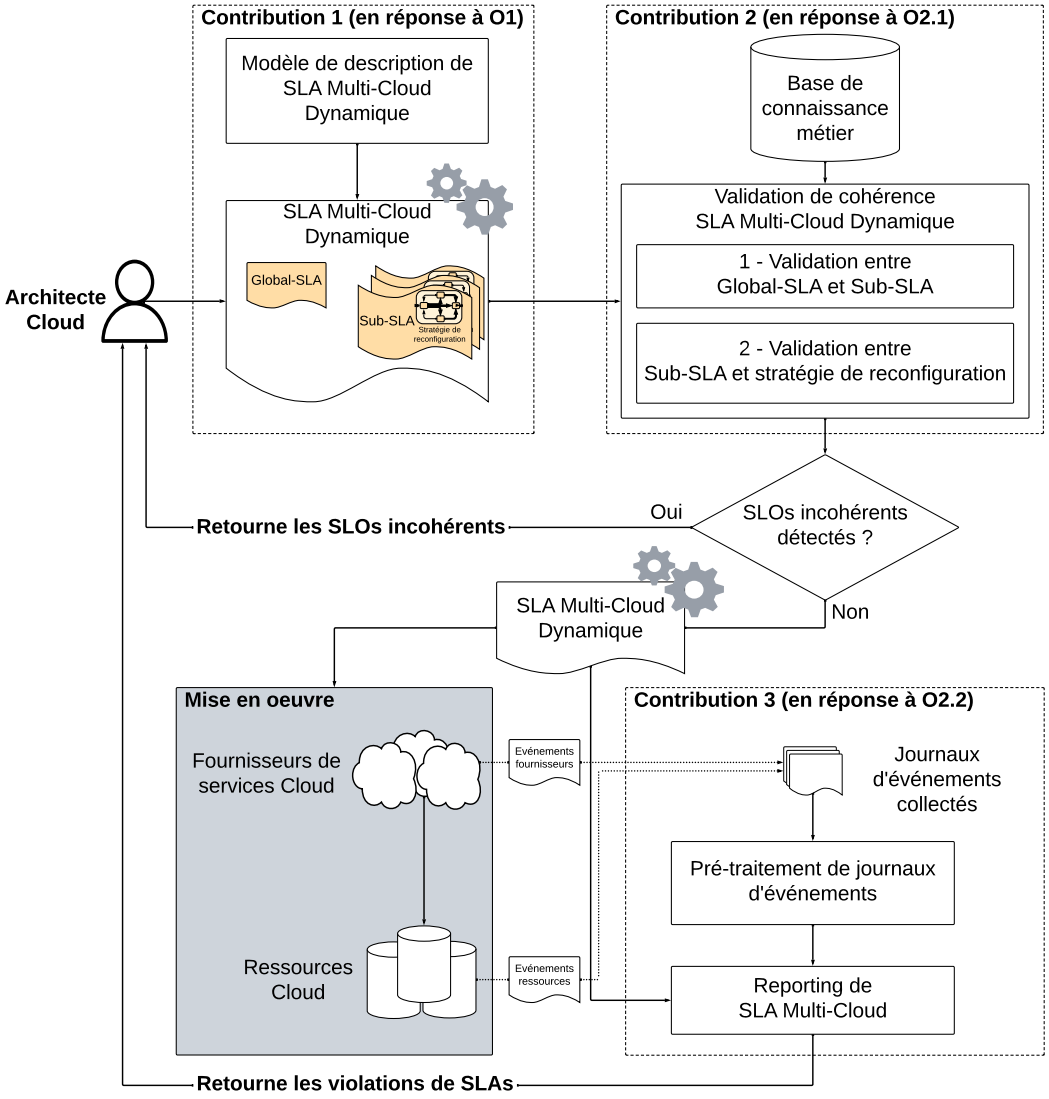


FIGURE 1.2 – Approche de gestion et d'assurance de SLA multi-cloud dynamique

ments entre les parties prenantes, tels que les clients et les fournisseurs, en prenant en compte les exigences variées de performances, de disponibilité, de sécurité et de coûts. De plus, nous cherchons également à prendre en compte les mécanismes de reconfiguration dynamique, permettant de s'adapter efficacement aux fluctuations de la charge de travail et d'assurer le respect continu des SLA.

Pour ce faire, nous proposons une description des différents SLOs requis par les utilisateurs finaux, en tenant compte des multiples composants constitutifs d'un système multi-cloud. Ce modèle traite par la modélisation de stratégie de reconfiguration la nécessité pour le maintien de ces SLOs de considérer les fluctuations de la charge de travail à traiter. En effet, il peut être nécessaire de reconfigurer les différents composants pour leur permettre de supporter la charge de travail et de maintenir certains SLOs. Cette reconfiguration peut être nécessaire en cas de pics d'activité, de pannes, etc. Toute conception de système multi-cloud doit, par conséquent, tenir compte de cette dynamicité pour maintenir des SLOs [76].

Pour modéliser les stratégies de reconfiguration, nous proposons d'utiliser le formalisme des machines à états [40]. Ce modèle permettra donc aux architectes cloud de modéliser leur SLA Multi-cloud dynamique au format YAML incluant les objectifs de différents niveaux et les comportements dynamiques via une machine à état.

1.5.2 Validation pré-mise en œuvre de la cohérence des composants d'un SLA multi-cloud dynamique

Notre seconde contribution, répondant à l'**O2.1**, porte sur la validation préalable à la mise en application de la cohérence des SLAs multi-cloud et des stratégies de reconfiguration. Cette validation préalable permet d'assurer que les SLAs définis et les stratégies de reconfiguration associées sont conformes et réalisables avant leur mise en œuvre concrète. En développant cette approche de validation automatique, nous visons à identifier et à résoudre les incohérences potentielles ou les incompatibilités qui pourraient survenir, minimisant ainsi les risques de violations des SLAs. Cette validation est effectuée en deux temps : d'abord entre les SLOs composant le global-SLA et les SLOs composant les sub-SLAs puis entre les SLOs composant les sub-SLAs et les stratégies de reconfiguration. Ces validations de cohérence sont basée sur l'adaptation des méthodes d'agrégation de SLA [88] proposées dans le cadre de SLA hiérarchique [36], ainsi que sur une méthode de traduction de SLA (*angl. SLA translation*) [61]. Grâce à cette validation préalable, nous cherchons à anticiper des violations lors de la mise en œuvre des SLAs et de l'exécution des stratégies de reconfiguration associées.

1.5.3 Vérification post-mise en œuvre de SLA multi-cloud dynamique

La troisième contribution, répondant à l'**O2.2**, est dédiée à la vérification post-mise en œuvre de SLA multi-cloud et des stratégies de reconfiguration. L'objectif est d'évaluer rétrospectivement la conformité des engagements contractuels. Grâce à cette vérification, des déviations éventuelles par rapport aux objectifs définis peuvent être identifiées, permettant ainsi de détecter les points d'amélioration et de prendre des mesures correctives améliorant ainsi la qualité des services rendus. Pour ce faire, nous tirons parti des techniques de fouille de processus (*angl. Process Mining*) [2] en analysant des journaux d'événements collectés au niveau des fournisseurs de service cloud, des orchestrateurs de ressources et des ressources fournies. Ces journaux d'événement nécessitent un pré-traitement pour pouvoir être comparé à notre modèle de description de SLA multi-cloud puis être utilisé dans une technique de vérification de conformité [3] (*angl. conformance checking*). Ainsi, cette comparaison permettra d'identifier d'éventuelles déviations et éventuellement réclamer des compensations en cas de violation de SLA défini.

1.6 Publications

1. Jeremy Mechouche, Roua Touihri, Mohamed Sellami, Walid Gaaloul : Towards higher-level description of SLA-aware reconfiguration strategies based on state machine, IEEE International Conference on E-Business Engineering, Guangzou, China, 2021
2. Jeremy Mechouche, Roua Touihri, Mohamed Sellami, Walid Gaaloul : Conformance Checking for Autonomous Multi-Cloud SLA Management & Adaptation, Journal of Supercomputing 78(11) : 13004-13039, 2022
3. Jeremy Mechouche, Mohamed Sellami, Zakaria Maamar, Roua Touihri, Walid Gaaloul : Process mining approach for Multi-Cloud SLA Reporting, IEEE International Conference on Big Data, Dec 15-18, 2023 @ Sorrento, Italy.

1.7 Plan de la thèse

Ce manuscrit de thèse de doctorat est organisé en 6 chapitres :

Chapitre 2 : Notions fondamentales et État de l'art propose une exploration et une analyse approfondie de l'état de l'art autour de nos trois objectifs. Nous commençons par introduire les concepts clés du cloud et du multi-cloud. Puis, nous explorons les différentes propositions concernant les SLAs dans les contextes cloud et multi-cloud. Nous présentons ensuite les stratégies de reconfiguration multi-cloud, soulignant comment elles exploitent le concept d'élasticité dans le cloud. Pour conclure, nous analysons les travaux existants sur le contrôle de la conformité des SLA, aussi bien pré que post-mise en œuvre.

Chapitre 3 : Description de SLA multi-cloud dynamique présente notre première contribution répondant à notre objectif **(O1)** de modèle de description de SLA multi-cloud dynamique permettant leur description par des architectes cloud.

Chapitre 4 : Cohérence de SLA multi-cloud dynamique introduit notre deuxième contribution répondant à notre objectif **(O2.1)** de validation de cohérence pré-mise en œuvre de SLA multi-cloud dynamique.

Chapitre 5 : Reporting de SLA multi-cloud dynamique introduit notre troisième contribution répondant à notre objectif **(O2.2)** de vérification post-mise en œuvre de SLA multi-cloud dynamique.

Chapitre 6 : Conclusion résume les contributions proposées et présente un aperçu des perspectives potentielles que nous entendons aborder à court et moyen terme.

Chapitre 2

Notions fondamentales et État de l'art

Sommaire

Dans cette thèse, nous cherchons à adresser trois objectifs : **(O1)** la modélisation de SLAs multi-cloud dynamique, **(O2.1)** la validation pré-mise en œuvre de SLA multi-cloud dynamique et **(O2.2)** la vérification post-mise en œuvre de SLA multi-cloud dynamique. Par conséquent, nous examinons, dans ce chapitre, les approches existantes dans la littérature pertinente à ces sujets. L'objectif est de positionner notre travail par rapport à ces approches tout en mettant en évidence les enjeux sous-jacents de cette thèse.

Dans le but de faciliter la présentation de ces approches, nous présentons tout d'abord dans la section 2.1, quelques notions fondamentales relatives au domaine du cloud et du multi-cloud. Ensuite, dans la section 2.2, nous passons en revue les langages existants pour la modélisation des SLAs multi-cloud dynamique. Dans la section 2.3, nous examinons les approches proposées pour la validation pré-mise en œuvre et la vérification post-mise en œuvre des SLAs multi-cloud dynamique.

2.1 Notions fondamentales : Cloud et multi-cloud

En 1961, John McCarthy conceptualise l'informatique utilitaire où la puissance de calcul, au même titre que le téléphone, le gaz, l'eau ou l'électricité serait un service d'utilité publique. Près de 45 ans plus tard, en 2006, Amazon propose ses premiers services cloud avec **Amazon Elastic Compute Cloud** (abrégé EC2). Après une période de flottement et de structuration du marché de quelques années, le *National Institute of Standards and Technology* [1](abrégé NIST) formalise une définition du concept de cloud computing en 2011 [68]. Cette dernière pose les bases de la nature d'un service cloud, en détaillant les différents modèles de services, les stratégies de déploiement, ainsi que les principaux acteurs du secteur.

2.1.1 Cloud computing

En se basant sur la définition proposée par le NIST dans [68], nous présentons dans la suite de cette section les caractéristiques d'un service cloud, les modèles de services, les modèles de déploiement puis les acteurs qui contribuent à la fourniture de services cloud.

2.1.1.1 Caractéristiques d'un service cloud

Les 5 caractéristiques essentielles d'un service pour être qualifié de cloud sont :

Service à la demande - Un consommateur doit pouvoir demander la fourniture de services informatiques (tel que le stockage ou le calcul) sans interaction humaine avec le fournisseur ;

1. `https://www.nist.gov/`

Accessible largement par réseau - Les services fournis doivent être accessibles via un réseau (par exemple, Internet ou réseau large d'entreprise) et utilisables par différents clients et plateformes ;

Mise en commun des ressources - Les fournisseurs de service doivent mettre en commun les ressources informatiques afin de servir plusieurs consommateurs en utilisant différentes techniques de mutualisation ;

Élasticité rapide - Les fournisseurs de services doivent offrir des ressources virtuellement infinies, correspondant toujours aux besoins des consommateurs. De plus, ces ressources doivent pouvoir être rapidement et automatiquement augmentées ou réduites ;

Service mesuré - Les fournisseurs de services doivent gérer et optimiser de manière autonome l'utilisation des ressources. Cette utilisation doit être surveillée, contrôlée, mesurée et rapportée pour assurer la transparence pour le fournisseur et le consommateur du service. Cela permet ainsi de facturer aux clients uniquement ce qu'ils ont consommé.

2.1.1.2 Modèles de services

Les 3 modèles de services, illustré dans la Figure 2.1 sont : Infrastructure-as-a-Service (IaaS), Plaform-as-a-Service (PaaS) et Software-as-a-Service (SaaS). La principale différence entre ces modèles se trouve dans les responsabilités portée par le client et le fournisseur de service. En effet, on peut observer dans ce schéma les responsabilités portées par le consommateur en vert et celles portées par le fournisseur en blanc.

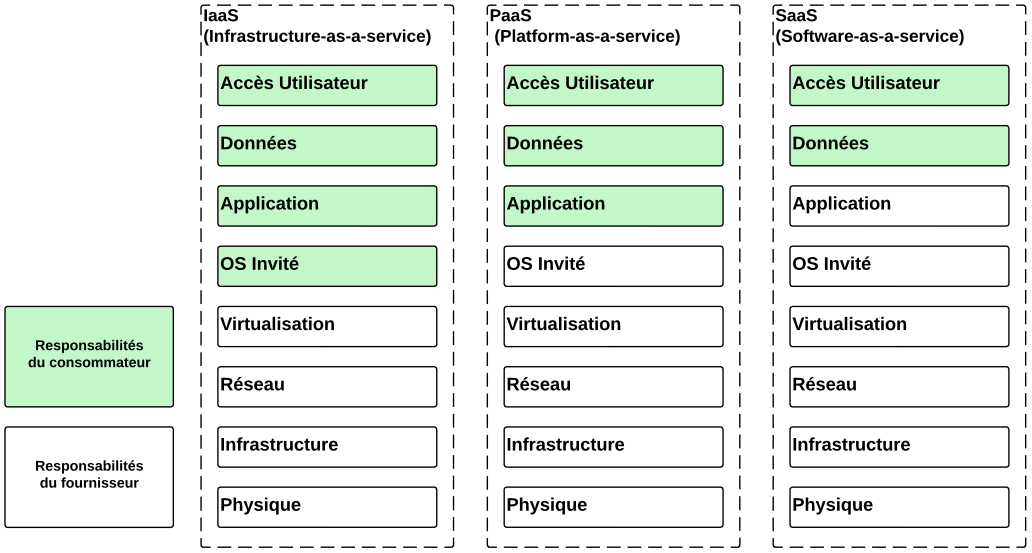


FIGURE 2.1 – Modèle de services du cloud computing

IaaS - Infrastructure-as-a-Service est un modèle qui fournit des services de calcul, de stockage et de réseau. L'utilisateur ne gère ni ne contrôle l'infrastructure où le service est fourni, mais a cependant le contrôle sur le système

d'exploitation, les applications installées et un contrôle limité sur les composants réseau. Le fournisseur de services gère toute l'infrastructure, tandis que le client est uniquement responsable des aspects du déploiement du système. On retrouve des services IaaS chez tous les fournisseurs de service cloud tel que AWS avec EC2 [2], GCP avec GCE [3] ou Azure avec Compute [4].

PaaS - Platform-as-a-Service est un modèle qui permet aux clients d'exécuter des applications d'infrastructure dans le Cloud. Ces applications peuvent être créées à l'aide de langages, de bibliothèques et d'outils pris en charge par le fournisseur de services. L'utilisateur n'a aucun accès ni contrôle sur l'infrastructure cloud, y compris le réseau, les serveurs, le système d'exploitation ou le stockage. Comparée aux environnements traditionnels de développement d'applications, l'utilisation du PaaS peut entraîner une réduction du temps de développement et offre des dizaines d'outils et de services permettant une élasticité rapide des applications. On retrouve des services PaaS chez tous les fournisseurs de service cloud tel que AWS avec Elastic Beanstalk [5], GCP avec App Engine [6] ou Azure avec App Service [7].

SaaS - Software-as-a-Service est un modèle qui fournit des systèmes logiciels à des fins spécifiques accessibles aux utilisateurs à partir de plusieurs appareils via une interface client. Dans l'architecture SaaS, les utilisateurs ne gèrent ni ne contrôlent l'infrastructure, ils accèdent directement à l'application. La seule responsabilité des utilisateurs est d'envoyer et de gérer les données que l'application traitera et les étapes d'interaction avec l'application. Ainsi, de nouvelles fonctionnalités peuvent être incorporées automatiquement dans les systèmes logiciels sans que les utilisateurs ne remarquent ces actions, rendant transparents le développement et la mise à jour des systèmes. Tout le reste relève de la responsabilité du fournisseur de service cloud.

2.1.1.3 Modèles de déploiement

Les 4 principaux modèles de déploiement de services possibles sont : privé, public, hybride et communautaire. Ces derniers sont illustrés avec leurs interactions avec les utilisateurs finaux dans la Figure 2.2.

Privé - est utilisé exclusivement par une seule organisation. Il est généralement hébergé en interne, au sein de l'organisation elle-même. Sa gestion peut être assurée soit par l'organisation elle-même, soit par un prestataire tiers.

Public - est accessible au grand public, que ce soit un grand groupe ou tout autre type d'organisations. Il est géré par un fournisseur de services cloud, qui met à disposition des services cloud. Par exemple, AWS, Azure et GCP sont des fournisseurs de services d'infrastructure de cloud public.

2. `https://aws.amazon.com/fr/ec2/`
3. `https://cloud.google.com/products/compute`
4. `https://azure.microsoft.com/fr-fr/products/category/compute`
5. `https://aws.amazon.com/fr/elasticbeanstalk/`
6. `https://cloud.google.com/appengine`
7. `https://azure.microsoft.com/en-us/products/app-service`

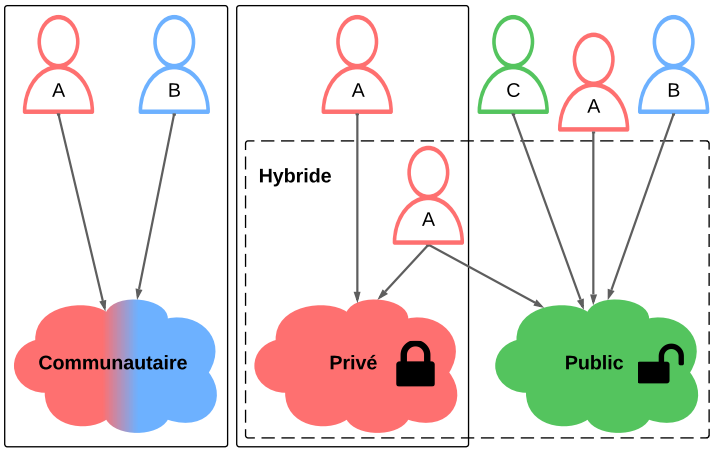


FIGURE 2.2 – Modèles de déploiement cloud

Hybride - est la combinaison de deux ou plusieurs types de modèle de déploiement. Une organisation peut créer un cloud privé pour ses applications qui sont étroitement intégrées à des systèmes existants et nécessitent une mise à l'échelle que le cloud privé ne peut pas fournir.

Communautaire - est un modèle partagé entre plusieurs organisations pour répondre à des besoins spécifiques tels que des missions collaboratives, des exigences de sécurité ou des politiques particulières. Il est conçu pour répondre aux exigences d'une communauté ou d'autres types d'entreprises spécifiques, par exemple, des prestataires pour l'armée, des institutions de recherche, etc.

2.1.1.4 Acteurs du cloud computing

Les différents acteurs du cloud et leurs interactions sont : le consommateur, le fournisseur, le transporteur, l'auditeur et le courtier.

Consommateur - (*angl. consumer*) est une organisation ou une entité qui utilise les services des fournisseurs ;

Fournisseur - (*angl. provider*) est une organisation ou une entité qui fournit un ou plusieurs services disponibles aux parties intéressées. Par exemple : AWS, Microsoft Azure et GCP ;

Transporteur - (*angl. carrier*) est un tiers impliqué dans la fourniture du service qui assure la connectivité et transporte les services des fournisseurs vers les consommateurs. De nombreux transporteurs sont des entreprises de télécommunications telles qu'Orange, Bouygues ou Colt ;

Auditeur - (*angl. auditor*) est une partie indépendante qui évalue le service cloud fourni en termes, par exemple, de niveau de sécurité ou de performance ;

Courtier - (*angl. broker*) est une entité qui négocie les relations entre les fournisseurs et les consommateurs et gère l'utilisation et la fourniture du service.

2.1.2 Multi-cloud

Le multi-cloud [77] est un concept qui fait référence à l'utilisation séquentielle ou simultanée et cohérente de services cloud provenant de plusieurs fournisseurs différents. Il s'agit d'une approche où les utilisateurs et les organisations peuvent exploiter des ressources et des services provenant de divers environnements cloud, tels que des fournisseurs publics, privés ou hybrides.

Dans un environnement multi-cloud, les utilisateurs peuvent choisir et combiner des services de cloud computing de manière flexible en fonction de leurs besoins spécifiques, sans être liés à un seul fournisseur. Cela leur offre une plus grande flexibilité, une meilleure évolutivité et une réduction des risques de dépendance à l'égard d'un fournisseur unique. Le multi-cloud implique des défis tels que l'interopérabilité entre différents services cloud, la gestion de la sécurité et des politiques de données sur plusieurs plates-formes, ainsi que l'optimisation des coûts et des performances dans un environnement hétérogène. Cependant, cette approche offre également de nombreux avantages en termes de résilience, de redondance, de diversité des fonctionnalités et de possibilités de personnalisation.

Les principales raisons de l'utilisation du multi-cloud [77], en fonction du type d'utilisation, sont classifiées dans le Tableau 2.1. Concernant l'utilisation séquentielle du multi-cloud, c'est-à-dire l'emploi de plusieurs ressources de manière non simultanée, cela permet de réagir aux évolutions des offres des fournisseurs et de changer de fournisseur si les conditions tarifaires ou les performances deviennent défavorables pour le consommateur. Pour l'utilisation simultanée des ressources chez plusieurs fournisseurs, cette approche permet de consommer des services qui sont plus adaptés chez un autre fournisseur et conserver le reste de notre infrastructure chez un fournisseur principal.

Afin de garantir la qualité des différents services cloud selon un standard précis, il est essentiel de contractualiser les attentes. Cette démarche se concrétise par la mise en place de SLA, sujet que nous approfondissons dans la section suivante de ce chapitre.

2.2 Modélisation de SLA multi-cloud dynamique

Dans cette section, nous cherchons à adresser notre objectif **(O1)** de modélisation de SLA multi-cloud dynamique. Nous commençons, dans la section 2.2.1, par la modélisation des SLAs multi-cloud, en explorant leur cycle de vie, les langages de spécification de SLA, ainsi que les langages spécifiques aux SLA multi-cloud. Ensuite, nous abordons, dans la section 2.2.2, la modélisation de la dynamicité dans ces SLA, en examinant l'élasticité dans le contexte cloud et comment elle contribue au maintien des SLOs.

Tableau 2.1 – Détails des raisons d'utiliser une approche multi-cloud

Type d'utilisation	Raisons
Séquentielle	Réagir à l'évolution de l'offre des fournisseurs
	Prendre en compte des contraintes comme l'emplacement ou la restriction légale
	Éviter la dépendance à un seul fournisseur externe
	Assurer la sauvegarde pour faire face aux catastrophes ou la planification de l'inactivité
	Optimiser les coûts ou améliorer la qualité des services
Simultanée	Consommation de différents services en fonction de leurs particularités qu'on ne trouve pas ailleurs
	Réplication d'application ou service en utilisant différents fournisseurs pour garantir leur disponibilité
	Faire face aux pics d'activités et à la demande de ressources en utilisant des services externes à la demande
	Amélioration de sa propre infrastructure cloud en passant des accords avec d'autres fournisseurs
	Agir comme un intermédiaire

2.2.1 Modélisation de SLA multi-cloud

Le SLA est un document ("contrat") qui définit la qualité de service et la prestation définie entre un fournisseur de service et un client. Autrement dit, il s'agit de clauses contractuelles définissant précisément le service à fournir et le niveau de service que souhaite obtenir un client et sur lequel le fournisseur est prêt à s'engager. Ce document fixe les responsabilités des différentes parties prenantes au contrat. Différents cycles de vie pour le SLA ont été proposés dans la littérature, nous étudions les propositions les plus pertinentes dans la section 2.2.1.1. Ensuite, dans la section 2.2.1.2, nous présentons différents langages de spécification de SLA existant. Dans la section 2.2.1.3, nous présentons les propositions faites pour la spécification de SLA multi-cloud. Nous concluons, dans la section 2.2.1.4, par une discussion présentant les manques identifiés pour répondre à notre premier objectif **(O1)** de modélisation de SLA multi-cloud dynamique.

2.2.1.1 Cycle de vie de SLA

Différentes propositions ont été faites au fil des années pour décrire le cycle vie du SLA. Ron et al. [85] proposent en 2001 un cycle de vie du SLA en trois étapes, illustré dans la Figure 2.3 :

1. **Création** : Le SLA est créé lorsqu'un client souscrit à un service proposé

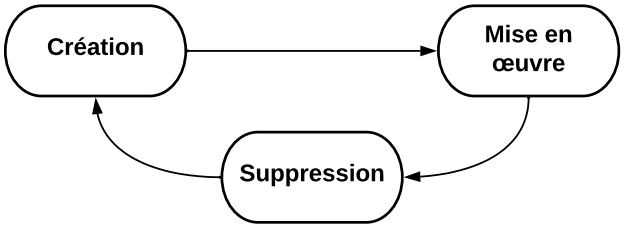


FIGURE 2.3 – Cycle de vie du SLA en 3 étapes proposé par Ron et al. [85]

par un fournisseur. Ce processus débute par une chaîne d'événements menant le client à découvrir et évaluer le service pour déterminer s'il répond à ses besoins. La création du SLA implique l'établissement officiel et juridiquement contraignant de l'accord, la configuration des sous-systèmes de service nécessaires et déclenche la planification des services ;

2. **Mise en œuvre** : Le service est fourni au client. Ce dernier a accès en ligne aux termes statiques et en lecture seule du SLA, ainsi qu'aux paramètres dynamiques et en lecture seule du service, y compris les données de performance qui l'intéressent. Le client pourra éventuellement modifier les paramètres du SLA dans des conditions préalablement convenu avec le fournisseur ce qui entraînera généralement des frais supplémentaires pour le service ;
3. **Suppression** : Lorsque le SLA arrive à échéance, il est nécessaire de supprimer le SLA ainsi que toutes les informations de configuration liées au service. De plus, ces informations de configuration doivent être modifiées ou éliminées, et les ressources allouées pour ce service doivent être libérées.

Ce cycle de vie présente les principales étapes que la fourniture de services doit suivre. Cependant, ce modèle demeure trop simpliste et présente une granularité trop large pour représenter adéquatement toutes les phases du cycle de vie d'un SLA. C'est pourquoi, le groupe Sun Micro Systems Internet Data Center [83] a proposé un cycle de vie de SLA plus spécifique en six étapes, illustré dans la Figure 2.4 :

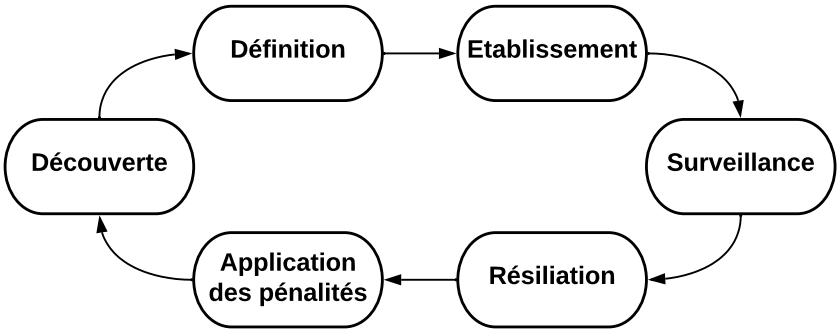


FIGURE 2.4 – Cycle de vie du SLA en 6 étapes proposé par Sun Micro Systems [83]

1. **Découverte des fournisseurs de services** : où les fournisseurs de services sont identifiés en fonction des besoins des consommateurs.

2. **Définition du SLA** : inclut la définition des services, des parties prenantes, des politiques de pénalité et des paramètres de qualité de service. À cette étape, les parties peuvent négocier pour parvenir à un accord.
3. **Établissement de l'accord** : un modèle de SLA est établi et rempli par un accord spécifique et les parties prenantes s'engagent formellement via cet accord. Le service est alors fourni à cette étape.
4. **Surveillance du respect du SLA** : la performance de la prestation du fournisseur est mesurée par rapport au contrat.
5. **Résiliation du SLA** : se termine en raison d'une arrivée à échéance du SLA ou d'une violation de ses obligations par l'une des parties.
6. **Application des pénalités en cas de violation de SLA** : si l'une des parties viole les conditions du contrat, les clauses de pénalité correspondantes sont invoquées et exécutées.

Bien que ce modèle soit plus précis que le précédent, il ne permet toujours pas de représenter intégralement toutes les étapes et les interactions entre les parties prenantes à la définition du SLA. Pour cela Kritikos et al. proposent, dans [55], un cycle de vie basé sur le cycle de vie des services en huit étapes, illustré dans la Figure 2.5, intégrant la définition de modèles de SLA prêts à l'emploi qui ne nécessitent pas que les consommateurs définissent toutes les caractéristiques du SLA :

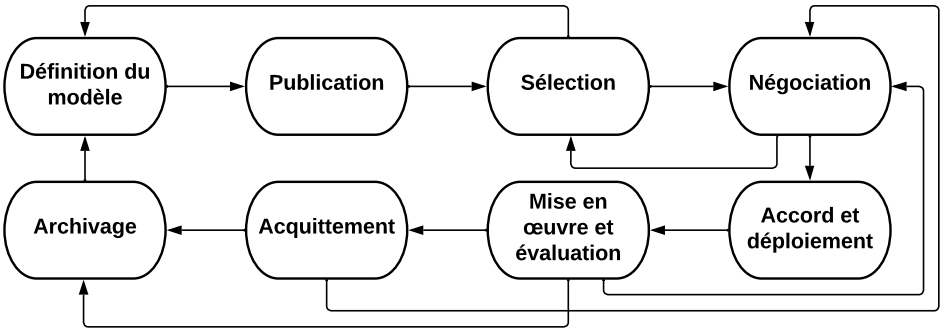


FIGURE 2.5 – Cycle de vie en 8 étapes proposé dans [55]

1. **Définition du modèle de SLA** : Le modèle de SLA est élaboré par le fournisseur de services en fonction des capacités et des exigences de qualité du service. Ce modèle sera utilisé dans les prochaines étapes la contractualisation avec un consommateur de service.
2. **Publication du SLA** : Après l'élaboration du modèle de SLA, celui-ci est publié dans des répertoires intra ou inter-organisationnels afin d'être découvert par des consommateurs de services potentiels.
3. **Sélection du SLA** : Les consommateurs de services soumettent une demande représentée par un modèle de SLA à un courtier afin de trouver les modèles de SLA des services qui répondent à leurs exigences. En conséquence de cette étape, les exigences du consommateur peuvent changer, s'il est trop

contraignant, pour pouvoir être respecté dans une situation de performance réaliste.

4. **Négociation** : Le fournisseur de services avec le SLA sélectionné négocie avec le consommateur de services en se basant sur le contenu de son modèle de SLA. Ce modèle de SLA peut changer au cours de la négociation pour refléter les compromis faits par les deux parties.
5. **Accord et déploiement** : Comme la négociation de service n'aboutit pas toujours, cette étape est séparée de la précédente. Si un accord est atteint alors un SLA spécifique est produit et signé par les deux parties correspondantes. La cohérence et la faisabilité des caractéristiques du SLA sont d'abord validées puis déployé. L'étape de déploiement du SLA consiste en la préparation à la mise en œuvre du service et se déroule en deux étapes : (*i*) les informations essentielles du SLA sont extraites et transmises aux parties prenantes concernées, par exemple, SLO à surveiller par l'auditeur ; (*ii*) les parties prenantes se préparent à la fourniture de service, par exemple, l'auditeur configure ses outils de monitoring avec les informations transmises.
6. **Mise en œuvre et évaluation** : Lors de cette étape du cycle de vie le SLA est mis en oeuvre, c'est-à-dire que le service couvert par le SLA est mis à disposition du consommateur. Puis, est évalué, cela signifie que le service est périodiquement surveillé pour s'assurer que les SLOs convenus dans le SLA sont respectés. La surveillance est effectuée par des composants de mesure des systèmes (monitoring, éventuellement assuré par un auditeur), qui maintiennent des informations sur l'état actuel du système. Ces composants mesurent les différentes métriques définies dans les SLOs soit de l'"intérieur", en récupérant directement les métriques des ressources gérées, soit de l'"extérieur" du domaine du fournisseur de services, par exemple, en sondant ou en interceptant les requêtes des clients. L'évaluation effectuée est ensuite retournée aux différentes parties prenantes. Si la violation ou le nombre de violations est très critique, le SLA est renégocié ou annulé conformément à ce qui a été convenu dans le SLA. Sinon, une action corrective, comme éventuellement convenu dans le contrat, appropriée est sélectionnée et exécutée par le système de gestion du fournisseur de service (par exemple, plus de ressources sont fournies au service sous-performant) en fonction du contexte actuel et des objectifs et politiques commerciaux du fournisseur de service.
7. **Acquittement** : Lors de cette étape du cycle de vie l'on détermine, en fonction du résultat de l'étape d'évaluation précédente, le coût final à payer par le consommateur de services et les éventuelles pénalités appliquées au fournisseur de services pour violation des termes du SLA. Des négociations pour la résiliation du SLA peuvent être menées entre les parties ou pour une réexécution du service dans un niveau de service et un coût différents.
8. **Archivage** : Après l'acquittement du SLA, celui-ci peut être détruit ou ar-

chivé conformément aux politiques des parties signataires. Cependant, même si l'on décide de ne pas archiver le SLA, il existe généralement une période légale (connue sous le nom de "période de limitation") pendant laquelle le SLA doit être conservé, car il s'agit d'un document juridique décrivant comment les services ont été fournis. Si cette étape est accompagnée d'un mécanisme d'audit de trace, elle peut être utilisée pour identifier les problèmes et les tendances de comportement incorrect du service ou les tendances des exigences des utilisateurs, afin d'améliorer le contenu des modèles de SLA développés à l'avenir et de faire évoluer l'implémentation du service pour corriger les problèmes identifiés et répondre aux besoins clients.

Nous avons constaté que le premier cycle de vie en trois étapes, proposé par Ron et al. dans [85], est insuffisant en raison de son manque de précision et de granularité dans la représentation des étapes. Le second cycle en six étapes, proposé par le groupe Sun Micro Systems Inernet Data Center dans [83], permet de représenter plus finement les étapes du cycle de vie du SLA, mais ne considère pas toutes les interactions entre les parties prenantes, notamment les étapes de négociation. En revanche, le troisième cycle de vie en huit étapes, proposé par Kritikos et al. dans [55], offre une représentation détaillée et complète du cycle de vie, incluant les différentes interactions entre les parties prenantes. Pour la suite de cette thèse, nous adopterons la terminologie proposée dans ce cycle de vie pour référer aux différentes étapes et pour démontrer comment nos propositions s'intègrent dans ce cadre, car il est également applicable au contexte multi-cloud.

Nous nous intéressons particulièrement dans cette thèse aux étapes de définition de SLA et en particulier à la modélisation des SLAs lors d'une telle définition, à la validation des SLAs modélisée et à la vérification de leur mise en œuvre. Dans la prochaine section, nous examinons les langages de spécification des SLAs en détaillant les éléments constitutifs de chacun de ces langages.

2.2.1.2 Langages de spécification de SLA

Dans cette section, nous étudions les différents langages permettant la définition de SLA. Nous commençons par le premier langage de spécification de SLA faisant référence dans la littérature : WSLA (*angl. Web Services Level Agreement*) [51], proposé en 2003 par IBM pour la spécification des accords de niveau de service pour les services Web. En se basant sur le modèle de WSLA, plusieurs propositions de langage de spécification de SLA ont été faites pour le contexte du cloud, telles que SLA* [50], rSLA [62] et ySLA [32]. On retrouve également des langages de spécification de SLA cloud supportant la définition de comportement dynamique dans le SLA, telles que CSLA [52] et SLAC [90].

Dans la suite cette section, nous étudions en détail les composants et les concepts couverts par chacun des langages de spécification cités.

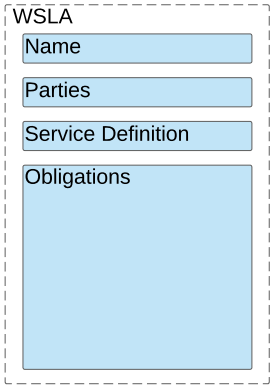


FIGURE 2.6 – Diagramme des composants de WSLA

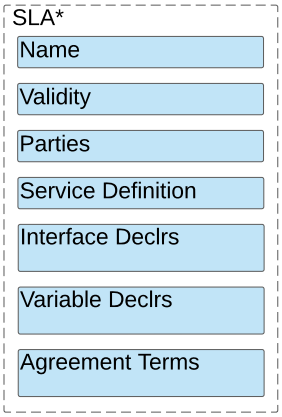


FIGURE 2.7 – Diagramme des composants de SLA*

WSLA [51] (Web Services Level Agreement) fournit un cadre pour spécifier et surveiller les SLAs pour les services Web, incluant les métriques de performances, les obligations de service et les actions en cas de non-respect des accords. Ce langage couvre les concepts de définition de parties prenantes au contrat ($Parties$), définition du service couvert ($ServiceDefinition$) et de la définition des SLOs ($Obligations$).

SLA* [50] issue du projet SLA@SOI dont l'objectif est de fournir une généralisation des concepts proposés dans les standards spécifiques aux services web : *WS-Agreement* [10], *WSLA* [51] et WSDL [18]. Les concepts couverts par ce langage sont la période de validité ($Validity$), les parties prenantes au contrat ($Parties$), la déclaration d'interfaces ($InterfaceDclrs$) et de variables ($VariableDeclrs$), la définition de service couvert ($ServiceDefinition$) et de SLOs ($AgreementTerms$).

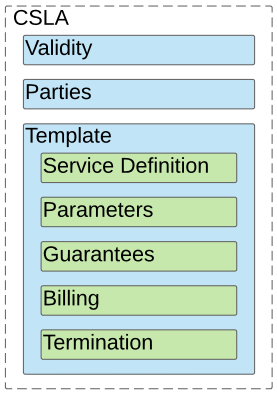


FIGURE 2.8 – Diagramme des composants de CSLA

FIGURE 2.9 – Diagramme des composants de SLAC

CSLA [52] proposé par Kouki et al. est un langage supportant les concepts clés du SLA des langages précédent avec en plus la possibilité d'intégrer une notion de dynamicité des SLOs. Cette dynamicité se base sur les concepts de la logique floue (*"fuzziness"*) et de confiance (*"confidence"*) en les fournisseurs. En

d'autres termes, plus les fournisseurs sont fiables, plus les SLOs sont souples. Ces concepts permettent en plus des anciens modèles de définir des marges acceptables de conformité aux objectifs plus flexibles. Les concepts couverts par ce langage sont la période de validité ($Validity$), les parties prenantes au contrat ($Parties$), la définition de modèle ($template$), la définition de service couvert ($ServiceDefinition$, $Parameters$), les SLOs ($Guarantees$), modèle de prix ($Billing$) et la procédure de finalisation ($Termination$).

SLAC [90] présenté par Uriarte et al. est un langage de SLA où les parties prenantes peuvent définir à l'avance plusieurs niveaux de service et l'adaptation des services fournis en fonction de ces définitions sous condition de validation des parties prenantes. Les concepts couverts par ce langage sont la période de validité ($Validity$), les parties prenantes au contrat ($Parties$), la définition de service couvert ($Terms$), les SLOs ($Guarantees$), la définition de comportement dynamique des services ($Dynamism$) et le modèle de prix ($Billing$).

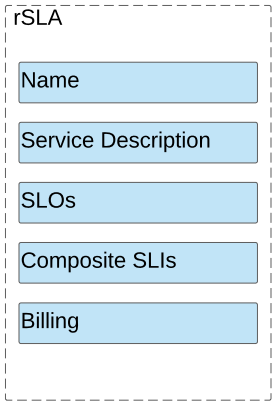


FIGURE 2.10 – Diagramme des composants de rSLA

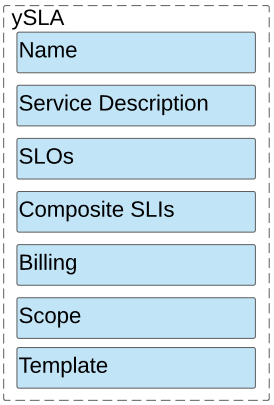


FIGURE 2.11 – Diagramme des composants de ySLA

rSLA [62] présenté par Ludwig et al. est basé sur des objets ruby [8] supportant les concepts courants de définition des SLOs. L'objectif principal de cette approche est de simplifier la définition des SLAs, de se rapprocher des outils contemporains de scripting tel que chef [9] utilisés par les administrateurs qui définissent les SLAs. Ils proposent également une méthode flexible de monitoring de SLOs réutilisable. Les concepts couverts par ce langage sont la définition du service couvert ($ServiceDescription$), les SLOs ($SLOs$), les SLIs composite ($CompositeSLIs$) et le modèle de prix ($Billing$).

ySLA [32] présenté par Engel et al. est une extension des concepts proposés dans rSLA en YAML [10] intégrant en plus les concepts de "Templates" (ou modèle) et de "Scopes" (ou champs d'application des SLOs). Les "Templates" permettent de définir des modèles réutilisables pour différents services. Les concepts couverts par ce

8. `https://www.ruby-lang.org/`
9. `www.chef.io`
10. `https://yaml.org/`

langage sont la définition du service couvert (*ServiceDescription*), les SLOs (*SLOs*), les SLIs composite (*CompositeSLIs*), le modèle de prix (*Billing*), la définition de scopes (*Scope*) et la définition de modèles (*Template*).

Dans cette section, nous avons examiné les langages de spécification de SLA suivant : WSLA, SLA*, CSLA, SLAC, rSLA et ySLA. Nous résumons dans le Tableau 2.2 le comparatif de ces langages de spécification de SLA. WSLA fournit un cadre pour spécifier et surveiller les SLAs pour les services web, incluant des métriques de performance, des obligations de service et des actions en cas de violations de l'accord. Ses points forts résident dans son approche exhaustive de la définition des niveaux de service et de la surveillance de la conformité. Cependant, il est principalement orienté vers les services web et manque de support pour les comportements dynamiques. SLA* vise à généraliser les concepts des standards de services web tels que WS-Agreement et WSLA. Bien qu'une large gamme d'éléments de SLA soit supporté, les aspects dynamiques des environnements cloud ne sont pas abordé adéquatement.

CSLA introduit le comportement dynamique au niveau des SLOs dans les SLAs en incorporant la logique floue et les niveaux de confiance permettent des SLOs plus flexibles qui peuvent s'adapter aux conditions variables en fonction de la fiabilité des fournisseurs de services. Toutefois, la dynamicité des services fournis n'est pas supportée. SLAC se concentre sur l'intégration du comportement dynamique directement dans le cadre du SLA. Il permet aux parties prenantes de définir plusieurs niveaux de service à l'avance, offrant une approche structurée pour gérer la variabilité des services. Cependant, son support partiel pour le comportement dynamique des services limite son applicabilité aux environnements cloud hautement dynamiques.

rSLA simplifie la définition des SLAs en utilisant des objets Ruby, visant à aligner la gestion des SLAs avec des outils de script contemporains comme Chef. Ce langage est conçu pour être flexible et réutilisable pour surveiller les SLOs, ce qui le rend adapté aux administrateurs préférant une approche scriptée. Néanmoins, rSLA ne supporte pas les aspects dynamiques nécessaires pour les systèmes multi-cloud complexes. ySLA étend les concepts de rSLA en intégrant des modèles YAML et des périmètres, permettant des modèles réutilisables pour divers services. Bien qu'il améliore la flexibilité et la réutilisabilité, les limitations de ySLA en termes de comportement dynamique et de support multi-cloud le rendent moins adapté aux environnements nécessitant une grande adaptabilité.

Bien que ces différents langages de spécification permettent de formaliser des SLAs, des SLAs cloud et des SLAs cloud dynamique, aucun ne permet la description de SLA multi-cloud dynamique, comme visé par notre objectif **(O1)**, incluant une décomposition des SLAs pour chaque composant d'un système multi-cloud et la dynamicité de chacun de ces composants. Nous verrons donc dans la prochaine section les propositions existantes pour les langages de spécification de SLA multi-cloud.

2.2.1.3 Langages de spécification de SLA multi-cloud

Comme mentionné dans la première partie de ce chapitre, les systèmes multi-cloud sont par nature distribués et dynamique. Il est alors nécessaire, pour permettre la description de SLA multi-cloud, de supporter ces spécificités. Les travaux adressant la problématique de modélisation de SLAs multi-cloud sont, à ce jour, peu nombreux. Nous présentons dans cette section les différentes approches que nous avons identifiées permettant la modélisation de SLA multi-cloud relatif à notre objectif **(O1)**.

Tout d'abord, nous pouvons retrouver dans les travaux portant sur la composition de services dans le contexte multi-cloud, de premières approches de représentation de SLA multi-cloud. En effet, ces travaux proposent des approches de sélection automatiques de services appropriés à partir d'un ensemble de services basé sur les SLOs. Farokhi et al. [35, 36], Jrad et al. [49] et Taha et al. [87] proposent des approches de sélection de services basée sur les SLAs pour les environnements multi-cloud. Ces approches considèrent les SLOs lié à la sécurité dans leur processus de sélection. D'après différentes revues de littérature sur la composition de service cloud basé sur les SLAs, tel que Lahmar et al. [58] et Fard et al. [34], la majorité de ces travaux ne considèrent qu'une version simplifiée de système multi-cloud représentant les services sélectionnés et leur SLOs associé. En outre, ces travaux supposent que les SLAs sont homogènes et utilisent des modèles simplifiés de SLA. Ces travaux se concentrent principalement sur les étapes préliminaires à la mise en œuvre des SLAs, c'est-à-dire la sélection et la négociation. Également, ils n'intègrent pas la possibilité de modéliser l'intégralité des éléments constitutif d'un SLA multi-cloud ni la dynamicité qui peut être associée aux SLAs.

Cette notion de SLA multi-cloud est également prise en compte dans certains projets open source et européens qui visent à orchestrer des ressources sur plusieurs fournisseurs de services cloud. Le projet "MODAClouds" [11], par exemple, propose une approche basée sur les modèles pour la conception et l'exécution d'applications sur plusieurs clouds [11]. Pour la gestion des SLAs, Ardagna et al. évoquent, dans [11], un modèle de SLA à deux niveaux : un premier niveau qui décrit l'accord entre les clients et les fournisseurs de cloud et un deuxième niveau qui décrit la qualité de service attendue du fournisseur de cloud. Cette approche de SLA à deux niveaux est également utilisée dans le projet "SeaClouds" [16]. Toutefois, ces deux projets s'appuient sur *WS-Agreement* [10], lequel est fondé sur *WSLA* [51] pour la définition de leurs SLA. Bien que cette méthode permette de représenter deux niveaux de SLA, à savoir *(i)* le niveau de l'application exécutée dans les composants d'un système multi-cloud, et *(ii)* le niveau des composants du système multi-cloud, elle demeure limitée par le langage de spécification de SLA initialement conçu pour les services Web, comme discuté dans la section précédente.

Dans le projet DECIDE [21], les auteurs s'intéressent au concept de SLA

11. `https://cordis.europa.eu/project/id/318484`

multi-cloud, l'objectif principal du projet est de proposer un framework permettant la conception, le développement et le déploiement des services multi-cloud. Dans ce cadre, les auteurs proposent un langage de spécification de SLA multi-cloud (*MCSLA*) basé sur un standard de spécification de SLA cloud (ISO-19086 [44]) pour la spécification de leur SLA multi-cloud. Les principaux avantages de ce langage résident dans sa capacité à représenter plusieurs niveaux de SLA : un global à l'application présenté à l'utilisateur final et un pour chacun des composants du système multi-cloud. Toutefois, aucune de ces différentes propositions ne permettent la description du comportement dynamique des services cloud dans un contexte multi-cloud.

2.2.1.4 Discussion

Dans cette section, nous avons étudié les différents modèles de cycles de vie de SLA (Section 2.2.1.1), les langages existant pour la spécification de SLA cloud (Section 2.2.1.2) et les langages de spécification de SLA multi-cloud (Section 2.2.1.3). Nous comparons, dans le Tableau 2.2, huit langages que nous considérons les plus pertinents avec notre objectif de modélisation des SLA multi-cloud dynamique.

Tableau 2.2 – Tableau récapitulatif des langages de spécification de SLA

(a) Partie 1

Langages de spécification	Champs d'application	Concepts Couverts					
		Parties Prenantes	Service Definition	SLO	SQO	SLI Composite	SLA Composite
WSLA [51]	Services Web	Oui	Oui	Oui	-	-	-
SLA* [50]	Cloud	Oui	Oui	Oui	-	-	-
CSLA [52]	Cloud	Oui	Oui	Oui	-	-	-
SLAC [90]	Cloud	Oui	Oui	Oui	-	-	-
rSLA [62]	Cloud	-	Oui	Oui	-	-	-
ySLA [32]	Cloud	-	Oui	Oui	-	Oui	-
ModaClouds [11]	Multi-Cloud	Oui	Oui	Oui	-	-	Oui
MCSLA [21]	Multi-Cloud	-	Oui	Oui	Oui	Oui	Oui

(b) Partie 2

Langages de spécification	Concepts Couverts				
	Période de validité	Facturation	Template	Dynamicité	Finalisation
WSLA [51]	-	-	-	-	-
SLA* [50]	Oui	-	-	-	-
CSLA [52]	Oui	Oui	Oui	SLOs	Oui
SLAC [90]	-	Oui	-	Services	-
rSLA [62]	-	Oui	Oui	-	-
ySLA [32]	-	Oui	Oui	-	-
ModaClouds [11]	-	-	-	-	-
MCSLA [21]	Oui	-	-	-	-

Nous présentons 2 critères de comparaisons pour chacun des langages : leurs champs d'application (Service Web, Cloud ou Multi-Cloud) et les concepts couverts par ces langages. Nous avons identifiés 11 concepts comme étant les plus important pour modéliser un SLA multi-cloud dynamique. Ces concepts sont : **Parties prenantes** : Identifier les parties prenantes du SLA (acteurs du cloud voir Section 2.1.1.4) ; **Service definition** : Décrire les services couverts par le SLA ; **SLO** : Définir un objectif de niveau de service quantitatif ; **SQO** : Définir un objectif de niveau de service qualitatif ; **SLI composite** : Définir des SLOs basés sur des indicateurs de niveaux de service ; **SLA composite** : Créer un SLA composé de plusieurs SLA ; **Période de validité** : Spécifier une période de validité pour le SLA ; **Facturation** : Définir les modèles de facturation des services couverts ; **Template** : Créer des templates de SLA réutilisables pour plusieurs services ; **Dynamicité** : Supporter la dynamicité dans les SLAs ; **Finalisation** : Décrire la procédure de fin (Termination) du SLA.

Nous pouvons voir que WSLA [51] avec comme champs d'application les services Web, couvre les concepts des parties prenantes, définition de service et SLO. SLA*[50] avec comme champs d'application le cloud, couvre les concepts de parties prenantes, définition de service, SLO et période de validité.

CSLA [52] et SLAC [90] avec pour champs d'application le cloud couvrent les concepts de parties prenantes, de définition de service, de SLO et de facturation. CSLA couvre en plus les concepts de la période de validité, le template, la finalisation et supporte également la dynamicité des SLOs. SLAC supporte en plus une dynamicité des services cloud.

rSLA[62] et ySLA [32] avec pour champs d'application le cloud couvrent les concepts de définition de service, SLO, facturation et de template. Ysla couvre en plus le concept de SLI composite.

ModaClouds [11] et MCSLA [21], avec pour champs d'application le multi-cloud, couvrent quant à eux les concepts de définition de services, SLO et SLA composite. ModaClouds couvre spécifiquement le concept de parties prenantes. MCSLA couvre spécifiquement les concepts de SQO, SLI composite et de période de validité.

Nous avons identifié un certain nombre de travaux portant sur les SLAs dans le contexte cloud, cependant, le contexte du multi-cloud reste relativement peu couvert. On peut voir dans ces différents travaux qu'aucun ne répond complètement à notre objectif d'avoir un champs d'application multi-cloud, de couvrir l'ensemble des concepts nécessaires à la représentation de SLA et un support de la dynamicité des services. Toutefois, celui qui s'en rapproche le plus est le langage de spécification MCSLA proposé dans le cadre du projet Decide H2020 [21].

D'après nos connaissances à ce jour, il n'existe pas de langage permettant la définition de SLA multi-cloud dynamique qui répond à notre objectif **(O1)** de modélisation de SLA multi-cloud dynamique. Le principal manque que nous identifions est dans la définition de la dynamicité. En effet, comme nous avons pu le voir dans la section 2.2.1.2, la majorité des propositions faites pour les langages de spécification de SLA ne supportent pas l'aspect dynamique dans le SLA, à l'excep-

tion de CSLA [52] et SLAC [90]. Dans la section suivante, nous nous intéresserons à la mise en oeuvre des stratégies de reconfiguration dans le contexte multi-cloud.

2.2.2 Mise en oeuvre de la dynamicité dans les SLA multi-cloud

Dans le cadre de l'objectif **(O1)** de cette thèse, nous avons mis en évidence l'importance de modéliser des SLAs multi-cloud dynamiques, intégrant des mécanismes d'ajustement des services en fonction des fluctuations de la charge utilisateur. Comme établi dans la section précédente, il existe une lacune en termes de langage de spécification pour les SLAs dynamiques multi-cloud répondant à cette exigence [46].

Par conséquent, dans cette section nous présentons notre étude de la modélisation des stratégies de reconfiguration dans un contexte multi-cloud, en tenant compte de la dynamicité inhérente à ces environnements, afin d'enrichir les modèles de SLAs. Toutefois, comme nous avons pu le voir dans la section précédente, il n'existe que deux langages de spécification qui couvrent partiellement les concepts nécessaires à la définition d'un SLA multi-cloud dynamique. C'est pourquoi, nous nous intéresserons dans cette section aux mécanismes permettant la mise en œuvre de la dynamicité dans un contexte multi-cloud.

2.2.2.1 Élasticité dans le contexte cloud

L'élasticité est de nos jours considérée comme l'une des caractéristiques essentielles du cloud. Dans cette section, nous commençons par définir la notion d'élasticité, puis décrivons ses principales caractéristiques associées. L'élasticité est définie comme "la capacité d'un système à reconfigurer dynamiquement ses ressources pour s'adapter aux exigences et contraintes d'une charge de travail variable" [42]. Habituellement, cela est réalisé par l'exécution d'actions de reconfiguration qui se déroulent suite à des événements, permettant de configurer automatiquement ou de reconfigurer des ressources cloud. Pour permettre une compréhension précise de l'élasticité, nous commençons par clarifier les caractéristiques communes des identifiés dans la revue de littérature. D'après les différentes revues de la littérature [12, 23, 74] sur l'élasticité des ressources dans le contexte cloud, nous avons identifié les caractéristiques suivantes : Portée, Objectif, Action, Mode et Fournisseur.

Portée : spécifie la cible de l'élasticité, qui peut être l'infrastructure (machine virtuelle, serveur, stockage, réseau) ou la plateforme (conteneur, base de données, environnement d'exécution).

Objectif : représente les multiples objectifs de support de l'élasticité qui sont la réduction des coûts opérationnels, le maintien des performances souhaitées, l'assurance de la disponibilité et la réduction de l'empreinte énergétique.

Actions d'élasticité : se réfèrent à la manière dont la reconfiguration est effectuée en utilisant l'élasticité des ressources cloud. Nous distinguons quatre

méthodes principales : la mise à l'échelle horizontale, la mise à l'échelle verticale, la migration et la reconfiguration de l'application.

- **Mise à l'échelle Horizontale :** représente la possibilité d'ajouter ou de supprimer des instances (par exemple, machine virtuelle (VM), applications, conteneurs ou base de données).
- **Mise à l'échelle Verticale :** contrairement à la mise à l'échelle horizontale, la mise à l'échelle verticale vise à augmenter ou réduire finement les ressources d'une instance telles que le CPU, la RAM et le stockage plutôt que des instances.
- **Migration :** se réfère au déplacement d'un composant de son fournisseur de service initial vers un autre. La migration peut être appliquée au niveau de la VM ou de l'application. La première est appelée migration de VM, qui permet de transférer une VM en cours d'exécution sur un serveur physique vers un autre. La dernière est référencée comme migration d'application, qui permet de migrer seulement l'un des composants spécifiques de l'application au lieu de la VM entière, comme une base de données, par exemple.
- **Reconfiguration de l'Application (RA) :** consiste à changer des aspects spécifiques de l'application telles que la taille du cache d'une base de données, la politique de récupération de base de données, etc. Ceci est complémentaire à la mise à l'échelle verticale qui ne concerne que les attributs liés à la capacité de l'instance.

Mode : indique la manière d'exécuter les actions d'élasticité. Nous distinguons deux modes, qui sont : Manuel et Automatique. Le mode manuel nécessite l'intervention d'un utilisateur en charge des phases d'observation, de décision et de réaction afin d'effectuer des actions d'élasticité. Le mode automatique signifie que le processus de gestion de l'élasticité est automatisé, c'est-à-dire que le système prend en charge le contrôle de l'élasticité sans aucune intervention externe. La mise en œuvre du mode automatique est généralement basée sur l'une des stratégies suivantes :

- **Réactif :** signifie que les actions d'élasticité sont déclenchées sur la base de règles qui indiquent l'état du système en termes de charge de travail ou d'utilisation des ressources. Si l'une de ces règles est satisfaite, le contrôleur d'élasticité déclenche des actions pour s'adapter aux changements observés en conséquence.
- **Proactif :** le déclenchement des actions d'élasticité est basé soit sur des calendriers prédéfinis, soit sur des prédictions qui sont définies à l'aide de techniques prédictives anticipant les besoins futurs du système.
- **Hybride :** combine les approches réactives et proactives pour exécuter des actions d'élasticité.

Fournisseur : les actions d'élasticité peuvent être appliquées aux ressources de fournisseurs de cloud uniques ou multiples.

L'élasticité dans le contexte multi-cloud se réfère à la capacité de redimensionner

dynamiquement les ressources et les services en fonction des variations de charge, des pannes ou d'autres facteurs, tout en veillant à maintenir les objectifs de niveau de service (SLOs) prédéfinis qui composent un accord de niveau de service (SLA). Cette approche permet aux applications d'ajuster leur capacité en temps réel pour répondre à la demande changeante tout en garantissant une performance optimale et une disponibilité continue. L'élasticité multi-cloud tire parti de la flexibilité offerte par la combinaison de différents fournisseurs cloud, permettant ainsi une adaptation plus fine aux besoins spécifiques du système, tout en minimisant les coûts et en optimisant l'utilisation des ressources.

Dans la prochaine section, nous présentons notre étude détaillée des travaux tirant parti des mécanismes d'élasticité pour le maintien des SLOs.

2.2.2.2 Élasticité pour le maintien de SLO

Différentes revues de la littérature [12, 23, 74] présentent des moyens de mise en œuvre de ces mécanismes d'élasticité spécifiquement pour le maintien d'objectifs de niveau de service. On retrouve notamment des travaux qui permettent d'en tirer parti automatiquement, c'est-à-dire d'adapter les ressources à la charge de travail tel que l'approche présentée par Kouki et al. dans [53] dans le cadre d'une planification de capacité pour la fourniture de service SaaS à des utilisateurs finaux. Les auteurs s'intéressent particulièrement au point de vue d'un fournisseur de service SaaS qui adapte les ressources de ses services pour garantir des SLOs de latence, de coût et de taux de non-réponse aux requêtes. Uriarte et al. proposent dans [90] le langage de spécification *SLAC* qui intègre les mécanismes d'élasticité avec les SLAs cloud. Ce langage donne la possibilité de définir des comportements dynamiques des services en prenant en compte la validation de différentes parties prenantes du SLA dans une logique de courtier de services cloud. Par exemple, différents scénarios pourront être défini où les services fournis sont reconfigurés à la demande du client, en fonction de métrique technique prédéfinie ou en fonction de la capacité d'un fournisseur. On peut voir dans ces travaux une première possibilité d'utiliser des mécanismes d'élasticité au service du maintien de différent SLOs. Kritikos et al. présentent dans [56] une approche permettant de définir des stratégies de reconfiguration appelé règles d'adaptation. La formalisation de ces règles d'adaptation implique l'utilisation de la syntaxe et la sémantique étendues du langage CAMEL [80]. Cette approche permet de définir des règles complexes et conditionnelles pour l'adaptation d'applications dans des environnements multi-cloud. Les règles peuvent spécifier des actions d'adaptation basées sur des conditions tel que les performances et la disponibilité ainsi que d'autres métriques cruciales pour le fonctionnement optimal de l'application.

Ces différents travaux présentent des approches techniques permettant de mettre en place les mécanismes d'élasticité. Toutefois, on retrouve également des travaux qui permettent l'association de SLO à des mécanismes d'élasticité et qui donnent les outils pour formaliser ces associations. Nastic et al. introduisent SLOC dans [75] pour le

projet Arktos [12] qui vise à étendre Kubernetes [13] (Orchestrateur de conteneurs) pour des infrastructures cloud à grande échelle. SLOC est un framework mettant en avant un paradigme axé sur les SLOs et leur lien avec les comportements d'élasticité. L'une des principales contributions de leur approche est dans la considération de 4 types de SLOs associable à des mécanismes d'élasticité : les ressources, le coût, la qualité et le cas d'usage. En effet, l'objectif est de permettre à l'architecte cloud de définir en fonction de ces 4 dimensions des mécanismes d'élasticité. De plus, ils proposent une taxonomie sur le lien entre les métriques de bas niveau et de haut niveau pour faciliter l'association entre les caractéristiques techniques des ressources physiques et les SLOs au niveau fonctionnel. Cependant, cette association est particulièrement importante mais n'est pas triviale à définir et nécessite une étude approfondie ou des connaissances métiers pour la définir.

En parallèle à ces travaux, nous avons également identifié des projets collaboratifs entre le domaine industriel et académique donnant la possibilité d'appliquer dans des architectures réelles les différents concepts présentés précédemment. En effet, ces différentes approches académiques sont très proches du domaine industriel comme on peut le voir dans les extensions de produits largement répandus comme Kubernetes. On retrouve également différents projets open source qui s'intéressent à ces problématiques et permettant de tirer parti des SLOs dans des environnements de production. Au cours des dernières années, une tendance s'est dessinée autour des SLOs dans le domaine de l'ingénierie de la Fiabilité d'infrastructure (SRE) [15]. Dans cette communauté, les SLOs servent surtout aux équipes à suivre l'état de leur propre application mais ne représente pas un engagement contractuel. C'est pourquoi les SLAs sont souvent perçus comme rigides et comme un engagement fort de la responsabilité de l'équipe en charge. Ils jouent un rôle crucial en fournissant aux équipes une perspective claire des composants architecturaux essentiels. Les concepts tels que les SLOs et les SLIs (Indicateurs de Niveau de Service) sont devenus des éléments centraux dans les discussions autour de la fiabilité et de la performance des systèmes.

On retrouve des outils qui permettent de définir des SLOs et de les associer à des outils de monitoring tel que prometheus [14]. Les principaux projets permettant de définir ces méthodes sont : Sloth [15] et Pyrra [16]. Les projets Sloth et Pyrra sont des générateurs de SLOs pour Prometheus conçu pour être simple et permettre de créer des spécifications standardisées des tableaux de bord prêts à l'emploi pour Grafana [17]. Le projet OpenSLO [18] vise quant à lui à établir une spécification ouverte pour la définition des SLOs. L'ambition est de promouvoir une approche commune, indépendante du fournisseur, pour le suivi et l'interaction avec les SLOs. Bartelucci

12. https://github.com/CentaurusInfra/arktos
13. https://kubernetes.io/
14. https://prometheus.io/
15. https://sloth.dev
16. https://github.com/pyrra-dev
17. https://grafana.com/
18. https://github.com/OpenSLO/OpenSLO

et al. présentent dans [12] d'autres projets collaboratifs tirant parti des mécanismes d'élasticité. En effet, dans cette revue de littérature, les auteurs étudient les différents mécanismes d'élasticité pour les prochaines générations d'applications cloud. On retrouve notamment les concepts de base de Kubernetes et des extensions de ces derniers comme KEDA [19] et Karpenter [20] qui permettent de définir des stratégies de reconfiguration au service des SLOs dans Kubernetes.

L'ensemble de ces initiatives témoigne de l'importance croissante accordée à la définition et à la gestion des SLOs dans l'environnement actuel du cloud natif. Toutefois, la recherche sur le lien entre les SLOs et les stratégies de reconfiguration demeure un domaine nécessitant davantage d'explorations.

2.2.2.3 Discussion

Dans cette section, nous avons établi que l'élasticité constitue un mécanisme essentiel dans le contexte cloud. Ce concept permet notamment d'adapter les architectures cloud aux fluctuations de la charge de travail des utilisateurs finaux. On retrouve dans la littérature un certain nombre d'approches permettant de modéliser des stratégies de reconfiguration tirant parti de l'élasticité. Ces stratégies assurent le respect des SLOs (qui composent les SLAs) en exécutant des séquences d'actions d'élasticité, incluant l'ajout ou la suppression de ressources, la modification des types de ressources utilisés et l'ajustement de configuration de services. Ces stratégies sont liées aux SLOs pour en assurer le respect mais la cohérence entre eux n'est pas vérifiée. Par cohérence entre stratégie de reconfiguration et SLOs, cela signifie que les stratégies de reconfiguration ne sont pas la cause des violations de SLOs. Toutefois, nous ne retrouvons pas d'approches répondant à notre objectif **(O1)** de description de SLA multi-cloud dynamique associant des SLAs multi-cloud à des stratégies de reconfiguration. Dans notre travail, nous proposons donc de modéliser un SLA multi-cloud dynamique décomposé en 2 niveaux ; application et composants du système multi-cloud couvert enrichi par des stratégies de reconfiguration permettant le maintien de ces SLOs malgré une charge de travail fluctuante.

2.3 Validation et vérification de la conformité de SLA Dynamique

Dans la question de recherche **(RQ2)** de cette thèse, nous avons souligné l'importance du contrôle de SLA multi-cloud dynamique. Cette question a été décomposée en deux objectifs pour s'intéresser à deux phases de contrôle : dans l'objectif **(O2.1)** la phase pré-mise en œuvre d'un SLA et dans l'objectif **(O2.2)** la phase post-mise en œuvre de SLA. Dans cette section, nous présentons dans un premier temps les

19. `https://keda.sh`
20. `https://karpenter.sh`

travaux existants sur la validation pré-mise en œuvre de SLA, dans un second temps, les travaux existants sur la vérification post-mise en œuvre de SLA.

2.3.1 Validation pré-mise en œuvre de SLA

Comme mentionné dans la description du cycle de vie d'un SLA (Section 2.2.1.1), lors de la modélisation d'un SLA multi-cloud dynamique par un architecte cloud ce dernier définit des SLOs globaux que son système multi-cloud doit respecter. Ce système multi-cloud est dit composite car composé de plusieurs services et ressources distribués sur différents fournisseurs cloud distincts pour fournir un service à ses utilisateurs finaux. La composition de services fait référence à la technique d'assemblage et d'orchestration de plusieurs services cloud distincts pour créer un service composé plus complexe. Lors de la composition du système chaque composant aura également des SLOs propres. L'étape de validation pré-mise en œuvre consiste en la validation de la cohérence entre les différents objectifs globaux au système et locaux aux composants, permettant ainsi de s'assurer qu'il n'y aura pas d'incohérences empêchant la garantie des SLO définis au moment de la mise en œuvre du SLA.

Dans cette section, nous étudierons cette étape de validation dans trois types d'approches différentes : section 2.3.1.1 dans le selection de services, section 2.3.1.2 dans la découverte de services, section 2.3.1.3 dans les langages de spécification et section 2.3.1.4 nous discutons ces différents travaux.

2.3.1.1 Validation dans la sélection de services

Une première méthode proposée dans le cadre des Services Web est proposée par Müller et al. dans [70] porte sur la vérification de cohérence des SLAs défini avec le langage de spécification WSLA [51]. Le problème de validation de cohérence est modélisé comme un problème de satisfaction de contrainte (*angl. constraint satisfaction problem, CSP*) qui utilise un solveur CSP conjointement avec un moteur d'interprétation [22] pour vérifier la cohérence et retourner les termes incohérents. Les SLAs, comprenant des termes fonctionnels et non fonctionnels, des garanties, des conditions de résiliation et autres informations pertinentes, doivent être rédigés rigoureusement pour éviter les contradictions qui pourraient entraîner des violations de SLOs. En représentant les termes des SLAs sous forme de contraintes, le problème est ensuite résolu par un solveur CSP, comme le solveur Choco utilisé dans la preuve de concept décrite par les auteurs. En cas d'incohérence, le moteur d'explication, tel que Palm, identifie et retourne les termes responsables de l'incohérence.

En appliquant la même logique de modélisation de la validation en tant que problème de satisfaction de contraintes, plusieurs travaux de recherche se sont concentrés sur la composition de services en fonction de SLOs prédéfinis. Ces travaux incluent non seulement la composition de services, mais aussi des approches intégrant la vérification de la cohérence de cette composition vis-à-vis des SLOs

prédéfinis. En effet, après l'étape de composition, une étape de validation est effectuée pour s'assurer que les services sélectionnés sont cohérents entre eux.

Elhabbash et al. proposent dans [30] une approche permettant d'automatiser la sélection des services en fonction de SLOs spécifique dans un contexte Multi-Cloud. Cette approche introduit SLO-ML, un langage de modélisation conçu pour capturer les exigences de SLOs simplifié. Grâce à une syntaxe simplifiée, SLO-ML permet aux utilisateurs de spécifier les SLOs pour les différentes composantes d'une application multi-cloud. Ensuite, la sélection des services est effectuée en utilisant une approche basée sur l'évaluation de la pertinence de chaque service à l'aide d'une fonction d'utilité. Cette fonction permet de quantifier dans quelle mesure chaque service satisfait les SLOs requis, en attribuant une valeur d'utilité qui est ensuite maximisée pour sélectionner le service optimal.

Heilig et al. répondent dans [41] au problème de gestion des ressources cloud dans un environnement multi-cloud en s'intéressant particulièrement à la réduction de coût et de temps d'exécution des applications utilisateur consommant des services IaaS. Pour ce faire, les auteurs proposent une approche basée sur un algorithme génétique à clés aléatoires biaisées (*angl. biased random key genetic algorithm*). Cet algorithme utilise une population de solutions potentielles représentées par des vecteurs de clés aléatoires, où chaque clé correspond à une décision dans le processus de sélection des ressources. L'approche vise à optimiser la sélection des ressources cloud en tenant compte des coûts financiers et des performances d'exécution. Les clés aléatoires biaisées permettent d'explorer l'espace de solutions en favorisant les combinaisons de ressources les plus prometteuses.

2.3.1.2 Validation dans la découverte de services

Kritikos et al. proposent dans [54] une méthode de composition de service dans un environnement multi-cloud basé sur un processus de hiérarchie analytique (*angl. Analytical Hierarchy Process, AHP*) permettant de pondérer les objectifs en fonction des priorités des utilisateurs. Cette approche vise à optimiser la conception des applications multi-cloud en prenant en compte divers types de services cloud tout en satisfaisant simultanément les exigences variées des utilisateurs finaux, telles que les besoins en qualité, en déploiement, en sécurité, en placement et en coûts. L'approche se distingue par sa capacité à intégrer et à hiérarchiser différents types de critères de performance et de contraintes utilisateur dans un modèle de déploiement abstrait. En utilisant AHP, cette méthode attribue des poids relatifs à chaque paramètre de qualité et de coût, permettant ainsi une optimisation globale basée sur une fonction de pondération additive simple. Pour résoudre ce problème d'optimisation, les auteurs ont recours à des techniques de résolution de problèmes d'optimisation par satisfaction de contraintes.

Taha et al. proposent dans [87] une approche de composition de services cloud basé sur modèle de classement relatif (*angl. relative ranking model*) pour trouver une composition de services correspondant aux attentes utilisateurs. Cette méthode

répond à la problématique de gestion des dépendances entre services en offrant une solution pour réduire les conflits tout en optimisant la sélection des services selon les exigences spécifiques des utilisateurs. L'originalité de cette approche réside dans l'extension et l'adaptation des techniques de prise de décision utilisées pour le cloud unique aux environnements multi-cloud. Le cadre proposé évalue les SLOs offerts par divers fournisseurs de services cloud et sélectionne la combinaison optimale de services provenant de plusieurs fournisseurs pour satisfaire au mieux les exigences des clients. La méthode comporte deux volets principaux : la construction et la gestion d'un SLA multi-cloud et la sélection des services. Le premier volet implique l'évaluation des niveaux de service fournis par les différents fournisseurs et la création d'un SLA multi-cloud qui inclut toutes les relations de dépendance entre les services. Le second volet utilise une technique de sélection basée sur les scores pour choisir les services en fonction des SLAs et des exigences des clients. Cette technique permet non seulement de détecter automatiquement les conflits résultant des dépendances entre services, mais aussi de fournir des explications sur ces conflits pour aider à les résoudre. L'approche est validée à l'aide de données réelles issues de SLOs pour la sécurité, conforme à la norme ISO/IEC 19086. Ils intègrent dans leur approche à l'issue de l'étape de composition de services une étape de validation permettant de vérifier que la sortie de l'algorithme de composition est cohérente.

2.3.1.3 Validation dans les langages de spécification

Uriarte et al. proposent dans [90] avec leur langage de spécification SLAC un solveur permettant de vérifier la cohérence des SLAs définis avec SLAC. Les objectifs définis dans un SLA sont traduits en contraintes puis vérifié qu'elles sont cohérentes entre elles. Cette approche inclut un solveur permettant de vérifier la cohérence des SLAs définis avec SLAC. Ce solveur traduit les différents objectifs spécifiés dans un SLA en un problème de satisfaction de contraintes (CSP) et vérifie que toutes les contraintes sont cohérentes entre elles. Cette vérification est effectuée à la fois au moment de la conception pour identifier d'éventuelles incohérences et au moment de l'exécution pour s'assurer que les caractéristiques du service respectent les valeurs spécifiées. En cas de non-conformité, des actions correctives peuvent être déclenchées automatiquement.

Pusztai et al. étendent dans [78] le travail initial de SLOC [75] pour proposer une mise en œuvre dans le projet Polaris SLO Cloud [21]. Les deux évolutions majeures par rapport à SLOC sont : (*i*) une fonctionnalité de vérification de type entre les SLOs et les stratégies d'élasticité, et (*ii*) une abstraction permettant de séparer les SLOs des stratégies d'élasticité. La vérification de type assure la compatibilité entre les SLOs et les stratégies d'élasticité, tandis que l'abstraction permet de décorréler les SLOs des stratégies d'élasticité, facilitant ainsi leur réutilisation et augmentant le nombre de combinaisons possibles entre SLOs et stratégies d'élasticité. Le lan-

21. `https://polaris-slo-cloud.github.io/polaris-slo-framework/`

gage SLO Script, proposé dans ce cadre, permet aux fournisseurs de services de définir des SLOs complexes et aux consommateurs de services de les configurer et de les appliquer à leurs charges de travail de manière indépendante de l'orchestrateur. Ce langage inclut des abstractions fortement typées avec des fonctionnalités de sécurité de type, des constructions permettant la séparation des SLOs et des stratégies d'élasticité, une API de métriques fortement typée, et un modèle d'objet indépendant de l'orchestrateur qui promeut l'extensibilité.

2.3.1.4 Discussion

Dans le Tableau 2.3, nous comparons les différentes approches relatives à notre objectif **(O2.1)** de validation pré-mise en œuvre de SLA multi-cloud sur trois critères, leur technique utilisé associé à la validation, leurs champs d'application et les éléments validés.

Tableau 2.3 – Tableau récapitulatif des travaux de validation pré-mise en œuvre de SLA

Approches	Technique utilisé	Champs d'application	Elements validé
Müller et al. [70]	Satisfaction de contrainte	Web-Services	SLOs
Elhabbash et al. [30]	Composition de service	Multi-Cloud	SLOs
Heilig et al. [41]	Composition de service	Multi-Cloud	SLOs
Kritikos et al. [54]	Composition de service	Multi-Cloud	SLOs
Taha et al. [87]	Composition de service	Multi-Cloud	SLOs
Uriarte et al. [90]	Satisfaction de contrainte	Multi-Cloud	SLOs
Pusztai et al. [78]	Validation de type	Cloud-Native	SLO, Règles d'élasticité

Le principal problème identifié dans les approches actuelles [30, 41, 54, 70, 87, 90] est qu'elles se concentrent uniquement sur les SLOs sans prendre en compte les stratégies de reconfiguration. La seule approche identifiée qui considère ces deux éléments est celle de Pusztai et al. [78]. Toutefois, cette approche se limite à l'association des types de SLOs avec les types de règle d'élasticité, par exemple en veillant à ce qu'un SLO sur le CPU soit correctement associé à une règle d'élasticité pour les CPU, sans prendre en compte le contenu détaillé des SLOs et des stratégies. En conclusion, il existe des travaux apparentés à la validation de SLO basés sur la composition de services. Toutefois, on retrouve peu de travaux permettant spécifiquement la validation pré-mise en œuvre de SLA cloud et aucun traitant les SLAs multi-cloud dynamique vérifiant la cohérence pré-mise en œuvre de SLA multi-cloud dynamique. D'autant plus qu'aucun ne permet une validation complète en différents niveaux de SLO composant un SLA multi-cloud et des stratégies de reconfiguration. On peut donc conclure qu'il n'existe pas de travaux répondant complètement à notre objectif **(O2.1)** de validation pré-mise en œuvre de SLA multi-cloud.

Dans ce travail, nous proposons de vérifier préalablement à la mise en œuvre du SLA multi-cloud dynamique, la cohérence des SLOs et des stratégies de reconfigura-

tion associés entre elles pour s'assurer qu'ils sont réalisables et permettre d'anticiper de potentielles violations. Dans cette section, nous avons discuté des travaux relatifs à la validation pré-mise en œuvre de SLA multi-cloud dynamique. Dans la section suivante, nous verrons les méthodes associées au contrôle post-mise en œuvre de SLA multi-cloud.

2.3.2 Vérification post-mise en œuvre de SLA

Dans cette section, nous examinons les différentes méthodes relatives à notre objectif **(O2.2)** pour la vérification post-mise en œuvre de SLA multi-cloud dynamique permettant d'identifier les violations et de les rapporter aux architectes cloud. Comme décrit dans la section 2.2.1.1, de nombreux travaux académiques se sont concentrés sur l'étude du suivi des performances d'un système et la détection de violations de SLOs. Toutefois, peu de travaux portent sur l'étape de remontée des violations aux architectes cloud, étape du cycle de vie du SLA évaluation et d'acquittement, aussi appelé *reporting*. Faniyi et al. présentent dans leur revue de la littérature sur les SLAs [33], une répartition des publications comme suit : négociation et établissement de SLA (22%), suivi et déploiement de SLA (73%), gestion des violations de SLA (3%) et reporting de SLA (1%). La faible attention accordée aux deux dernières phases peut être attribuée à la nature exploratoire de la plupart des efforts de recherche. Par recherche exploratoire, nous entendons les techniques de validation de la recherche qui ne prennent pas en compte les cas d'utilisation réels, où les utilisateurs se préoccupent réellement des rapports/audits de SLA. Cependant, l'état des pratiques industrielles est en avance dans la phase de reporting du SLA. Le service Amazon CloudWatch [22] est un exemple de service qui fournit une surveillance et des notifications automatisées. Peu d'attention a été accordée au reporting des SLAs, bien que ce soit une étape essentielle à considérer du cycle de vie du SLA pour pouvoir l'implémenter dans un cas d'usage réel. Comme précisé avant, cette étape consiste à vérifier la bonne mise en œuvre du SLA et à remonter les potentielles violations.

On retrouve des approches qui mettent en place des chaînes de blocs (*angl. Blockchain*) pour le suivi et plus spécifiquement le reporting des SLAs cloud [9, 89, 95, 97]. Les chaînes de blocs sont des registres distribués et sécurisés qui enregistrent les transactions de manière immutable. Elles permettent de garantir la transparence et l'intégrité des données partagées entre les différentes parties prenantes. Uriarte et al. présentent dans [91] un cadre de gestion des SLAs basé sur SLAC [90]. Le cadre proposé repose sur une architecture de blockchain à deux niveaux. Au premier niveau, le SLA est transformé en un contrat intelligent (*angl. smart contract*) qui guide dynamiquement le provisionnement des services. Au deuxième niveau, une blockchain privée (sur laquelle seul un groupe d'utilisateurs autorisés peut participer) est construite à travers une fédération d'entités de surveillance pour générer

22. `http://aws.amazon.com/cloudwatch/`

des mesures objectives pour l'évaluation des SLAs.

Zhou et al. proposent dans [97] une approche impliquant plusieurs auditeurs cloud utilisant des smart contract pour la vérification de SLO et établir un consensus entre les auditeurs afin de détecter les éventuelles violations. Dans le cadre de la blockchain, les participants à une tâche peuvent être récompensés. Dans ces travaux, le rôle d'auditeur cloud est introduit et récompensé pour la génération de rapports de violation de SLO (identification à partir de son infrastructure propre). La fonction de récompense est soigneusement conçue pour inciter les auditeurs à dire la vérité et pour dissuader les comportements malhonnêtes, tels que la génération de faux rapports pour un gain financier ou la tentative de nuire à des concurrents. Ce fait que l'auditeur doit être honnête est analysé et prouvé en utilisant le principe d'équilibre de Nash de la théorie des jeux. Pour s'assurer que les auditeurs choisis sont aléatoires et indépendants, un algorithme de sélection impartial est proposé pour éviter d'éventuelles collusions.

Cependant, ces approches basées sur les chaînes de blocs pour la gestion des SLAs cloud présentent deux principaux inconvénients. Premièrement, la consommation énergétique associée aux chaînes de blocs, surtout celles utilisant des mécanismes de consensus intensifs comme la preuve de travail, peut être très élevée, posant des défis environnementaux significatifs. Deuxièmement, l'adoption généralisée de solutions basées sur les chaînes de blocs par tous les fournisseurs de services cloud reste un défi. La nécessité pour chaque fournisseur d'intégrer et de maintenir une telle technologie peut représenter une barrière considérable, retardant son adoption à grande échelle et limitant ainsi les bénéfices potentiels de ces solutions.

Une première proposition utilisant des techniques de fouille de processus (*angl. process mining*) pour le reporting des SLAs a été faite dans le contexte de la gestion des services ITIL [23] par Mager dans [64]. Cette approche propose la détection des goulots d'étranglement dans un processus par une analyse détaillée de la perspective temporelle. Cependant, dans cette dernière approche, seules les perspectives de flux de contrôle et de temps sont prises en compte. Kritikos et al. proposent dans [57] un framework basé sur traitement des événements complexes (*angl. complex event processing*) pour l'évaluation de SLO et le déclenchement de mécanisme d'adaptation dans le contexte des applications cloud. Ce framework permet la définition de motif (*angl. pattern*) d'événement et de les associer à des mécanismes d'adaptation. Müller et al. presentent dans [72] une solution pour permettre la compréhension des violations de SLAs étendant leurs travaux dans [70]. Cette solution nommée SALMonADA prend en entrée le SLA sur lequel ce sont entendus le consommateur et le fournisseur et le traduit comme un problème de satisfaction de contraintes. Puis, agis comme un auditeur cloud en monitorant les objectifs et en fournissant des explications en cas de violation. Müller et al. proposent également dans [71] une méthode automatisée de vérification des SLAs qui ont une compensation en cas de violation.

Afin d'assurer le cycle de vie du SLA, une vérification périodique est nécessaire.

23. `https://fr.wikipedia.org/wiki/Information_Technology_Infrastructure_Library`

Comme nous avons pu le voir aucun des travaux existant n'étudient complètement l'étape de vérification de SLA multi-cloud couvrant l'objectif (**O2.2**) de vérification post-mise en œuvre des SLAs multi-cloud dynamique. En effet, aucun ne permet également le suivi des stratégies de reconfiguration. Dans ce travail nous proposons une vérification de la bonne mise en œuvre des SLA multi-cloud dynamique en s'assurant du bon déroulement des stratégies de reconfiguration et du respect de leurs SLOs.

2.3.3 Discussion

Dans cette section, nous avons étudié le contrôle de la conformité de SLA multi-cloud dynamique. Tout d'abord, nous avons décrit les travaux liés à la validation préalable à la mise en œuvre de SLA multi-cloud dynamique (section 2.3.1). Bien que certaines méthodes puissent être associées à de la validation pré-mise en œuvre de SLA multi-cloud dynamique, aucune ne permet de répondre entièrement au besoin de validation pré-mise en oeuvre défini dans notre objectif **(O2.1)** entre les SLOs du SLA multi-cloud et des stratégies de reconfiguration permettant la définition de la dynamicité. Ensuite, nous nous sommes intéressés aux travaux portant sur la vérification post-mise en œuvre de SLA multi-cloud dynamique (section 2.3.2). Nous avons pu voir que les approches traditionnelles permettent le monitoring des SLOs mais aucuns ne permettent également le suivi des stratégies de reconfiguration défini dans leur globalité définie dans notre objectif **(O2.2)**.

2.4 Conclusion

Dans ce chapitre d'état de l'art, nous avons exploré les solutions pertinentes pour notre travail couvrant : (*i*) la modélisation de SLA multi-cloud dynamique, (*ii*) la validation pré-mise en œuvre de SLA multi-cloud dynamique et (*iii*) la vérification post-mise en œuvre de SLA multi-cloud dynamique. Pour chacun, nous avons brièvement présenté les approches connexes et fourni une analyse approfondie pour positionner notre travail par rapport à ces approches.

Pour la première étape, nous avons établi que malgré l'existence de modèles de description de SLA cloud, aucun ne permet la modélisation de SLA multi-cloud tout en prenant en compte le comportement dynamique des services. Dans la seconde étape, nous avons identifié que les techniques existantes de vérification pré-mise en œuvre ne permettent pas de vérifier la cohérence entre les différents SLOs composant un SLA multi-cloud, ni avec les stratégies d'élasticité associés. Pour la troisième étape, nous avons constaté qu'aucun des travaux existants n'étudie complètement la vérification de SLA multi-cloud dynamique, couvrant à la fois les SLOs et la bonne exécution des stratégies d'élasticité associés et ne fournit un rapport aux architectes cloud.

Cet état de l'art nous a permis de confirmer les besoins découlant de nos deux questions de recherche (**RQ1**) sur la modélisation des SLAs multi-cloud associés à

des stratégies de reconfiguration, et (**RQ2**) sur la vérification de la mise en œuvre des SLAs multi-cloud. Ces deux verrous seront donc traités dans les prochains chapitres de cette thèse. Dans le chapitre 3, nous présentons notre modèle de description de SLA multi-cloud dynamique. Dans le chapitre 4, nous introduisons notre approche permettant la validation de cohérence pré-mise en œuvre. Dans le chapitre 5, nous présentons notre approche de vérification post-mise en œuvre de SLA multi-cloud dynamique basée sur la fouille de processus.

Chapitre 3

Description de SLA multi-cloud dynamique

Sommaire

Dans le chapitre précédent, nous avons présenté l'état de l'art relatif aux accords de niveau de service (SLA), à leur utilisation dans le contexte multi-cloud et à la définition de stratégies de reconfiguration. Cela nous a permis de mettre en évidence certaines limites, notamment liées aux modèles de description des SLAs multi-cloud et aux dépendances avec les stratégies de reconfiguration permettant le maintien de SLO.

Dans ce chapitre, nous cherchons à adresser notre objectif **O1** en proposant un modèle de description de SLA multi-cloud dynamique permettant d'adresser les limitations identifiées dans l'état de l'art. Nous prenons le point de vue d'un architecte de système multi-cloud où la problématique de la description de la qualité de service pour un système multi-cloud peut être décomposée en deux sous-problématiques :

(*i*) Comment décrire le niveau de qualité de service d'un système multi-cloud et de ses composants ?

(*ii*) Comment représenter la dynamicité nécessaire au maintien de la qualité de service d'un système multi-cloud ?

Pour adresser la sous-problématique (i), nous proposons un modèle de description de SLA multi-cloud. Ce modèle est composé d'un *Global-SLA*, représentant les SLOs attendus pour un système multi-cloud dans sa globalité et de plusieurs *Sub-SLA* représentant chacun les SLOs des composants de ce système. Pour adresser la sous-problématique (ii) de la dynamicité des services cloud, nous étendons le modèle en décrivant les stratégies de reconfiguration des différents composants d'un système multi-cloud. Ces stratégies de reconfiguration sont un ensemble de règles pour l'adaptation du système et de ses composants permettant d'assurer le respect des SLOs. Nous proposons de décrire ces stratégies sous la forme de machine à états. Ces machines à états sont ensuite associées aux Sub-SLAs liant ainsi les stratégies de reconfiguration aux SLOs de chaque composant d'un système multi-cloud.

Dans la suite de ce chapitre, nous commençons par décrire le modèle de description de SLA multi-cloud dans la section 3.1. Ensuite, nous présentons la définition des stratégies de reconfiguration sous la forme de machine à états et notre modèle de description de SLA dynamique dans la section 3.2. Pour finir, nous présentons dans la section 3.3 la mise en œuvre de notre interpréteur de SLA multi-cloud et l'évaluation de notre modèle de description par rapport aux autres modèles de la littérature.

3.1 Modèle de description de SLA multi-cloud

Dans cette section, nous commençons par présenter les concepts fondamentaux des SLAs dans la section 3.1.1, puis notre proposition de modèle de description de SLA multi-cloud dans la section 3.1.2.

3.1.1 Structure de SLA

Les SLAs définissent la qualité de service requise fournie par un fournisseur de service. Dans un contexte cloud, un SLA est principalement composé de quatre éléments [63] : (a) les *acteurs* du SLA, (b) les termes, ou caractéristiques, des services fournis, (c) les SLOs à maintenir et (d) les pénalités. Ces différentes parties sont récapitulées dans la Figure 3.1. Dans la suite de cette section, nous présentons plus en détail chaque élément.

La première partie du SLA (a) décrit ses acteurs, en décrivant leur rôle et leur dénomination. Nous retrouvons les rôles présentés caractérisés par le **NIST** [1] dans la section état de l'art : **Consommateur**, **Fournisseur**, **Auditeur**, **Transporteur** et **Courtier**.

Les termes du service (b) décrivent les caractéristiques des services fournis d'un point de vue fonctionnel. Cette partie correspond plus particulièrement à la description du service couvert par le SLA. Cette description est faite à partir de caractéristiques techniques permettant de définir le service fourni. Parmi ces caractéristiques, nous retrouverons notamment le type de service et ses fonctionnalités.

1. **NIST** : National Institute of Standards and Technology `www.nist.gov`

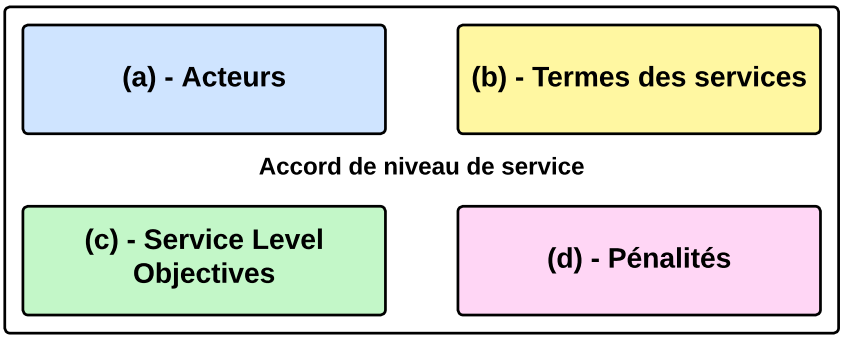


FIGURE 3.1 – Principaux éléments composants un SLA

Par exemple, dans le cadre de la fourniture de machines virtuelles, nous retrouverons, entre autres, dans les termes du SLA fourni le type de service : *IaaS* et les caractéristiques telles que, le type de CPU, de RAM, de stockage et les capacités réseau fournies.

Les *SLOs* (c) sont des caractéristiques spécifiques et mesurables permettant de caractériser la qualité de services cibles à atteindre. Les SLOs expriment les exigences non fonctionnelles (*angl : non-functional requirements, (NFR)*), tels que, la disponibilité, la performance, le coût ou la sécurité [33]. Généralement, ces derniers sont définis pour une période donnée. En d'autres termes, le SLO est utilisé pour décrire la valeur cible d'une caractéristique spécifique et mesurable d'un service, tel qu'un temps de réponse inférieur à 50 ms.

Les pénalités (d) définissent la compensation prévue dans le cas où une violation de SLO est constatée. Ces compensations sont généralement des crédits réutilisables chez le fournisseur [63]. Pour déclencher ces compensations, le client doit prouver que le SLO a été violé. Pour cette raison, la surveillance continue des SLOs est particulièrement importante.

Afin de promouvoir une uniformisation de ces éléments constitutifs et dans les pratiques associées aux SLAs cloud, un récent standard ISO (ISO-19086) a été élaboré, fournissant des recommandations pour leur définition [44, 45]. Ce standard est composé de quatre parties présentant des recommandations de bonnes pratiques présentées dans 4 documents différents :

ISO/IEC 19086-1 : 2016 - définit une base de définition commune pour les SLAs cloud (concept, termes, définitions, contextes) qui peuvent être utilisés pour créer universellement des SLAs cloud.

ISO/IEC 19086-2 : 2018 - définit les caractéristiques de spécifications des métriques des SLAs cloud et inclut des applications du modèle avec des exemples. Ce document établit une terminologie commune pour spécifier les mesures.

ISO/IEC 19086-3 : 2017 - définit les exigences principales attendues d'un SLA cloud conforme à l'ISO/IEC 19086-1.

ISO/IEC 19086-4 : 2019 - spécifie les composants de sécurité et de protec-

tion des informations personnellement identifiables, les objectifs de niveau de service (SLO) et les objectifs de qualité de service (SQO) pour les SLAs cloud.

Ces recommandations sont de plus en plus suivies dans différents travaux tels que les SLAs spécifiques à la sécurité comme présentés dans des travaux académiques [87], ou de l'industrie comme le registre public STAR (Security, Trust and Assurance Registry) [2]. Ce registre documente les offres de services cloud en spécifiant comment sont traités les aspects de sécurité et de gestion de la vie privée par les fournisseurs de ces derniers.

De ces directives, nous reprenons principalement les concepts liés à la définition des SLOs et à la terminologie utilisée. Toutefois, nous ne considérons pas l'intégralité des concepts proposés. En particulier, les éléments spécifiques à la modélisation de la sécurité cloud spécifiés dans les différents documents dépassent le périmètre de notre problématique actuelle, car ils requièrent des études plus détaillées. Cependant, ces derniers pourront être intégrés dans des travaux futurs qui se concentreront spécifiquement sur les aspects de sécurité dans le contexte du cloud.

Notre objectif étant de considérer les SLAs multi-cloud, en l'état ces directives ne suffisent pas à considérer un système multi-cloud. Nous présentons donc dans la prochaine section, la structure de notre modèle de description de SLA multi-cloud.

3.1.2 Structure de SLA multi-cloud

Dans la suite de cette section, nous présentons les définitions de base de notre modèle de description de SLA multi-cloud. Ce modèle de description est principalement à destination des systèmes multi-composant faiblement liés ou micro-service. On pourra parler aussi d'application composite [1]. Nous avons choisi de suivre les recommandations de l'ISO-19086 pour permettre à notre proposition d'être la plus interopérable possible avec tous les fournisseurs de service qui suivront ces lignes directrices. Les définitions qui suivent se basent donc sur ces directives.

Les deux premières définitions posent la base de notre modèle de description avec le SLA (Définition 3.1) et le SLO (Définition 3.2).

Définition 3.1 (SLA) *Un SLA (angl. Service Level Agreement) est défini par un triplet (id, s, SLOs) où : id correspond à un identifiant du SLA, s représente un ensemble de service couvert par le SLA et SLOs définit l'ensemble des SLOs associés aux services comme défini dans la Définition 3.2.*

Définition 3.2 (SLO) *Un SLO (angl. Service Level Objective) est défini comme étant un quintuplet (c, t, v, u, op) où : c représente la caractéristique spécifique mesurée (e.g., disponibilité, temps de réponse, coût, performance ou sécurité) ; t définit le type de c, i.e., quantitatif ou qualitatif ; v définit la valeur attendue de c ; u représente l'unité de mesure de c ; et op* $\in \{<, >, \geq, \leq or =\}$ *définit l'opérateur de comparaison de c.*

2. `https://cloudsecurityalliance.org/star/`

Pour permettre de s'adapter au contexte multi-cloud, nous définissons un SLA multi-cloud (Définition 3.3) comme étant composé de deux types de SLA : un *Global-SLA* et plusieurs *Sub-SLA*. Ces deux types permettent de représenter les exigences de haut niveau du système multi-cloud dans un Global-SLA et les exigences au niveau des composants dans les Sub-SLA. Cette décomposition des SLAs multi-cloud permet une représentation précise des différents SLOs du système multi-cloud et de leurs relations. Cette approche facilite ainsi l'identification des composants touchés en cas de violation de SLA, ce qui permet de réagir plus rapidement et efficacement pour corriger cette violation.

Définition 3.3 (SLA multi-cloud) *Un SLA multi-cloud est défini par un couple* $(sla^{Global}, SLA^{Sub})$ *où :* SLA^{Sub} *est un ensemble de* sla^{Sub} *défini comme* $SLA^{Sub} = \langle sla_n^{Sub}, ..., sla_m^{Sub} \rangle$, sla^{Global} *et* sla^{Sub} *sont défini en suivant le même formalisme que la Définition 3.1. On notera les Sub-SLOs comme étant les SLOs associés au Sub-SLA et les Global-SLOs comme étant les SLOs associés au Global-SLA.*

Nous illustrons ce concept de SLA multi-cloud avec le système multi-cloud de notre exemple de motivation de la section 1.4 à travers la Figure 3.2. Nous voyons que chaque composant du système est associé à un Sub-SLA et un Global-SLA pour le système multi-cloud.

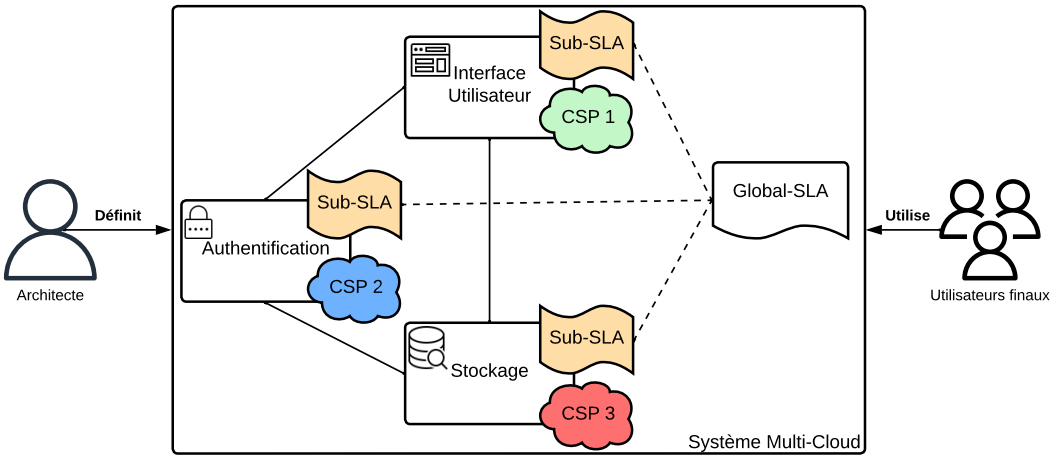


FIGURE 3.2 – Schéma d'un système multi-cloud intégrant un SLA multi-cloud (Définition 3.3)

Le Global-SLA sla^{Global} est composé de 3 services couverts **UI, Auth, Stor** et de 3 SLOs sur le *Temps de réponse*, la *Disponibilité* et le *Coût*. Ce Global-SLA est défini comme :
sla^{Global} = ⟨ id : **EDM**, s : ⟨ **UI, Auth, Stor** ⟩, SLOs ⟩ où SLOs est l'ensemble des objectifs de niveau de service suivants :

$$slo_1 = (Temps\ de\ réponse, Quantitative, 20, ms, \leq),$$
$$slo_2 = (Disponibilité, Quantitative, 99.9, \%, \geq),$$
$$slo_3 = (Coût, Quantitative, 10, €/h, \leq)$$

Ainsi, que le SLA^{Sub} composé de 3 Sub-SLAs $\langle sla_{UI}^{Sub}, sla_{Auth}^{Sub}, sla_{Stor}^{Sub} \rangle$ chacun composé de 2 SLOs sur la *Disponibilité* et le *Temps de réponse* défini comme suit : $sla_{UI}^{Sub} = \langle$ id : **UI**, s : **UI**, SLOs $\rangle$ où SLOs est l'ensemble des objectifs de niveau de service suivants :

$$slo_1 = (\textit{Temps de réponse}, \textit{Quantitative}, 5, ms, \leq),$$
$$slo_2 = (\textit{Disponibilité}, \textit{Quantitative}, 99.8, \%, \geq)$$

$sla_{Auth}^{Sub} = \langle$ id : **Auth**, s : **Auth**, SLOs $\rangle$ où SLOs est l'ensemble des objectifs de niveau de service suivants :

$$slo_1 = (\textit{Temps de réponse}, \textit{Quantitative}, 5, ms, \leq),$$
$$slo_2 = (\textit{Disponibilité}, \textit{Quantitative}, 99.8, \%, \geq)$$

$sla_{Stor}^{Sub} = \langle$ id : **Stor**, s : **Stor**, SLOs $\rangle$ où SLOs est l'ensemble des objectifs de niveau de service suivants :

$$slo_1 = (\textit{Temps de réponse}, \textit{Quantitative}, 5, ms, \leq),$$
$$slo_2 = (\textit{Disponibilité}, \textit{Quantitative}, 99.8, \%, \geq)$$

On illustre dans cet exemple ce modèle de description qui permet la description des SLOs de chaque composant ainsi que le système multi-cloud complet de l'exemple de motivation. Toutefois, ce modèle ne permet pas de représenter la dynamicité des services cloud.

Dans cette section, nous avons présenté notre modèle de description de SLA multi-cloud. Dans la section suivante, nous présenterons comment sont décrites les stratégies de reconfiguration.

3.2 Stratégies de reconfiguration

Dans cette section, nous présentons dans un premier temps une introduction aux concepts fondamentaux des stratégies de reconfiguration dans la sous-section 3.2.1. Ensuite, dans la sous-section 3.2.2, nous définissons le formalisme de machine à états permettant de représenter une stratégie de reconfiguration. De plus, dans la sous-section 3.2.2.2, nous présentons notre proposition de modèle de description de SLA multi-cloud. En suivant les définitions proposées dans les différentes sections, nous modélisons un exemple de stratégie de reconfiguration.

3.2.1 Lien entre SLA et stratégies de reconfiguration

Par essence, les SLAs sont statiques et ne considèrent pas la dynamicité des services couverts, ce qui est pourtant un concept clé du cloud computing d'après

la définition faite par le **NIST** [68]. Cette dynamicité est mise en œuvre à travers des stratégies de reconfiguration définies par des architectes de système multi-cloud pour modéliser les comportements dynamiques des services.

Une stratégie de reconfiguration désigne un ensemble prédéfini d'actions et de règles orchestrées pour adapter dynamiquement les composants d'un système multi-cloud. Cette adaptation vise à maintenir les SLOs. Ces stratégies de reconfiguration peuvent inclure des actions telles que l'ajustement des capacités de calcul, l'allocation de mémoire ou la migration de charges de travail entre différents fournisseurs de services cloud. L'objectif principal de ces stratégies est de répondre aux variations de la charge de travail, aux pannes éventuelles ou aux changements dans les conditions du fournisseur afin d'assurer une exécution optimale du système tout en respectant les SLOs.

Pour représenter ces stratégies de reconfiguration formellement, nous proposons d'utiliser des machines à états. Les machines à états [82], également connues sous le nom d'automates finis, sont des modèles abstraits qui décrivent le comportement d'un système en fonction de différents états et de transitions possibles entre ces états. Chaque machine à états est constituée d'un ensemble fini d'états distincts et d'une série de règles définissant les conditions pour passer d'un état à un autre en réponse à des événements spécifiques. Lorsque le système reçoit un événement, il se trouve dans un état donné, puis suit la transition appropriée pour atteindre un nouvel état. Ces transitions peuvent être déclenchées par des entrées externes, des actions internes ou des conditions internes du système. Les machines à états offrent une représentation claire et formelle du comportement dynamique des systèmes, ce qui les rend largement utilisées pour modéliser divers processus dans différents domaines.

Dans la sous-section suivante, nous présentons le formalisme des machines à états que nous utilisons.

3.2.2 Modèle de description de SLA multi-cloud sensible aux stratégies de reconfiguration

3.2.2.1 Machine à état

Nous avons donc choisi d'utiliser le formalisme des machines à états pour son adoption généralisée, son intuitivité de définition, son efficacité pour modéliser le comportement dynamique d'un système et la possibilité de formellement la valider. Nous présentons notre définition de la stratégie de reconfiguration dans la Définition 3.4.

Définition 3.4 (Stratégie de reconfiguration) *Une stratégie de reconfiguration est définie par une machine à états (SM) décrite par un couple (S, T) où : S est l'ensemble des états de la machine à états en suivant la définition 3.5 et T est l'ensemble de transitions de la machine à états en suivant la définition 3.7.*

Pour illustrer les différents concepts présentés dans les définitions de cette sous-

section, nous présentons dans la Figure 3.3 la machine à états associés au composant "UI" de notre exemple de motivation de la section 1.4. Cette machine a état est composée de 4 états et de 7 transitions, qui décrivent le comportement dynamique du composant d'interface utilisateur.

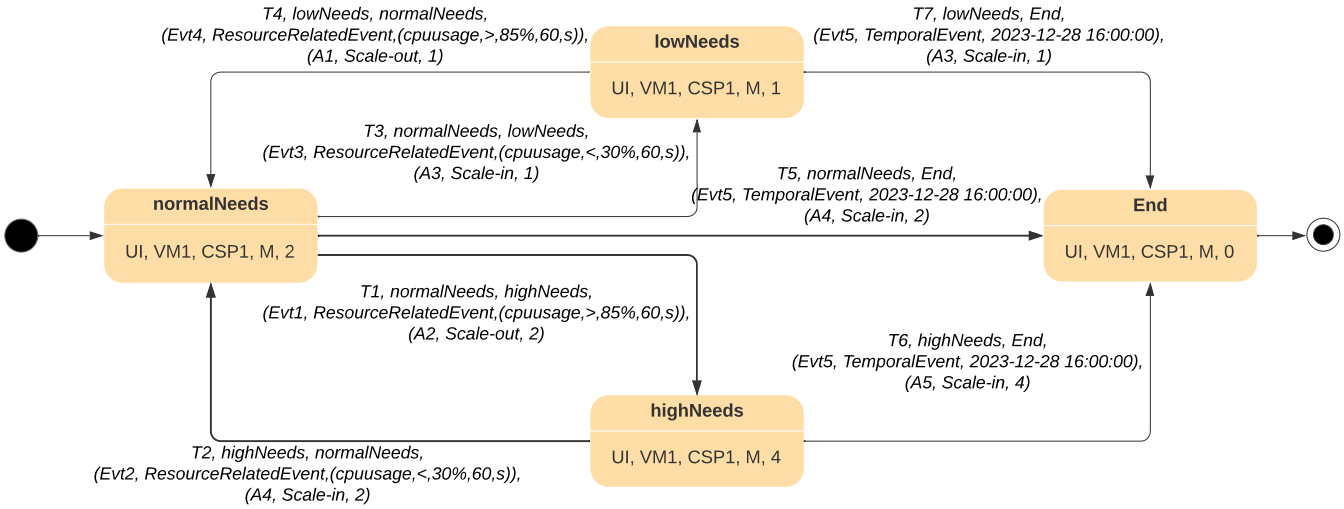


FIGURE 3.3 – Représentation graphique de la machine à états lié au composant UI

Définition 3.5 (État) *Un état est défini par un triplet (l, t, R) où : l est le nom de l'état ou **label** ; $t \in \{isInitial, isNormal, isFinal\}$ décris le type d'état et indique si l'état est initial, intermédiaire ou final, respectivement ; et $\boldsymbol{R}$ est un ensemble de ressources cloud $\boldsymbol{r}$ (Définition 3.6) qui compose les éléments d'une application ou une application composite caractérise un état.*

Nous considérons une définition simple des ressources cloud r suivant le formalisme présenté dans la définition 3.6.

Définition 3.6 (Ressource) *Une ressource est définie par un quintuplet $(id, CSP, typeRessource, size, nb)$ où : id est l'identifiant de la ressource, CSP est le nom du fournisseur de service, $typeRessource$ correspond au type de la ressource, $size$ définit la taille de la ressource, nb correspond au nombre de ressources.*

Tout d'abord, les états de la machine à états de notre exemple de la Figure 3.3 sont représentés dans le Tableau 3.1. Par exemple, l'état $normalNeeds$ est initial et composé de 2 machines virtuelles de taille M pour le composant UI.

Définition 3.7 (Transition) *Une transition est définie comme étant un quintuplet (id, s_s, s_t, E, A) où : id est l'identifiant de la transition, s_s est l'état source, s_t est l'état de destination, E est un ensemble d'événements (Définition. 3.8) qui peuvent déclencher la transition et A est un ensemble d'actions de reconfiguration (Définition. 3.9) exécutées quand la transition est déclenchée.*

TABLEAU 3.1 – États de la machine à états associés au Sub-SLA du composant UI

Label	**Type**	**Ressources**				
		id	**idResource**	**CSP**	**size**	**nb**
$normalNeeds$	$isInitial$	UI	$VM1$	$CSP1$	M	2
$highNeeds$	$isNormal$	UI	$VM1$	$CSP1$	M	4
$lowNeeds$	$isNormal$	UI	$VM1$	$CSP1$	M	1
end	$isFinal$	UI	$VM1$	$CSP1$	M	0

Puis, les transitions de la machine à états de notre exemple (Figure 3.3) sont représentées dans le Tableau 3.2. Par exemple, la transition T_1 permet de passer de l'état $normalNeeds$ vers l'état $highNeeds$ grâce à l'action A_2 quand l'événement evt_1 survient.

TABLEAU 3.2 – Transitions de la machine à états associée au Sub-SLA du composant UI

Id	**Source**	**Destination**	**Évènements**	**Actions**
T_1	$normalNeeds$	$highNeeds$	evt_1	A_2
T_2	$highNeeds$	$normal$	evt_2	A_4
T_3	$normalNeeds$	$lowNeeds$	evt_3	A_3
T_4	$lowNeeds$	$normal$	evt_4	A_1
T_5	$normalNeeds$	end	evt_5	A_4
T_6	$highNeeds$	end	evt_5	A_5
T_7	$lowNeeds$	end	evt_5	A_3

Une transition est activée à travers un ou plusieurs événements qui vont déclencher une ou plusieurs actions de reconfiguration. Ces événements représentent l'occurrence de tout changement entraînant l'activation d'actions spécifiques. Ces actions permettent la reconfiguration d'un service. Nous présentons formellement un événement dans la Définition 3.8.

Définition 3.8 (Évènement) *Un événement est représenté comme un triplet (id, t, p) où : id est l'identifiant de l'événement ; t ∈ {liés au temps, liés aux ressources, liés aux actions utilisateurs, composite} correspond au type de l'événement, les événements* ***liés au temps*** *se produisent à une date donnée ou après un certain laps de temps,* ***liés aux ressources*** *se produisent lorsqu'une mesure de ressource atteint une valeur de référence définie,* ***liés aux actions utilisateur*** *se produisent à la demande d'un utilisateur,* ***composite*** *se composent de plusieurs types d'événement spécifiés précédemment ; et p représente le prédicat dont le corps dépendra du type d'événement. En effet, le prédicat décrit ce qui déclenchera une action de reconfiguration : pour un événement lié au temps, ce sera le moment d'activation de*

l'événement ; pour un événement lié aux ressources, ce sera la métrique observée et le seuil ; pour un événement lié aux actions utilisateurs, ce sera l'action utilisateur.

Les événements composant les transitions de cette machine à états sont représentés dans le Tableau 3.3. Par exemple, l'événement evt_1 se déclenche quand la moyenne d'utilisation de CPU est supérieure à 85% pendant 60 secondes et l'événement evt_5 se déclenche le 28/12/2023 à 16h00.

TABLEAU 3.3 – Évènements de la machine à états associés au Sub-SLA du composant UI

Id	**Type**	**Prédicat**
evt_1	liés aux ressources	$cpuUsage, average, >, 85, \%, 60s$
evt_2	liés aux ressources	$cpuUsage, average, <, 30, \%, 60s$
evt_3	liés aux ressources	$cpuUsage, average, <, 30, \%, 60s$
evt_4	liés aux ressources	$cpuUsage, average, >, 85, \%, 60s$
evt_5	liés au temps	$2023 - 12 - 28\ 16 : 00 : 00$

Les actions de reconfiguration représentent les mécanismes associés avec les services cloud. Chacun des différents types d'actions requiert un ensemble d'attributs pour être exécuté, par exemple, le service cible ou les attributs cibles (voir définition 3.9).

Définition 3.9 (Action de Reconfiguration) *Une action de reconfiguration est définie par un triplet (id, t, AA) où : id est l'identifiant de l'action de reconfiguration ; $t \in \{HorizontalScaling, VerticalScaling, Migration, ApplicationReconfiguration, BasicAction\}$ correspond au type d'action de reconfiguration ; une action d'**Élasticité Horizontal** ajoute ou supprime des ressources, une action d'**Élasticité Vertical** augmente ou diminue les caractéristiques des ressources, une action de **Migration** migre la ressources vers un autre fournisseur, une action de **Re-configuration d'application** mets à jour les attributs des ressources, une **Action basique** telles que démarrer, arrêter ou redémarrer ; et AA est un ensemble d'attributs d'action requis pour exécuter l'action.*

Les actions de reconfiguration sont représentées comme suit dans le Tableau 3.4. Par exemple, l'action A_1 est une action d'élasticité horizontale qui ajoute une machine virtuelle au composant UI.

En utilisant les différentes définitions de cette sous-section, nous présentons dans la sous-section suivante comment nous enrichissons le modèle de description de SLA multi-cloud avec des stratégies de reconfiguration.

3.2.2.2 Modélisation des stratégies de reconfiguration à l'aide de machines à états

Dans les sections précédentes, nous avons présenté notre modèle de description de SLA multi-cloud ainsi qu'un modèle de description des stratégies de reconfiguration

Tableau 3.4 – Actions de reconfiguration de la machine à états associée au Sub-SLA du composant UI

Id	**Type**	[**Attributs d'action**]
A_1	HorizontalScaling	[UI, scale-out, 1]
A_2	HorizontalScaling	[UI, scale-out, 2]
A_3	HorizontalScaling	[UI, scale-in, 1]
A_4	HorizontalScaling	[UI, scale-in, 2]
A_5	HorizontalScaling	[UI, scale-in, 4]

sous forme de machine à états.

Toutefois, la dynamicité n'est pas supportée dans les différents modèles de description de SLA identifiés. Il est donc nécessaire d'intégrer dans le SLA des mécanismes d'élasticité. Le maintien des SLAs est étroitement lié à la nature dynamique des services et aux mécanismes d'élasticité mis en place. La dynamicité inhérente aux charges de travail et aux ressources disponibles dans un environnement multi-cloud peut impacter la capacité d'un système à maintenir les SLAs définies. Les mécanismes d'élasticité jouent alors un rôle crucial en ajustant automatiquement les ressources allouées en fonction des besoins fluctuants. Ces mécanismes assurent que les performances, la disponibilité et d'autres paramètres importants restent en conformité avec les exigences définies dans les SLA. Grâce à une gestion pro-active des ressources, les mécanismes d'élasticité contribuent à atténuer les risques de violation des SLA, même en présence de variations imprévues de la demande ou des conditions du système. Pour cette raison, nous présentons dans cette section, comment enrichir notre modèle de description de SLA multi-cloud avec les stratégies de reconfiguration.

Pour gérer cette dynamicité, nous enrichissons le modèle de description de SLA multi-cloud proposé dans la section 3.2 avec des machines à états définissant les stratégies de reconfiguration associées aux composants d'un système multi-cloud. Dans la suite de cette thèse, nous utiliserons le terme "SLA multi-cloud dynamique" pour désigner ce modèle. Sa structure, présentée dans la définition 3.10, enrichit la description des SLAs décrite dans la définition 3.1.

Définition 3.10 (SLA dynamique) *Un `SLA dynamique` est défini par un 4-tuple (id, s, SLOs, SM) où : id, s et SLOs sont définis comme dans la définition 3.1 et SM est défini comme dans la définition 3.4.*

Pour terminer cet exemple, nous considérons à nouveau ici le Sub-SLA associé au composant "UI" de notre exemple de motivation (section 1.4). Le Sub-SLA dynamique associé est représenté par :
$\langle\ UI, (slo_1, slo_2, slo_3)$,SM $\rangle$ où : UI est le nom du Sub-SLA, (slo_1, slo_2, slo_3) est l'ensemble de SLOs associé au Sub-SLA UI, $SM = \langle S, T \rangle$ représente la stratégie de reconfiguration associée au Sub-SLA UI où S est l'ensemble d'états décrits dans le Tableau 3.1 et T est l'ensemble des transitions décrites dans le Tableau 3.2.

3.3 Mise en œuvre et évaluation

Dans cette section, notre objectif est d'implémenter un outil preuve de concept permettant aux architectes d'utiliser notre modèle de description de SLA multi-cloud pour décrire leur système multi-cloud. Pour ce faire, nous présentons dans la section 3.3.1 un modèle conceptuel de notre modèle formel de la section précédente permettant d'implémenter une preuve de concept présenté dans le section 3.3.2. Notre second objectif est d'évaluer notre modèle vis-à-vis des autres modèles de la littérature que nous effectuons dans la section 3.3.3.

3.3.1 Modèle conceptuel de SLA multi-cloud dynamique

Dans cette section, nous présentons les différents composants de notre modèle de description de SLA multi-cloud. Les principaux composants du modèle sont représentés dans la Figure 3.4 : SLA, SLOs, Termes de services couverts et stratégies de reconfiguration. Nous détaillons chacun de ces composants dans les sous-sections suivantes. Le diagramme complet du modèle est présenté dans la Figure 3.10.

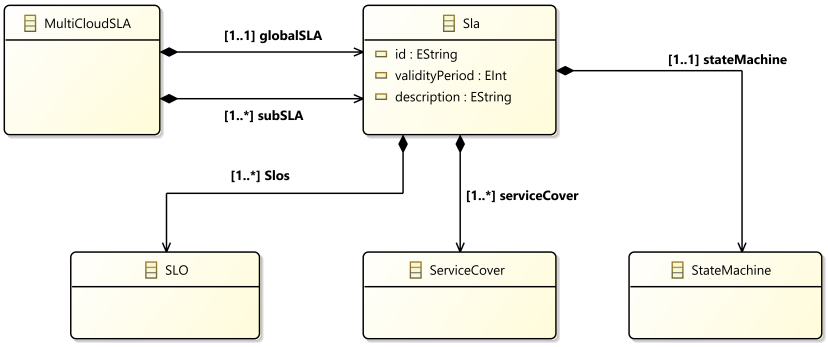


FIGURE 3.4 – Schéma abstrait du modèle de description de SLA multi-cloud enrichi

3.3.1.1 SLA

Le Sla est le composant principal de notre modèle, il est défini par un identifiant (*id*), une période de validité (*validityPeriod*) et d'une description (*description*). On note que le SLA multi-cloud est composé d'un Global-SLA et d'au minimum un Sub-SLA. La Figure 3.5 décrit la modélisation d'un SLA.

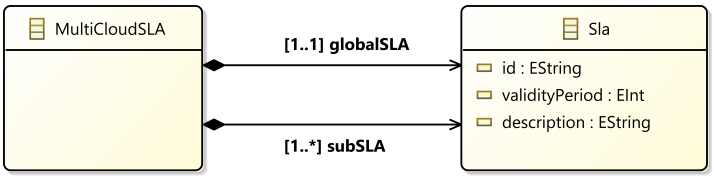


FIGURE 3.5 – SLA

3.3.1.2 SLOs

Le modèle de description complet d'un SLO est composé de métriques (*Metric*) et de pénalités. On note que les métriques peuvent être composées de trois éléments : paramètre, règle et expression. La règle est utilisée pour contraindre une métrique et indiquer la ou les méthodes possibles de mesure. L'expression permet de définir les méthodes de calcul des métriques. Le paramètre est utilisé pour ajouter une constante aux calculs des expressions ou des règles. Cette modélisation suit les directives de définition d'un SLO proposée dans l'ISO-19086-2 [45]. La Figure 3.6 décrit la modélisation d'un SLO.

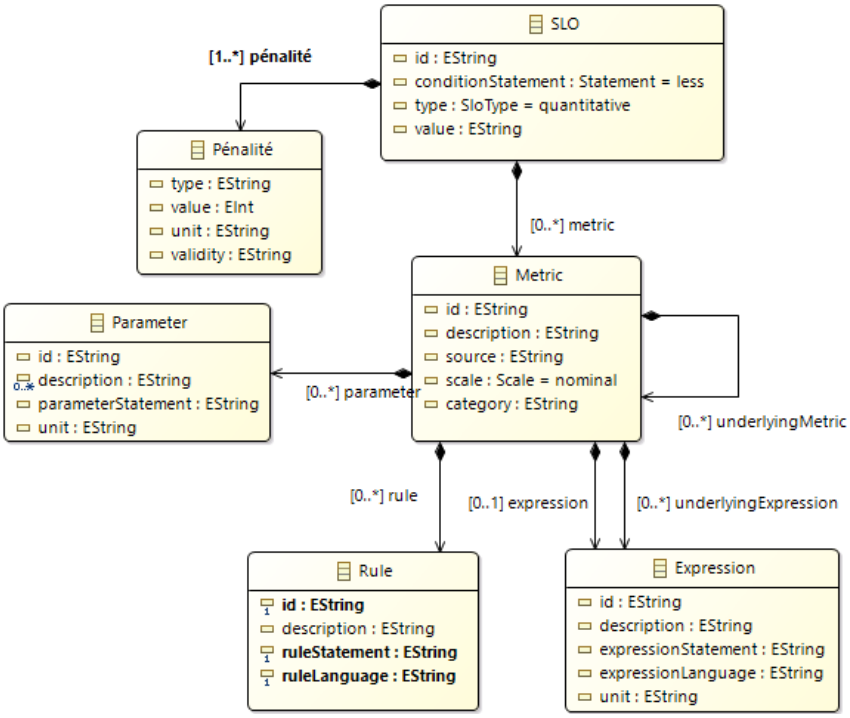


FIGURE 3.6 – SLO

3.3.1.3 Termes des services couverts

Les termes des services couverts sont décrits en utilisant le formalisme proposé dans [40]. Les services couverts sont composés d'un à plusieurs services définissant une entité du modèle de description des ressources cloud décrit au moyen d'un à plusieurs attributs. La Figure 3.7 décrit la modélisation d'un service couvert.

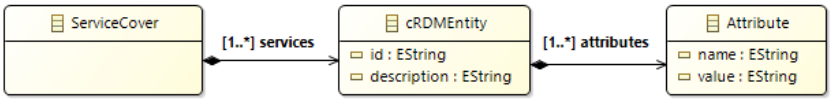


FIGURE 3.7 – Services couverts

3.3.1.4 Stratégies de reconfiguration

Les stratégies de reconfiguration sont définies comme décrites dans la section 3.2 sous forme de machine à états. Ces machines à états sont composées d'au moins deux états et d'au moins une transition. Un état est composé de services représentés comme dans la section 3.3.1.3. Une transition est composée d'un événement et d'au moins une action de reconfiguration. La Figure 3.8 décrit la modélisation d'un service couvert.

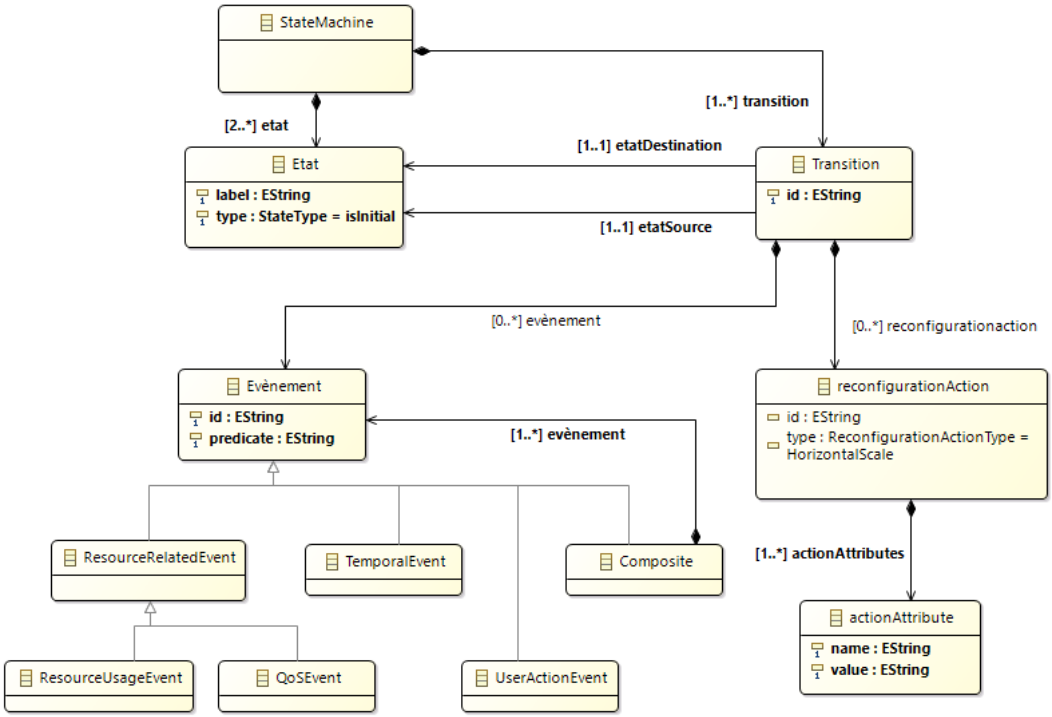


FIGURE 3.8 – Machine à états

3.3.2 Mise en œuvre de la preuve de concept

Nous représentons les SLAs multi-cloud au format YAML. Pour ce faire, nous utilisons un vérificateur de fichier YAML en python. Ce dernier nous permet de formaliser le schéma que doit suivre un SLA multi-cloud et de vérifier que le fichier YAML correspond au modèle de description que nous proposons. Nous créons un schéma de fichier qui nous permettra ensuite de valider sa conformité avec notre modèle et remonter les erreurs syntaxiques en cas de mauvaise définition. Pour cette tâche, nous avons choisi d'utiliser la bibliothèque python 'Schema' [3]. Cette bibliothèque offre une approche simple pour définir des schémas de validation en python. Elle nous permet de définir les règles et les contraintes (Termes et structure de données à utiliser) qui doivent être respectées par la description en YAML représentant un SLA multi-cloud. Nous présentons un exemple de SLA dynamique défini en un fichier YAML dans les listing 3.1 à 3.5. Le premier listing 3.1 représente

3. `https://github.com/keleshev/schema`

les éléments suivant le nom du SLA ligne 2, la période de validité de la ligne 3 à 5, une description du sub-SLA ligne 6 et deux sub-SLOs de la ligne 7 à 19 : un SLO de temps de réponse inférieur ou égale à 5 ms de la ligne 8 à 13 et un SLO de disponibilité supérieur ou égale à 99.8% de la ligne 14 à 19. Le second listing 3.2 représente le service couvert par le sub-SLA et les ressources composants ce service, dans cet exemple une seule ressource est représentée avec les éléments suivant un id de ressource $VM1$ ligne 3, une description de la ressource ligne 4, les attributs décrivant la ressource par son nom ligne 6, son type ligne 7, son fournisseur ligne 8, sa taille ligne 9 et son nombre ligne 10. Le troisième listing correspond à la première partie de la machine à état associé au sub-SLA définissant les états. Cette dernière est composée de 4 états :

normalNeeds (ligne 3-6) de type initial et composé de 2 ressource $VM1$ fournis par $CSP1$ de type VM de taille M,

lowNeeds (ligne 7-10) de type normal et composé de 1 ressource $VM1$ fournis par $CSP1$ de type VM de taille M,

highNeeds (ligne 11-14) de type normal composé de 4 ressource $VM1$ fournis par $CSP1$ de type VM de taille M et

End (ligne 15-18) de type final composé de 0 ressource $VM1$ fournis par $CSP1$ de type VM de taille M.

```yaml
---
name: "VM1-SLA"
validityPeriod:
  start: "2020-07-01 8:00:00"
  end: "2023-12-28 16:00:00"
description: "SLA Description of VM1"
Objectives:
  - id: Obj_1
    name: ResponseTime
    conditionStatement: '<='
    type: quantitative
    value: 5
    unit: "ms"
  - id: Obj_2
    name: Availability
    conditionStatement: '>='
    type: "quantitative"
    value: "99.8"
    unit: "%"
```

Listing (3.1) Sub-SLA décris en YAML - SLOs

```yaml
ServiceCover:
  resources:
    - id: VM1
      description: "VM1"
      attributes:
        name: "VM1"
        type: "VM"
        provider: "CSP1"
        size: "M"
        number: 2
```

Listing (3.2) Sub-SLA décris en YAML - Service Couvert

```yaml
StateMachine:
  States:
    - id: normalNeeds
      type: "isInitial"
      resources:
        - VM1: {provider: "CSP1", type: "VM", size: "M", number: 2}
    - id: lowNeeds
      type: "isNormal"
      resources:
        - VM1: {provider: "CSP1", type: "VM", size: "M", number: 1}
    - id: highNeeds
      type: "isNormal"
      resources:
        - VM1: {provider: "CSP1", type: "VM", size: "M", number: 4}
    - id: End
      type: "isFinal"
      resources:
        - VM1: {provider: "CSP1", type: "VM", size: "M", number: 0}
```

Listing (3.3) Sub-SLA décris en YAML - Machine à états

Les listing 3.4 et 3.5 représente la seconde partie de la machine à état avec les 7 transitions suivantes :

$T1$ (listing 3.4, ligne.2-18) a pour source l'état $normalNeeds$ et pour destination l'état $highNeeds$ l'événement evt_1 lié aux ressources avec pour prédicat une utilisation CPU supeérieure à 85% pour une période de 60 secondes déclenche l'action $A2$ de $Scale - out$ de 2 ressources $VM1$.

$T2$ (listing 3.4, ligne.19-35) a pour source l'état $highNeeds$ et pour destination l'état $normalNeeds$ l'événement evt_2 lié aux ressources avec pour prédicat une utilisation CPU inférieure à 30% pour une période de 60 secondes déclenche l'action $A4$ de $Scale - in$ de 2 ressources $VM1$.

$T3$ (listing 3.4, ligne.36-52) a pour source l'état $normalNeeds$ et pour destination l'état $lowNeeds$ l'événement evt_3 lié aux ressources avec pour prédicat une utilisation CPU inférieur à 30% pour une période de 60 secondes déclenche l'action $A3$ de $Scale - in$ de 1 ressource $VM1$.

$T4$ (listing 3.4, ligne.53-69) a pour source l'état $lowNeeds$ et pour destination l'état $normalNeeds$ l'événement evt_4 lié aux ressources avec pour prédicat une utilisation CPU supeérieure à 85% pour une période de 60 secondes déclenche l'action $A1$ de $Scale - out$ de 1 ressource $VM1$.

$T5$ (listing 3.5, ligne.1-13) a pour source l'état $normalNeeds$ et pour destination l'état End l'événement evt_5 lié au temps avec pour prédicat la date du 28/12/2023 à 16h déclenche l'action $A4$ de $Scale-in$ de 2 ressources $VM1$.

$T6$ (listing 3.5, ligne.14-26) a pour source l'état $highNeeds$ et pour destination l'état End l'événement evt_5 lié au temps avec pour prédicat la date du 28/12/2023 à 16h déclenche l'action $A5$ de $Scale-in$ de 4 ressources $VM1$.

$T7$ (listing 3.5, ligne.27-39) a pour source l'état $lowNeeds$ et pour destination l'état End l'événement evt_5 lié au temps avec pour prédicat la date du 28/12/2023 à 16h déclenche l'action $A3$ de $Scale - in$ de 1 ressource $VM1$.

```yaml
Transitions:
  - id: T1
    stateSource: "normalNeeds"
    stateTarget: "highNeeds"
    events:
      - id: "evt_1"
        type: "ResourceRelatedEvent"
        predicate:
          metric: "cpuusage"
          operator: ">"
          refValue: 85%
          time: "60s"
    actions:
      - id: "A2"
        type: "Scale-out"
        attributes:
          resource: "VM1"
          number: "2"
  - id: T2
    stateSource: "highNeeds"
    stateTarget: "normalNeeds"
    events:
      - id: "evt_2"
        type: "ResourceRelatedEvent"
        predicate:
          metric: "cpuusage"
          operator: "<"
          refValue: 30%
          time: "60s"
    actions:
      - id: "A4"
        type: "Scale-in"
        attributes:
          resource: "VM1"
          number: "2"
  - id: T3
    stateSource: "normalNeeds"
    stateTarget: "lowNeeds"
    events:
      - id: "evt_3"
        type: "ResourceRelatedEvent"
        predicate:
          metric: "cpuusage"
          operator: "<"
          refValue: 30%
          time: "60s"
    actions:
      - id: "A3"
        type: "Scale-in"
        attributes:
          resource: "VM1"
          number: "1"
  - id: T4
    stateSource: "lowNeeds"
    stateTarget: "normalNeeds"
    events:
      - id: "evt_4"
        type: "ResourceRelatedEvent"
        predicate:
          metric: "cpuusage"
          operator: ">"
          refValue: 85%
          time: "60s"
    actions:
      - id: "A1"
        type: "Scale-out"
        attributes:
          resource: "VM1"
          number: "1"
```

Listing (3.4) Sub-SLA décris en YAML - Transition T1 à T4

```yaml
  - id: T5
    stateSource: "normalNeeds"
    stateTarget: "End"
    events:
      - id: "evt_5"
        type: "TemporalEvent"
        predicate: "2023-12-28 16:00:00"
    actions:
      - id: "A4"
        type: "Scale-in"
        attributes:
          resource: "VM1"
          number: "2"
  - id: T6
    stateSource: "highNeeds"
    stateTarget: "End"
    events:
      - id: "evt_5"
        type: "TemporalEvent"
        predicate: "2023-12-28 16:00:00"
    actions:
      - id: A5
        type: "Scale-in"
        attributes:
          resource: "VM1"
          number: "4"
  - id: T7
    stateSource: "lowNeeds"
    stateTarget: "End"
    events:
      - id: "evt_5"
        type: "TemporalEvent"
        predicate: "2023-12-28 16:00:00"
    actions:
      - id: "A3"
        type: "Scale-in"
        attributes:
          resource: "VM1"
          number: "1"
```

Listing (3.5) Sub-SLA décris en YAML - Transition T5 à T7

3.3.3 Évaluation

Dans cette section, nous présentons une comparaison de notre langage avec les autres langages de spécification de SLA de l'état de l'art (section 2.2.1.2). Nous nous intéressons principalement à deux critères qui sont : (*i*) le support de SLA multi-cloud et (*ii*) le support de la dynamicité. Cette comparaison est résumée dans le Tableau 3.5.

TABLEAU 3.5 – Évaluation de notre modèle de description par rapport aux autres langages de spécification de SLA de la littérature

Nom	Année	Multi-Cloud	Dynamicité des services
SLA* [50]	2010	Non	Non
SLAC [90]	2014	Oui	Partiel
rSLA [62]	2015	Non	Non
CSLA [52]	2016	Oui	Partiel
ySLA [32]	2018	Non	Non
Notre Proposition	2022	Oui	Oui

Pour évaluer le niveau d'expressivité de notre SLA multi-cloud enrichi, nous suivrons une méthode d'évaluation bien connue dans la littérature [43] : la comparaison de modèles avec des normes ou d'autres modèles proposés. L'objectif de cette méthode est de dénombrer les concepts représentables par les autres modèles de la littérature. Puis, d'évaluer le nombre de concepts que notre modèle peut représenter.

Dans cette comparaison, nous considérons les 5 langages de spécification suivants : SLA* [50], CSLA [52], SLAC [90], rSLA [62] et ySLA [32] présentés dans le Tableau 3.5. Ces derniers ont été présentés en détail dans le chapitre 2. La Figure 3.9 illustre les concepts couverts par notre modèle de description de SLA multi-cloud dynamique.

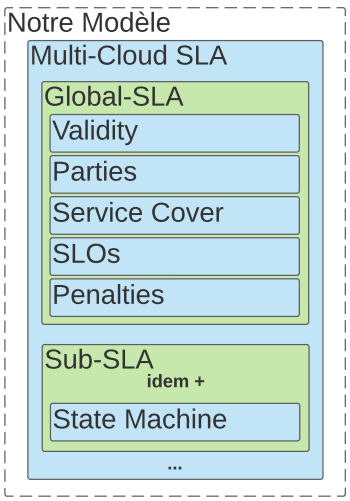


FIGURE 3.9 – Diagramme des composants de notre modèle de SLA Multi-Cloud

Les concepts considérés sont principalement la capacité du langage à représenter : un *SLO*, un *SLO* composite, les services couverts, les pénalités et la dynamicité. L'évaluation du niveau de couverture est représentée dans le Tableau 3.6. Chacun des différents langages est présenté plus en détail dans le chapitre 2, section 2.2.1.2.

TABLEAU 3.6 – Évaluation du niveau de couverture de notre modèle de SLA multi-cloud enrichi par rapport aux concepts des autres langages de spécification de SLA

	Concepts	Notre couverture	Taux de couverture
SLA* [50]	5	5	100%
CSLA [52]	9	7	70%
SLAC [90]	6	5	80%
rSLA [62]	6	6	100%
ySLA [32]	6	5	80%

SLA* et rSLA sont couverts à 100% car ils sont un portage et une abstraction des concepts proposés dans WSLA [51] dans le contexte du cloud et ne supporte que les concepts basiques du SLA.

Nous notons que notre proposition couvre la majorité des concepts de la littérature, à l'exception de deux concepts spécifiques proposés dans *CSLA* et *SLAC*. Ces concepts spécifiques sont liés à la dynamicité des SLOs (fluctuation des SLOs, par exemple, réduction du SLO de temps de réponse pendant la phase de mise en œuvre du SLA) et sont difficilement applicables à notre contexte multi-cloud en raison de la diversité des composants d'un système et de leur interdépendance. Avec SLAC, Uriarte et al. proposent un langage de SLA où les parties prenantes peuvent définir à l'avance plusieurs niveaux de service. Dans CSLA, Kouki et al. prennent en compte la dynamique des SLOs avec les concepts de '*fuzziness*' et de '*confidence*'. '*Fuzziness*' définit des marges acceptables pour les SLOs et '*confidence*' indique un pourcentage de conformité aux objectifs. Pour finir, nous remarquons que dans ySLA un concept intégrant en plus les concepts de '*Templates*' (ou modèle) et de '*Scopes*' (ou champ d'application des SLOs). Les '*Templates*' permettent de définir des modèles réutilisables pour différents services. Toutefois, ces concepts sortent de l'objectif de notre approche pour la représentation de SLA multi-cloud.

Dans cette section, nous avons évalué notre langage de spécification de SLA multi-cloud dynamique face aux autres modèles de la littérature. Cela nous a permis de valider que notre modèle supporte les concepts de la littérature et de présenter ses avantages par rapport aux autres modèles.

3.4 Conclusion

Dans ce chapitre, nous avons présenté notre modèle de description de SLA multi-cloud enrichi par des stratégies de reconfiguration représentées sous forme de machine à états. Nous avons détaillé les différents éléments composant notre SLA multi-

cloud (le Global et les Sub-SLAs). Ensuite, notre approche a été proposée en tirant parti du format YAML. Pour finir, l'expressivité du modèle a pu être évaluée en le comparant aux concepts des autres langages de spécification de SLA de l'état-de-l'art. Une partie des travaux présentés dans ce chapitre ont été publiés dans l'article de journal suivant : *Jeremy Mechouche, Roua Touihri, Mohamed Sellami, Walid Gaaloul : Conformance Checking for Autonomous Multi-Cloud SLA Management & Adaptation, Journal of Supercomputing 78(11) : 13004-13039, 2022*

Cependant, il est important de noter que le SLA multi-cloud enrichi est formalisé par un utilisateur, ce qui implique la possibilité de lacunes ou d'erreurs. L'utilisateur peut omettre certains cas spécifiques, faire des erreurs lors de la définition de machine à états ou même établir des Global-SLAs qui sont incohérents avec les Sub-SLAs. Par exemple, définir des SLOs dans le Global-SLA qui n'ont pas de lien avec les SLOs des Sub-SLAs. Par conséquent, nous nous intéressons dans le prochain chapitre à la vérification des éléments constitutifs de SLA multi-cloud avant la mise en application du SLA multi-cloud enrichi.

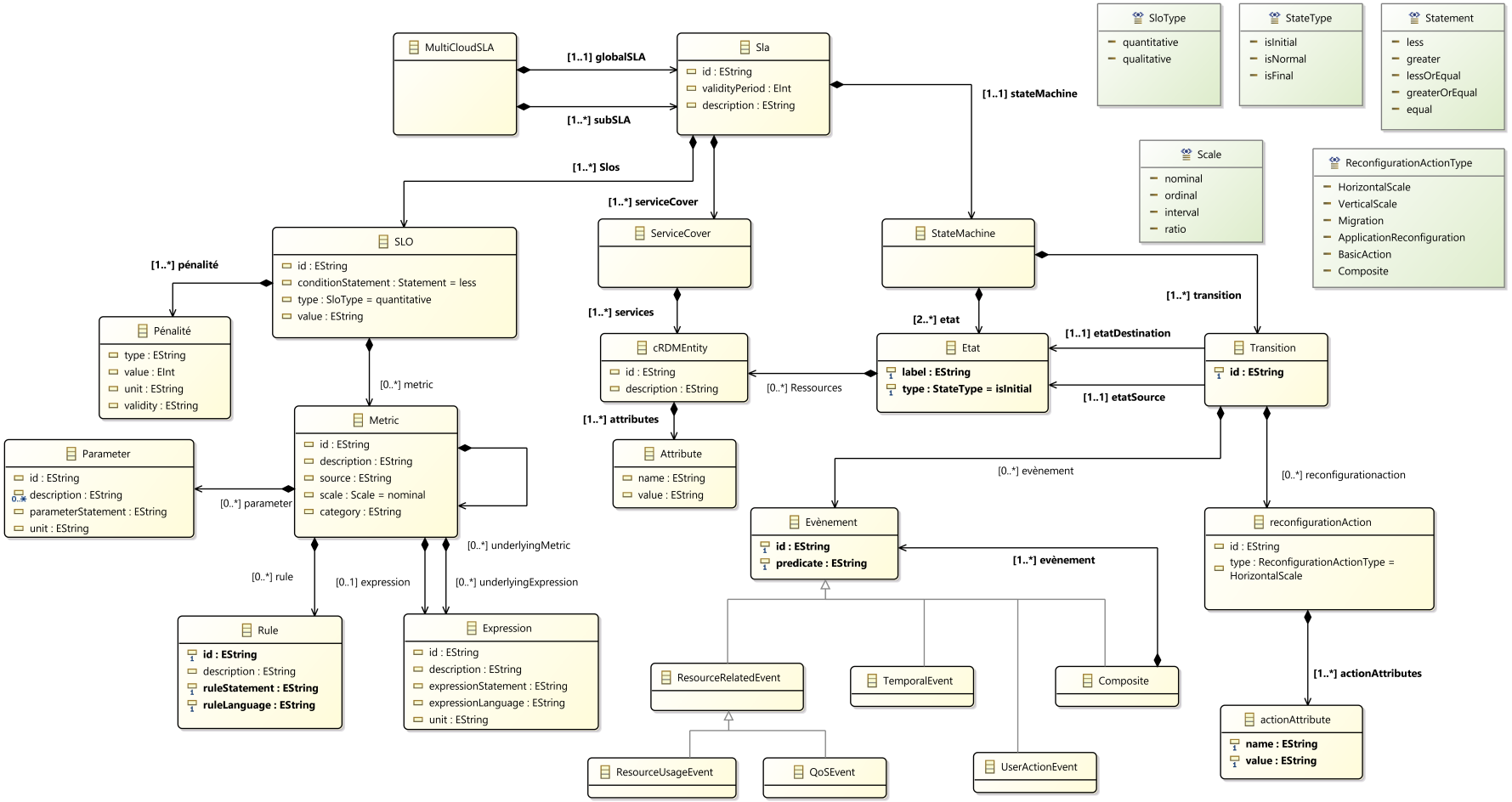


FIGURE 3.10 – Diagramme de classe de SLA multi-cloud

Chapitre 4

Cohérence de SLA multi-cloud dynamique

Sommaire

Dans le chapitre précédent, nous avons présenté notre modèle de description de SLAs multi-cloud enrichi par des stratégies de reconfiguration sous forme de machines à états adressant ainsi le premier objectif **O1** de cette thèse : "*Modélisation de SLA multi-cloud dynamique*". Les SLAs multi-cloud dynamique étant définis manuellement par un architecte cloud, ces derniers peuvent être sujets à des erreurs ou des incohérences qui pourraient entraîner une violation du SLA [70]. Ainsi, pour pallier ces incohérences et adresser notre second objectif **O2.1** de "*validation pré-mise en œuvre de SLA multi-cloud dynamique*", nous proposons une méthode de validation de cohérence pré-mise en œuvre automatisée de SLA multi-cloud dynamique. Nous considérons un SLA multi-cloud dynamique comme cohérent lorsque ses éléments (Global-SLA, Sub-SLAs et stratégies de reconfiguration) servent un objectif commun en suivant des règles non-contradictoires.

En d'autres termes, cela signifie que les global-SLOs doivent être réalisables avec les sub-SLOs données. Par exemple, un global-SLO défini sur la disponibilité à 99.999% ne peut pas être garanti avec des sub-SLOs à 80%. De même, que les stratégies de reconfiguration doivent être en adéquation avec les sub-SLOs et être déclenchées sur des métriques pertinentes pour les sub-SLOs considérés. Dans cette optique, ce chapitre se consacre à la validation pré-mise en œuvre de la cohérence entre les trois éléments constituant notre modèle de description de SLA multi-cloud dynamique : (*i*) les Global-SLAs, (*ii*) les Sub-SLAs et (*iii*) les stratégies de reconfiguration. Cette validation de cohérence sera basée sur l'adaptation des méthodes d'agrégation de SLA [88] proposées dans le cadre de SLA hiérarchique [36], ainsi que sur une méthode de traduction de SLA (*angl. SLA translation*) [61].

Le reste de ce chapitre est organisé comme suit. Dans la section 4.1, nous identifions les dépendances existantes entre les éléments d'un SLA multi-cloud dynamique. Dans la section 4.2, nous présentons notre méthode de validation de cohérence entre ces éléments. Enfin, dans la section 4.3, nous présentons la mise en œuvre et l'évaluation de notre approche de validation de cohérence.

4.1 Dépendances entre éléments de SLA multi-cloud dynamique

Dans cette section, nous présentons les dépendances entre les différents éléments d'un SLA multi-cloud dynamique. Comme présenté dans notre exemple de motivation (Section 1.4), les différents composants d'un système multi-cloud doivent fonctionner de concert pour garantir un certain niveau de service à l'utilisateur final. Cela induit, de fait, une dépendance entre les différents composants de ce système. En effet, la violation d'un sub-SLA pourra entraîner la violation d'autres sub-SLAs ou du global-SLA. Également, des stratégies de reconfiguration incohérentes avec les SLOs pourront entraîner des violations de sub-SLA. Nous identifions d'abord la dépendance entre global-SLOs et sub-SLOs (Section 4.1.1), puis, la dépendance entre sub-SLO et stratégie de reconfiguration (Section 4.1.2).

4.1.1 Global-SLOs et Sub-SLOs

Dans un système multi-cloud, chaque composant assume simultanément les rôles de client et de fournisseur [37], comme présenté dans notre exemple de motivation (Section 1.4). D'une part, ils fonctionnent en tant que fournisseur en offrant un service aux autres composants, comme le composant *Auth* proposant un service d'authentification, par exemple. D'autre part, ils agissent comme client en utilisant les services des autres composants, par exemple le composant UI utilise les services du composant *Auth* pour authentifier les utilisateurs.

Une violation du SLO de disponibilité du composant *Auth* entraînera donc une violation du SLO de disponibilité du composant UI et éventuellement du SLO de disponibilité du système global. En d'autres termes, une violation du sub-SLA d'un

composant pourra entraîner des violations au niveau global et/ou dans les autres composants. Nous pouvons alors conclure que : **Les global-SLOs dépendent des sub-SLOs.** Par conséquent, un global-SLO de 99,999% en termes de disponibilité ne peut pas être atteint lorsqu'il est lié à des sub-SLOs de valeur inférieure à 99,9999%. Nous définissons alors ces derniers comme incohérents.

Comme illustré dans la Figure 4.1, le SLO de disponibilité globale du système multi-cloud devra être cohérent avec les objectifs de disponibilité associés aux 3 composants $UI, Auth, Stor$ pour ne pas être violé pendant la phase de mise en œuvre. En effet, pour éviter d'entraîner une violation des SLOs, nous devons nous assurer que ces SLOs sont cohérents entre eux.

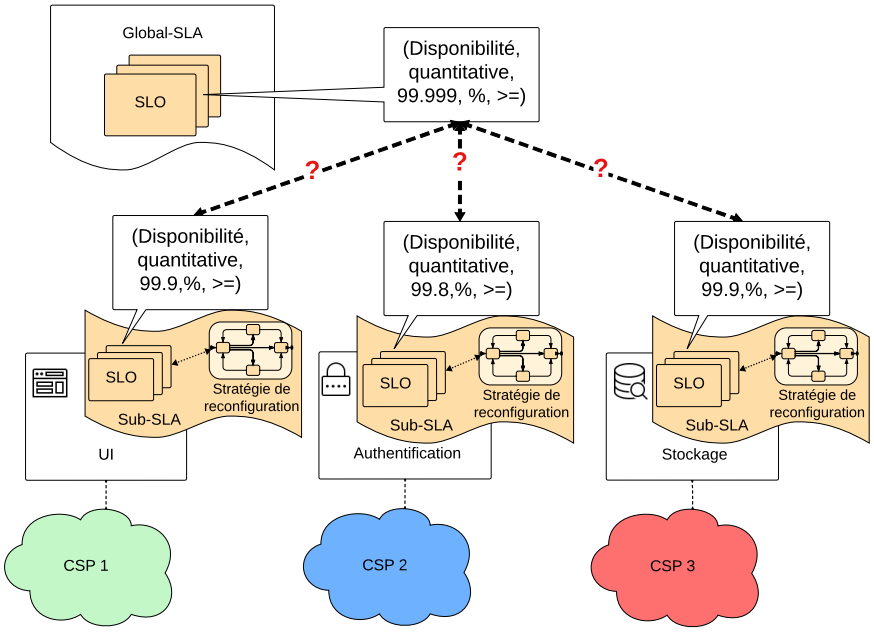


FIGURE 4.1 – Dépendance entre global-SLO et sub-SLO

La dépendance entre les global-SLOs et les sub-SLOs est étudiée dans les travaux liés aux méthodes d'agrégation de SLA [88]. Ces méthodes permettent d'agréger plusieurs objectifs (sub-SLOs) en un seul objectif global (global-SLO). Dans notre travail, nous utilisons donc une adaptation de ces méthodes pour valider la cohérence entre les global-SLOs et les sub-SLOs présenté dans la section 4.2.2.

4.1.2 Sub-SLOs et stratégies de reconfiguration

Les stratégies de reconfiguration (Section 3.2) permettent d'adapter les composants d'un système multi-cloud pour maintenir un SLA défini et prévenir les violations de sub-SLOs, malgré une charge de travail fluctuante. Cependant, comme illustré dans la Figure 4.2, il est crucial que ces stratégies soient correctement alignées avec les SLOs des composants auxquels elles s'appliquent, faute de quoi elles pourraient s'avérer inefficaces ou même provoquer des violations de ces derniers.

Par exemple, considérons un SLO de *temps de réponse* ou de *débit* pour un composant particulièrement consommateur en CPU. Si on associe ce dernier à une

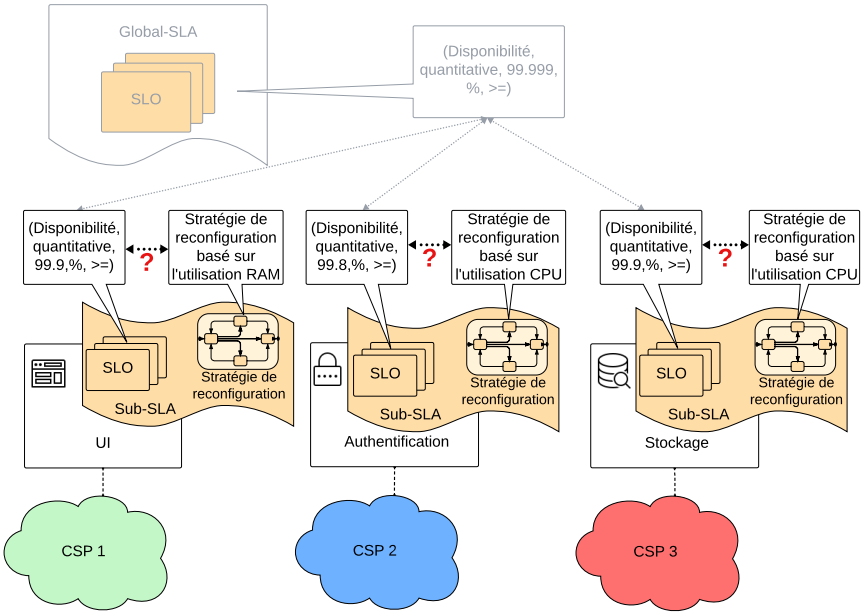


FIGURE 4.2 – Dépendance entre sub-SLO et stratégies de reconfiguration

stratégie de reconfiguration se déclenchant sur la consommation de mémoire (RAM) alors cette stratégie ne sera pas pertinente et pourra mener à une violation de SLO.

Pour approfondir cet exemple, nous reprenons l'expérimentation présentée par Rampérez et al. dans [79] qui s'intéresse aux brokers de messagerie et notamment les 4 suivants : *RabbitMQ*[1], *ActiveMQ*[2], *E-SilboPS* [93] et *BE-Tree* [81]. Le SLO étudié ici est le débit *(angl., throughput)* de notifications par seconde. L'expérimentation montre que *RabbitMQ* et *ActiveMQ* sont limités par la RAM et que *E-SilboPS* et *BE-Tree* sont limités par le CPU. Un SLO de débit de notifications par seconde pour *RabbitMQ* et *ActiveMQ* sera traduit en un paramètre opérationnel de RAM. Quand pour *E-SilboPS* et *BE-Tree*, ce dernier sera traduit en un paramètre opérationnel de CPU. Dans notre contexte, nous considérons donc comme cohérent :

— un SLO de débit de notification par seconde pour *RabbitMQ* ou *ActiveMQ*, avec une stratégie de reconfiguration déclenchée sur l'utilisation de RAM

— un SLO de débit de notification par seconde pour *E-SilboPS* ou *BE-Tree*, avec une stratégie de reconfiguration déclenchée sur l'utilisation de CPU

Cette expérimentation illustre que la définition de stratégie de reconfiguration doit être cohérente avec le SLO défini pour lui permettre d'être efficace et de ne pas induire une violation du SLO. **Les stratégies de reconfiguration dépendent des sub-SLOs auxquels elles sont associées.**

Pour anticiper ces violations et valider la cohérence entre sub-SLOs et stratégies de reconfiguration, nous considérons : (*i*) les caractéristiques des sub-SLOs (*slo.c*, Définition 3.2) et (*ii*) les événements déclencheurs (*T.E*, Définition 3.8) pour valider que l'événement qui déclenche la stratégie de reconfiguration est cohérent avec le sub-

1. `https://www.rabbitmq.com/`
2. `http://activemq.apache.org/`

SLO. Toutefois, ces éléments ne peuvent pas être validés en l'état, car ils sont à deux niveaux d'abstraction différents : *fonctionnels* pour les caractéristiques des sub-SLOs et *opérationnels* pour les événements déclencheurs des stratégies de reconfiguration.

Nous proposons donc de tirer parti du principe de traduction de SLA [61] qui permet de traduire les SLOs définis au niveau fonctionnel (par exemple : Disponibilité, Débit), proche des utilisateurs, en paramètres opérationnels (par exemple : RAM, CPU, I/O), proche des composants du système multi-cloud. En nous basant sur une technique de traduction de SLA, nous présentons dans la section 4.2.3 de ce chapitre notre approche de validation de SLA multi-cloud dynamique.

4.2 Validation de la cohérence de SLA multi-cloud dynamique

Dans cette section, nous exposons notre approche de validation de cohérence des composants de SLA multi-cloud dynamique. Dans un premier temps (Section 4.2.1), nous présentons un aperçu de notre approche de validation de la cohérence dans sa globalité. Dans un second temps, nous présentons les 2 phases de cette méthode de validation : entre global-SLOs et sub-SLOs (Section 4.2.2) puis entre sub-SLOs et stratégies de reconfiguration (Section 4.2.3). À noter, qu'il n'est pas nécessaire d'ajouter une validation entre les global-SLOs et les stratégies de reconfiguration, car cette dernière est déjà faite à partir de celle entre le global-SLO et le sub-SLO.

4.2.1 Approche de validation de la cohérence

Dans cette section, nous présentons un aperçu de notre méthode de validation illustrée dans la Figure 4.3. On retrouve dans cette figure le système multi-cloud de notre exemple de motivation avec son SLA multi-cloud dynamique associé. Notre méthode se décompose en 2 phases : ① la validation entre les global-SLOs et les sub-SLOs et ② la validation entre les sub-SLOs et les stratégies de reconfiguration. La première phase de validation est présentée en détail dans la section 4.2.2 et la seconde dans la section 4.2.3.

4.2.2 Global-SLOs et Sub-SLOs : Agrégation de SLOs

Dans la section 4.1.1, nous avons identifié que : ***Les global-SLOs dépendent des sub-SLOs.*** Dans cette section, nous nous intéressons à la validation de cohérence entre global-SLOs et sub-SLOs en utilisant une méthode d'agrégation de SLOs, méthode que nous détaillerons dans la suite de cette section. Pour effectuer cette validation, il est nécessaire de valider que les global-SLOs peuvent être respectées par les sub-SLOs (Validation descendante) et que les sub-SLOs peuvent être respectés par les global-SLOs (Validation ascendante). Étant donné que les global-SLOs et les sub-SLOs sont établis avec une importance équivalente, la double validation, ascendante et descendante, est essentielle pour repérer les SLOs susceptibles d'être incohérents, tout en évitant de privilégier les global-SLOs au

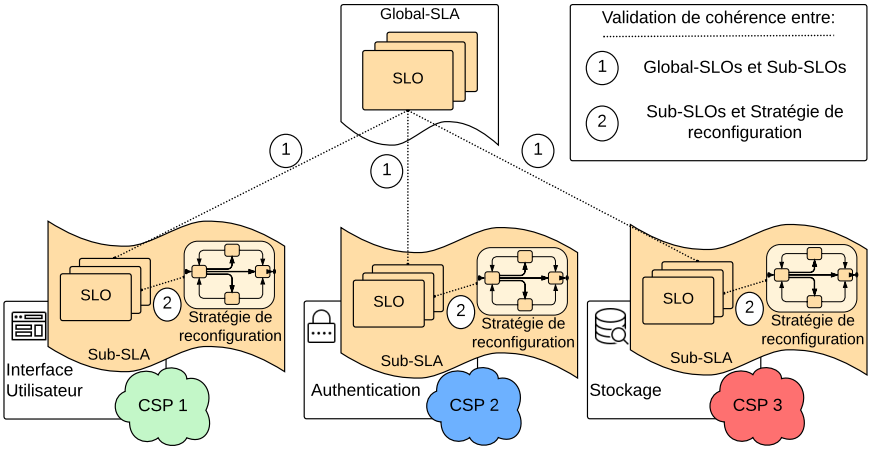


FIGURE 4.3 – Aperçu de validation de cohérence de SLA multi-cloud dynamique

détriment des sub-SLOs et inversement. Notre approche de validation de cohérence entre global et sub-SLOs est illustrée dans la Figure 4.4.

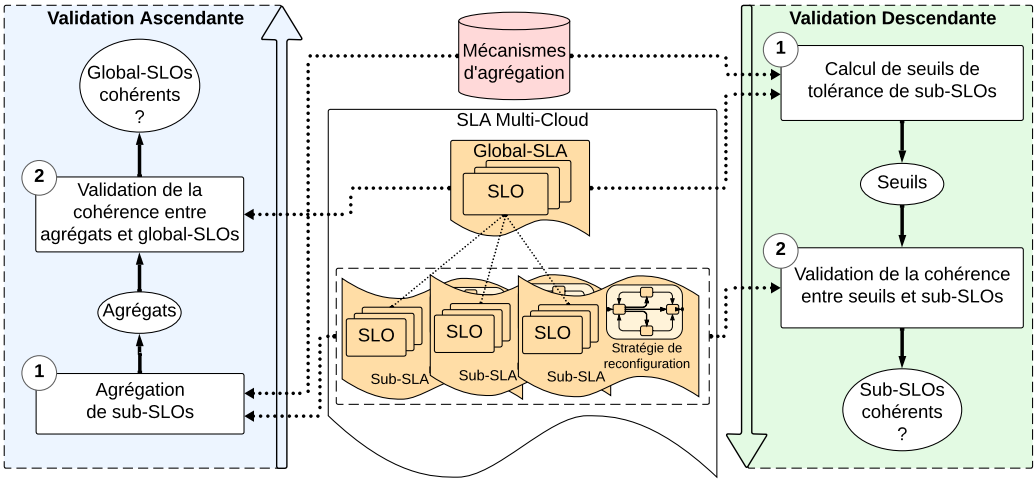


FIGURE 4.4 – Validation de la cohérence entre global-SLOs et sub-SLOs

Dans la suite de cette section, nous présentons tout d'abord les mécanismes d'agrégation de SLOs (Section 4.2.2.1), puis la validation de cohérence ascendante (Section 4.2.2.2) et pour finir, la validation de cohérence descendante (Section 4.2.2.3).

4.2.2.1 Mécanismes d'agrégation de SLOs

L'agrégation consiste en l'assemblage de plusieurs SLOs en un seul SLO au moyen d'un opérateur. L'agrégation de SLOs s'effectue de manière différente en fonction de la caractéristique du SLO agrégé. En effet, chaque SLO (Définition 3.2) représente des caractéristiques différentes en fonction de l'objectif qu'il représente (par exemple : disponibilité, temps de réponse, coût). Par exemple, l'agrégation d'un

objectif de temps de réponse (en millisecondes) diffère de celle d'un objectif de disponibilité (en pourcentage de requêtes réussies) ou de celle d'un coût (en euros). Ainsi, Ul Haq et Schikuta proposent dans [88] les opérateurs suivants pour agréger les valeurs de SLOs : `SumType`, `MaxType`, `MinType`, `Neutral`, `ANDType`, `ORType` et `XORType`. Ces différents opérateurs sont représentés avec leurs formules respectives dans la Figure 4.5.

Chaque opérateur reçoit les valeurs slo_1.v et slo_2.v comme entrées et produit en retour la valeur résultant du traitement spécifique appliqué. À titre d'illustration, l'opérateur *SumType* calcule et fournit la somme de slo_1.v et slo_2.v.

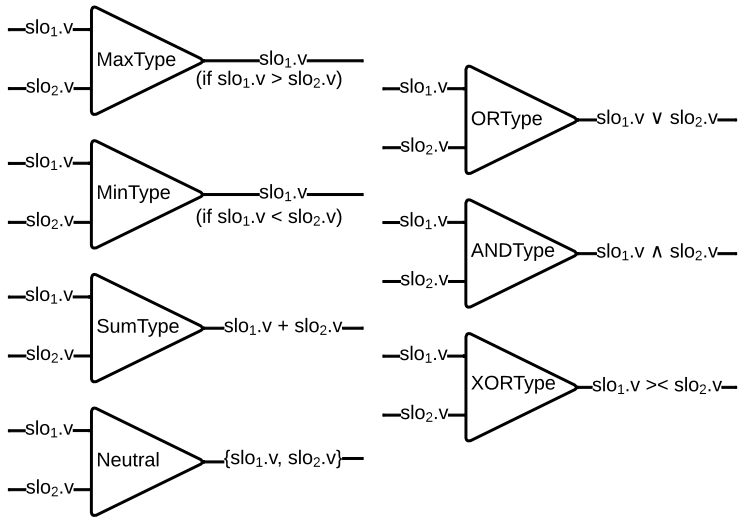


FIGURE 4.5 – Opérateurs d'agrégation de SLOs introduit dans [88]

Pour notre méthode de validation de cohérence, nous associons chaque SLOs à une méthode de calcul de seuil de tolérance et à un opérateur d'agrégation. Nous appelons ces triplets *mécanismes d'agrégation* qui permettent d'identifier pour un SLO donné la méthode de calcul des seuils de tolérance de sub-SLOs à utiliser pour la validation descendante et l'opérateur d'agrégation de sub-SLOs à utiliser pour la validation ascendante. Ces derniers sont représentés par un 3-tuple :

$$\langle \text{SLO.c}, \text{Opérateur d'agrégation}, \text{Calcul de seuil} \rangle$$

Dans notre travail, nous définissons 6 mécanismes d'agrégation correspondant à des caractéristiques de SLOs cloud répandus [8, 24, 29, 31, 49] (disponibilité, temps de réponse, coût, débit, uptime et latence) illustré dans le Tableau 4.1 [3]. Les opérateurs d'agrégation peuvent et doivent être adaptés en fonction de l'architecture cible par l'architecte cloud pour être pertinents comme pour l'objectif de Disponibilité ligne 1 du Tableau 4.1.

Dans ce mécanisme d'agrégation, l'opérateur d'agrégation `SumType` est adapté

3. Dans cette thèse, nous avons défini les mécanismes d'agrégation pour les six objectifs de niveau de service les plus utilisés. En cas de nouveau besoin, de nouveaux mécanismes pourront être définis

pour agréger le pourcentage de disponibilité, tel que proposé dans [21]. En effet, la disponibilité étant exprimée en pourcentage, il est nécessaire d'adapter l'opérateur d'agrégation SumType afin que ce dernier supporte l'agrégation de données exprimée en pourcentage. La troisième ligne illustre le mécanisme d'agrégation associé à un objectif de coût (*slo.c*). Pour la validation à partir de sub-SLOs (validation ascendante), l'opérateur d'agrégation `SumType` (colonne 2) est utilisé pour calculer un agrégat qui correspondra à une valeur atteignable pour les sub-SLOs donnés. Pour la validation à partir de global-SLOs (validation descendante), le calcul de seuil de tolérance des sub-SLOs est effectué avec $SLO.v/nb_c$ (colonne 3).

Tableau 4.1 – Mécanismes d'agrégation définis

Objectif de niveau de service	**Opérateur d'agrégation**	**Calcul de seuil**
Disponibilité	`Disponibilité_SumType`	$\frac{SLO.v+(100nb_c-100)}{nb_c}$
Temps de réponse	`MaxType`	$SLO.v$
Coût	`SumType`	$\frac{SLO.v}{nb_c}$
Débit	`MaxType`	$SLO.v$
Uptime	`ORType`	$SLO.v$
Latence	`MaxType`	$SLO.v$
Réglementation	`Neutral`	$SLO.v$

$SLO.v$: Valeur du SLO
nb_c : Nombre de composants du système multi-cloud

4.2.2.2 Validation ascendante

Pour la validation ascendante (Figure 4.4), nous prenons en entrée les sub-SLOs afin de valider les différents global-SLOs. Pour ce faire, nous agrégeons les valeurs de sub-SLOs de même type pour calculer une valeur de SLO agrégée (dénommée agrégat a) qui devra être atteignable par les valeurs associées aux sub-SLOs donnés. À noter qu'il est possible que tous les types de global-SLOs ne soient pas présents dans les sub-SLOs, car ils peuvent ne pas être applicables à certaines ressources, par exemple. Cela ne bloquera pas notre validation, mais sera signalé dans le rapport à l'architecte cloud.

Dans la suite de cette section, nous présentons l'algorithme 1 que nous proposons pour valider la cohérence entre les global-SLOs et les sub-SLOs. Cet algorithme prend en entrée un global-SLA (sla^G), un ensemble de sub-SLAs (SLA^S), le nombre de composants (nb_c) d'un SLA multi-cloud décrivant un système multi-cloud et une base de connaissance (KB) contenant les mécanismes d'agrégation. Cet algorithme est composé de 3 fonctions : *Extraction_Sub_SLOs* (Algorithme 2), *Calcul_Des_Agregats* (Algorithme 3) et *Validation_Global_SLOs* (Algorithme 4).

Algorithme 1 Validation ascendante

Entrée KB : Base de connaissance
sla^G : Global-SLA du système multi-cloud
SLA^S : Ensemble de Sub-SLAs du système multi-cloud
nb_c : Nombre de composants du système multi-cloud
Sortie $SLO_{incoherent}$: Ensemble de $SLOs$ incohérents
▷ Extraction des sub-SLOs à agréger dans chacun des sub-SLAs
1: $SLO_{a\_agreger} \leftarrow$ **Extraction_Sub_SLOs**(SLA^S)
▷ Agrégation des SLOs extraits de même caractéristique
2: $AGT \leftarrow$ **Calcul_Des_agregats**(KB, $SLO_{a\_agreger}$)
▷ Identification des SLOs incohérents issue du Global-SLA
3: $SLO_{incoherent} \leftarrow$ **Validation_Global_SLOs**(sla^G, AGT)
Return $SLO_{incoherent}$

La première fonction *Extraction_Sub_SLOs* (Algorithme 2) permets l'extraction des sub-SLOs à agréger depuis un ensemble de sub-SLAs. Nous commençons par initialiser une table de hachage (ligne 1) pour stocker les SLOs à agréger ($SLO_{a\_agreger}$). Ensuite, nous parcourons les sub-SLOs (slo^S) présents dans chaque sub-SLAs (sla^S) de l'ensemble de sub-SLA (SLA^S) puis nous les ajoutons dans la table de hachage $SLO_{a\_agreger}$ avec pour clé la caractéristique du SLO ($SLO.c$) (lignes 2 à 6). Puis, nous retournons la table de hachage $SLO_{a\_agreger}$.

Algorithme 2 Extraction des sub-SLOs à agréger

Fonction Extraction_Sub_SLOs(SLA^S)
Entrée SLA^S : Ensemble de Sub-SLAs du système multi-cloud
Sortie $SLO_{a\_agreger}$: Hashmap des sub-SLOs à agréger
1: $SLO_{a\_agreger}$: Hashmap avec pour clé une caractéristique de sub-SLO ($slo^S.c$) et pour valeur une liste de valeur de SLO à agréger ($slo^S.v$)
▷ Parcours des sub-SLOs présent dans chacun des sub-SLAs
2: **for each** $sla^S \in SLA^S$ **do**
3: **for each** $slo^S \in sla^S.O$ **do**
▷ Ajout du SLO à la liste des SLOs de même caractéristique à agréger
4: $SLO_{a\_agreger}.put(slo^S.c, slo^S.v)$
5: **end for**
6: **end for**
Return $SLO_{a\_agreger}$

La seconde fonction *Calcul_Des_agregats* (Algorithme 3) permets le calcul des agrégats de sub-SLOs à partir d'une Knowledge Base et des sub-SLOs à agréger. Nous initialisons une table de hachage pour stocker les agrégats calculés (AGT) (ligne 1). Puis, nous parcourons les listes de SLOs à agréger à partir des clés de $SLO_{a\_agreger}$, pour chacune, nous agrégeons la liste associée en fonction de l'opérateur ($operateurAgregation$) récupéré depuis la base de connaissance (KB) et insérons ces agrégats associés à leurs caractéristiques de SLO ($SLO.c$) dans la table de hachage AGT (lignes 2 à 6). Pour finir, nous retournons la table de hachage AGT.

Algorithme 3 Calcul des agrégats

Fonction Calcul_Des_Agregats(KB, $SLO_{a\_agreger}$)
Entrée KB : Base de connaissance
$SLO_{a\_agreger}$: Hashmap des sub-SLOs à agréger
Sortie AGT : Hashmap des agrégats calculés

1: AGT : Hashmap avec pour clé une caractéristique de sub-SLO ($slo^S.c$) et pour valeur l'agrégat calculé (a)
▷ Parcours des SLOs extraits par caractéristique à agréger
2: **for each** $slo.c \in SLO_{a\_agreger}.keys$ **do**
▷ Extraction de l'opérateur d'agrégation depuis la base de connaissance
3: $operateurAgregation \leftarrow$ KB.get($slo.c$)
▷ Agrégation des sub-SLOs d'une même caractéristique
4: $a \leftarrow operateurAgregation(SLO_{a\_agreger}.get(slo.c))$
▷ Ajout de l'agrégat de slo calculé dans la liste des agrégats
5: $AGT.put(slo.c, a)$
6: **end for**
Return AGT

La troisième fonction *Validation_Global_SLOs* (Algorithme 4) permets d'identifier les global-SLOs qui seraient incohérents avec les agrégats. Pour ce faire, nous initialisons un ensemble vide (ligne 1) pour stocker les SLOs incohérents ($SLO_{incoherent}$). Nous parcourons les global-SLOs et la liste d'agrégats (lignes 2 à 13), puis, nous vérifions les agrégats de même caractéristique que les global-SLOs en comparant en fonction de l'opérateur du global-SLO et du type du SLO (quantitatif ou qualitatif) s'il doit être plus grand, plus petit ou équivalent. Si le global SLO est considéré comme incohérent, il est alors ajouté à l'ensemble des SLOs incohérents qui sera retourné à l'architecte. Si l'ensemble retourné est vide alors les SLAs en entrées sont considérés comme valides.

Algorithme 4 Validation des global-SLOs

Fonction Validation_Global_SLOs(sla^G, AGT)
Entrée sla^G : Global-SLA du système multi-cloud
AGT : Hashmap des agrégats calculés
Sortie $SLO_{incoherent}$: Ensemble de $SLOs$ incohérents
▷ Initialisation de l'ensemble vide des SLOs incoherent
1: $SLO_{incoherent} \leftarrow \emptyset$
▷ Identification des SLOs incohérents issue du Global-SLA
2: **for each** $slo^G \in sla^G.O$ **do**
▷ Extraction de l'agrégat en fonction de la caractéristique des global-SLOs
3: $agt \leftarrow AGT.get(slo^G.c)$
4: **if** ($slo^G.t ==' Quantitatif'$) **then**
▷ Comparaison de l'agrégat en fonction de l'opérateur de comparaison slo.op
5: **if** ($slo^G.op ==' <=' $ et $slo^G.v > agt.v$) **or**
($slo^G.op ==' >=' $ et $slo^G.v < agt.v$) **then**
▷ Ajoute le SLO incohérent dans la liste de slo incohérent $SLO_{incoherent}$
6: $SLO_{incoherent} \leftarrow SLO_{incoherent} \cup slo^G$
7: **end if**
8: **else if** ($slo^G.t ==' Qualitatif'$) **then**
▷ Comparaison de l'agrégat
9: **if** ($slo^G.v \neq agt.v$) **then**
▷ Ajoute le SLO incohérent dans une liste
10: $SLO_{incoherent} \leftarrow SLO_{incoherent} \cup slo^G$
11: **end if**
12: **end if**
13: **end for**
Return $SLO_{incoherent}$

Un exemple détaillé de cet algorithme est présenté dans la section 4.2.2.4 puis sa mise en œuvre et évaluation sont abordées dans la section 4.3.

4.2.2.3 Validation descendante

Pour la validation descendante (Figure 4.4), nous prenons en entrée les global-SLOs afin de valider la cohérence avec les différents sub-SLOs définis. Pour ce faire, nous validons la compatibilité des types d'objectifs (par exemple : disponibilité, coût) et de leur valeur (par exemple : $\geq 99\%$) avec un seuil de tolérance calculé à partir des global-SLOs. Les sub-SLOs devront respecter ce seuil de tolérance calculé pour être définis comme cohérents.

Dans la suite de cette section, nous présentons l'algorithme 5 que nous proposons pour valider la cohérence entre les global-SLOs et les sub-SLOs. Cet algorithme prend en entrée un global-SLA (sla^G), un ensemble de sub-SLAs (SLA^S), le nombre de composants (nb_c) d'un SLA multi-cloud décrivant un système multi-cloud et une base de connaissance (KB) contenant les mécanismes d'agrégation. Cet algorithme est composé de 2 fonctions : *Analyse_Validation_Descendante* (Algorithme 6) et *Identification_SLO_incoherent* (Algorithme 7).

Algorithme 5 Validation descendante

Entrée KB : Base de connaissance
sla^G : Global-SLA du système multi-cloud
SLA^S : Ensemble de Sub-SLAs du système multi-cloud
nb_c : Nombre de composants du système multi-cloud
Sortie $SLO_{incoherent}$: Ensemble de $SLOs$ incohérents
▷ Analyse des global-SLOs et calcul des seuils de tolérance
1: $SEUILS \leftarrow$ **Analyse_Validation_Descendante**(KB, sla^G, nb_c)
▷ Identification des sub-SLOs incohérents
2: $SLO_{incoherent} \leftarrow$ **Identification_SLO_incoherent**($SEUILS$, SLA^S)
Return $SLO_{incoherent}$

La première fonction *Analyse_Validation_Descendante* (Algorithme 6) permets l'analyse des global-SLOs et le calcul des seuils de tolérance. Nous commençons par initialiser une table de hachage (ligne 1) pour stocker les seuils de tolérance ($SEUILS$). Ensuite, nous parcourons les global-SLOs (slo^G) présents dans le global-SLA (sla^G) (lignes 2 à 6). Pour chacun d'eux, nous extrayons la méthode de calcul (*calculSeuil*) à partir de la caractéristique du SLO ($SLO.c$) depuis la base de connaissance (KB) (ligne 3), ce qui nous permet de calculer le seuil de tolérance (s) (ligne 4) que nous ajoutons ensuite dans la table de hachage contenant les seuils ($SEUILS$) avec pour clé la caractéristique du SLO ($SLO.c$) (ligne 5). Enfin, nous retournons la table de hachage $SEUILS$.

Algorithme 6 Analyse et calcul des seuils de tolérance

Fonction Analyse_Validation_Descendante(KB, sla^G, nb_c)
Entrée KB : Base de connaissance
sla^G : Global-SLA du système multi-cloud
nb_c : Nombre de composants du système multi-cloud
Sortie $SEUILS$: Hashmap des seuils calculés
1: $SEUILS \leftarrow$ Hashmap avec pour clé une caractéristique de sub-SLO ($slo.c$) et pour valeur le seuil calculé (s)
▷ Parcours des global-SLOs du global-SLA
2: **for each** $slo^G \in sla^G.O$ **do**
▷ Récupération de la méthode de calcul de seuil de tolérance adéquate depuis la base de connaissance
3: $calculSeuil \leftarrow$ KB.get($slo^G.c$)
▷ Calcul du seuil de tolérance s des sub-SLOs
4: $s \leftarrow calculSeuil(slo^G.v, nb_c)$
▷ Ajout du couple (caractéristique de SLO, seuil de tolérance) à $SEUILS$
5: $SEUILS.put(slo^G.c, s)$
6: **end for**
Return $SEUILS$

Pour la seconde fonction *Identification_SLO_incoherent* (Algorithme 7), nous cherchons à identifier les sub-SLOs qui seraient incohérents avec les seuils de tolérance. Pour ce faire, nous initialisons un ensemble vide (ligne 1) pour stocker les SLOs incohérents ($SLO_{incoherent}$). Nous parcourons les sub-SLOs présents dans chaque sub-SLA de l'ensemble des sub-SLAs (lignes 2 à 15). Nous extrayons le seuil de tolérance (*seuil*) en fonction de la caractéristique du sub-SLO (ligne 4) et nous

comparons la valeur du sub-SLO avec ce seuil (lignes 5 à 11). Si le sub-SLO est considéré comme incohérent, il est ajouté à l'ensemble des SLOs incohérents. Enfin, nous retournons l'ensemble des SLOs incohérents.

Algorithme 7 Identification des sub-SLOs incohérents

Fonction Identification_SLO_incoherent($SEUILS$, SLA^S)
Entrée $SEUILS$: Hashmap des seuils calculés
SLA^S : Ensemble de Sub-SLAs du système multi-cloud
Sortie $SLO_{incoherent}$: Ensemble de $SLOs$ incohérents
▷ Initialisation de l'ensemble vide des SLOs incohérent
1: $SLO_{incoherent} \leftarrow \emptyset$
▷ Parcours des sub-SLOs présents dans chacun des sub-SLAs
2: **for each** $sla^S \in SLA^S$ **do**
3: **for each** $slo^S \in sla^S.O$ **do**
▷ Extraction du seuil de tolérance en fonction de la caractéristique des sub-SLOs
4: $seuil \leftarrow SEUILS.get(slo^S.c)$
5: **if** ($slo^S.t ==' Quantitatif'$) **then**
▷ Comparaison du seuil en fonction de l'opérateur de comparaison slo.op
6: **if** ($slo^S.op ==' <='$ et $slo^S.v > seuil$) **or**
($slo^S.op ==' >='$ et $slo^S.v < seuil$) **then**
▷ Ajout du SLO incohérent dans la liste de slo incohérents $SLO_{incoherent}$
7: $SLO_{incoherent} \leftarrow SLO_{incoherent} \cup slo^S$
8: **end if**
9: **else if** ($slo^S.t ==' Qualitatif'$) **then**
▷ Comparaison du seuil de tolérance
10: **if** ($slo^S.v \neq seuil$) **then**
▷ Ajout du SLO incohérent dans une liste
11: $SLO_{incoherent} \leftarrow SLO_{incoherent} \cup slo^S$
12: **end if**
13: **end if**
14: **end for**
15: **end for**
Return $SLO_{incoherent}$

Un exemple détaillé de cet algorithme est présenté dans la section 4.2.2.4 puis sa mise en œuvre et évaluation sont abordées dans la section 4.3.

4.2.2.4 Exemple

Pour illustrer notre approche de validation de la cohérence entre Global-SLO et Sub-SLO, nous considérons l'exemple du SLA^{Global} suivant composés des global-SLOs $Gslo_1$, $Gslo_2$ et $Gslo_3$:

$SLA^{Global} = \langle Gslo_1, Gslo_2, Gslo_3 \rangle$ où :
- $Gslo_1$ = (Temps de réponse, *quantitative*, 18,ms, $\leq$),
- $Gslo_2$ = (Disponibilité, *quantitative*, 99.999,%, $\geq$) et
- $Gslo_3$ = (Coût, *quantitative*, 23,€/h, $\leq$)

Nous considérons les sub-SLAs suivant : SLA^{Sub}_{UI}, SLA^{Sub}_{Auth} et SLA^{Sub}_{Stor}. SLA^{Sub}_{UI} représentant le sub-SLA du composant UI et est défini comme suit :

$SLA^{Sub}_{UI} = \langle UI, (CSP_1, slo_1, slo_2, slo_3)\rangle$ où :
- Slo_1 = (Temps de réponse, *quantitative*, 18, ms, $\leq$),
- Slo_2 = (Disponibilité, *quantitative*, 99.9,%, $\geq$) et
- Slo_3 = (Coût, *quantitative*, 12,€/h, $\leq$).

SLA^{Sub}_{Auth} représentant le `sub-SLA` du composant *Auth* et est défini comme suit :

$SLA^{Sub}_{Auth} = \langle Auth, (CSP_2, slo_1, slo_2)\rangle$ où :
- Slo_1 = (Temps de réponse, *quantitative*, 14,ms, $\leq$),
- Slo_2 = (Disponibilité, *quantitative*, 99.9,%, $\geq$),
- Slo_3 = (Coût, *quantitative*, 6,€/h, $\leq$) et
- Slo_4 = (Réglementation, *qualitative*, $RGPD$, ==).

SLA^{Sub}_{Stor} représentant le `sub-SLA` du composant *Storage* et est défini comme suit :

$SLA^{Sub}_{Stor} = \langle Stor, (CSP_3, slo_1, slo_2, slo_3)\rangle$ où :
- Slo_1 = (Temps de réponse, *quantitative*, 14,ms,$\leq$),
- Slo_2 = (Disponibilité, *quantitative*, 99.9,%, $\geq$) et
- Slo_3 = (Coût, *quantitative*, 5,€/h, $\leq$).

Validation Ascendante, visant à établir des agrégats à partir des sub-SLAs comme présenté dans l'algorithme 1. Nous établissons ces agrégats en nous basant sur les mécanismes d'agrégation décrits dans la base de connaissance (Tableau 4.1), en particulier les opérateurs d'agrégation liés à chacun des différents objectifs.

Pour la première partie de l'algorithme 1, nous commençons par l'extraction puis l'agrégation des SLOs en fonction des opérateurs d'agrégation de la ligne 3 à 13. Pour calculer l'agrégat ($a_{tps\_reponse}$) associé aux Slo_1 des SLA^{Sub}_{UI}, SLA^{Sub}_{Auth} et SLA^{Sub}_{Stor} ayant pour caractéristique le Temps de réponse, nous utilisons l'opérateur d'agrégation associé au Temps de réponse $MaxType$ comme indiqué dans la base de connaissance.

$$a_{tps\_reponse} = MaxType(18, 14, 14) = 18\ ms$$

Pour calculer l'agrégat ($a_{disponibilite}$) associé aux Slo_2 des SLA^{Sub}_{UI}, SLA^{Sub}_{Auth} et SLA^{Sub}_{Stor} ayant pour caractéristique la disponibilité, nous utilisons l'opérateur d'agrégation associé à la disponibilité $AvailabilitySumType$ comme indiqué dans la base de connaissance.

$$a_{disponibilite} = AvailabilitySumType(99.9, 99.9, 99.9) =$$
$$100 - ((100 - 99.9) + (100 - 99.9) + (100 - 99.9)) = 99.7\%$$

Pour calculer l'agrégat (a_{cout}) associé aux Slo_3 des SLA^{Sub}_{UI}, SLA^{Sub}_{Auth} et SLA^{Sub}_{Stor} ayant pour caractéristique le coût, nous utilisons l'opérateur d'agrégation associé au coût $SumType$ comme indiqué dans la base de connaissance.

$$a_{cout} = SumType(12, 6, 5) = 23€/h$$

Les agrégats obtenus suite à l'agrégation des sub-SLOs sont définis comme suit dans la liste AGT :

$AGT = \langle a_{tps\_reponse}, a_{disponibilite}, a_{cout} \rangle$ où :
- $a_{tps\_reponse}$ = (Temps de réponse, 18),
- $a_{disponibilite}$ = (Disponibilité, 99.7) et
- a_{cout} = (Coût, 23)

Ensuite, nous comparons les agrégats AGT avec les global-SLOs pour valider leur cohérence que nous retrouvons dans l'algorithme 1 de la ligne 14 à 25. Nous pouvons voir que les G_{slo1} ($G_{slo1}.v$: $18ms$ <= $a_{tps\_reponse}$: $18ms$) et G_{slo3} ($G_{slo3}.v$: 23\$ >= a_{cout} : 23\$) sont cohérents avec les agrégats et le G_{slo2} ($G_{slo2}.v$: 99.999 <= $a_{disponibilite}$: 99.7) est incohérent. Pour finir, nous retournons le G_{slo2} considéré comme incohérent dans la liste de SLOs incohérent ($SLO_{incoherent}$).

Validation Descendante, visant à établir des seuils à partir des SLOs globaux, comme présenté dans l'algorithme 2. Nous établissons ces seuils en exploitant les mécanismes d'agrégation décrits dans la base de connaissance (Tableau 4.1), les calculs de seuils liés à chacun des différents objectifs de la ligne 3 à 7.

Pour calculer le seuil de tolérance ($S_{tps\_reponse}$) associé à $Gslo_1$ ayant pour caractéristique le Temps de réponse, nous utilisons le calcul de seuil de tolérance qui nous permet d'obtenir le seuil de tolérance $S_{tps\_reponse}$ comme indiqué dans la base de connaissance.

$$s_{tps\_reponse} = SLO.v = 18$$

Pour calculer le seuil de tolérance (S_{dispo}) associé à $Gslo_2$ ayant pour caractéristique la disponibilité, nous utilisons le calcul de seuil de tolérance associé dans son mécanisme d'agrégation comme indiqué dans la base de connaissance.

$$s_{dispo} = \frac{SLO.v + (100 * nb_c - 100)}{nb_c}$$
$$= \frac{99.999 + (100 * 3 - 100)}{3}$$
$$= 99.9997$$

Pour calculer le seuil de tolérance (S_{cout}) associé à $Gslo_3$ ayant pour caractéristique le coût, nous utilisons le calcul de seuil de tolérance associé dans son mécanisme d'agrégation comme indiqué dans la base de connaissance.

$$s_{cout} = \frac{SLO.v}{nb_c} = \frac{23}{3} = 7.7$$

Ce qui nous permet d'obtenir la liste de seuils suivants :

$SEUILS = \langle S_{tps\_reponse}, S_{dispo}, S_{cout} \rangle$ où :
- $s_{tps\_reponse}$ = (Temps de réponse, 18),
- s_{dispo} = (Disponibilité, 99.9997) et
- s_{cout} = (Coût, 7.7)

Ensuite, nous comparons les seuils avec les sub-SLOs pour valider leur cohérence que nous retrouvons dans l'algorithme 2 de la ligne 8 à 21. Nous pouvons voir que les sub-SLOs suivants : $SLA^{Sub}_{UI}.Slo_2$ ($SLA^{Sub}_{UI}.Slo_2.v$: 99.9 >= s_{dispo} : 99.9997), $SLA^{Sub}_{Auth}.Slo_2$ ($SLA^{Sub}_{Auth}.Slo_2.v$: 99.9 >= s_{dispo} : 99.9997), $SLA^{Sub}_{Stor}.Slo_2$ ($SLA^{Sub}_{Stor}.Slo_2.v$: 99.9 >= s_{dispo} : 99.9997) et $SLA^{Sub}_{UI}.Slo_3$ ($SLA^{Sub}_{UI}.Slo_3.v$: 12 <= s_{cout} : 7.7) ne sont pas cohérent. Pour finir, nous retournons les Sub_{slo} $\langle SLA^{Sub}_{UI}.Slo_2, SLA^{Sub}_{Auth}.Slo_2, SLA^{Sub}_{Stor}.Slo_2,, SLA^{Sub}_{UI}.Slo_3 \rangle$ considérés comme incohérents dans la liste de SLOs incohérent ($SLO_{incoherent}$).

Cette double validation nous a permis d'identifier, avec une importance équivalente, les incohérences des global-SLOs par rapport aux sub-SLOs et vice versa. Cela est crucial pour détecter toutes les incohérences sans privilégier l'un ou l'autre, assurant ainsi une identification complète à travers un SLA multi-cloud.

4.2.3 Sub-SLOs et stratégies de reconfiguration : Traduction de SLAs

Dans la section 4.1.2, nous avons identifié que : ***Les stratégies de reconfiguration sont dépendantes des sub-SLOs.*** Ainsi, cela induit le besoin de valider aussi la cohérence entre les sub-SLOs et les stratégies de reconfiguration.

Dans cette section, nous présentons l'algorithme 8 basé sur la traduction de SLA [61] que nous proposons pour valider la cohérence entre les sub-SLOs et les stratégies de reconfiguration. Cette traduction devra être définie par un expert ou nécessitera la mise en place d'un protocole expérimental permettant d'observer

le lien entre paramètres fonctionnels (caractéristique de slo (slo.c)) et paramètres opérationnels (événement déclencheur de transition (T.E.)). En effet, l'association de *slo* de haut niveau, proche des utilisateurs, aux objectifs de bas niveau, proche des composants du système multi-cloud, n'est pas évidente.

Nous représentons cette connaissance par le quadruplet :

$$\langle \text{SLO}, \text{Type d'application}, \text{Application}, \text{Paramètre opérationnel} \rangle$$

indique qu'un SLO est associé à un paramètre opérationnel en fonction d'une application et de son type. Un exemple est représenté dans le Tableau 4.2.

TABLEAU 4.2 – Extrait de la base de connaissance pour la traduction de SLOs en paramètres opérationnel

SLO	**Type d'application**	**Application**	**Paramètre opérationnel**
($Debit$, $quantitatif$, 30000, $notifications/s$, $\geq$)	MessageQueue	RabbitMQ	(CPU_usage, 80%, $\leq$)
($Debit$, $quantitatif$, 15000, $notifications/s$, $\geq$)	MessageQueue	ActiveMQ	(Memory_usage, 80%, $\leq$)
($Debit$, $quantitatif$, 30000, $notifications/s$, $\geq$)	MessageQueue	E-SilboPS	(CPU_usage, 80%, $\leq$)
	...		

En effet, comme présenté dans l'exemple de *RabbitMQ* de la section 4.1.2 les sub-SLOs sont des exigences métier non fonctionnelles (Débit) qui ne peuvent pas être simplement traduites en exigences fonctionnelles (CPU) sans connaissance métier préalable. C'est pourquoi cette traduction se base sur des connaissances métier représentées dans une base de connaissance pour identifier cette association.

L'approche d'évaluation de la cohérence des sub-SLOs et des stratégies de reconfiguration est présentée dans l'algorithme 8. Cet algorithme prend en entrée un sub-SLA et une base de connaissance (KB). Nous commençons par parcourir les sub-SLOs du sub-SLA (ligne 1) pour chacun d'eux nous extrayions le paramètre opérationnel en fonction des caractéristiques des sub-SLOs à partir de la base de connaissance (ligne 2) et comparons ces valeurs extraites avec les prédicats des événements déclencheurs des transitions (lignes 3-7). Si l'ensemble retourné est vide alors le sub-SLA et la stratégie de reconfiguration en entrée sont considérés comme cohérents. Bien que la complexité de cet algorithme soit élevée, cela ne pose pas de problème car il ne sera exécuté qu'à la création d'un SLA multi-cloud sur un ensemble restreint de sub-SLOs.

Pour illustrer la validation de cohérence entre sub-SLOs et stratégies de reconfiguration, nous considérons le sub-SLO suivant de débit suivant :

$$Slo_1 = (\text{Débit}, quantitatif, 30000, notifications/s, \geq)$$

Algorithme 8 Validation entre Sub-SLA et Stratégie de reconfiguration

Entrée sla^S : sub-SLA
KB : Base de connaissance
Sortie $T_{incoherent}$: Transitions incohérentes de la machine à état

▷ Parcours des sub-SLOs
1: **for each** $slo^s \in sla^S.O$ **do**
▷ Traduction depuis la base de connaissance
2: $OperationalParam \leftarrow KB.getOperationalParam(sla^S.r, slo^s)$
▷ Parcours des transitions
3: **for each** $t \in sla^S.SM.T$ **do**
▷ Comparaison du paramètre opérationnel avec le prédicat de l'événement déclencheur de la transition extraite
4: **if** $OperationalParam \neq t.e.p$ **then**
▷ Ajout de la transition à liste des transitions considéré comme incohérente
5: $T_{incoherent} \leftarrow T_{incoherent} \cup t$
6: **end if**
7: **end for**
8: **end for**
Return $T_{incoherent}$

En appliquant l'algorithme 8 avec en entrée Slo_1 et la base de connaissance représentée dans la table 4.8. Nous obtenons le paramètre opérationnel suivant :

$$OperationalParam = (CPU\_usage, 80\%, \leq)$$

Ce résultat présente le *paramètre opérationnel* qui traduit l'objectif de débit et que les stratégies de reconfiguration doivent respecter pour éviter la violation de sub-SLO. Ainsi, les événements déclencheurs liés à l'utilisation de CPU ne doivent pas être supérieurs à 80% car sinon l'objectif de débit de 30000 notifications par seconde pourrait ne pas être atteint. Ce paramètre est alors comparé aux événements déclencheurs des transitions des stratégies de reconfiguration associées qui permet d'identifier les transitions incohérentes et de les ajouter à la liste $SM.t_{incoherent}$.

4.3 Mise en œuvre et évaluation

Dans cette section, nous présentons les travaux d'implémentation et d'évaluation de notre approche de validation de cohérence d'un SLA multi-cloud dynamique. Tout d'abord, nous mettons en œuvre les algorithmes 1, 2 et 3 proposés dans les sections précédentes. Ensuite, nous évaluons notre approche face à différents scénarios de SLA multi-cloud dynamique.

4.3.1 Mise en œuvre de l'approche de validation

Dans cette section, nous présentons la mise en œuvre de nos approches permettant la validation de cohérence proposée dans la section 4.2. Un aperçu de cette

approche est représenté dans la Figure 4.6. Cette approche prend en entrée un SLA multi-cloud dynamique définit par un architecte cloud. Dans un premier temps, nous évaluons la cohérence entre les global-SLOs et les sub-SLOs en utilisant les algorithmes 1 et 5. Dans un second temps, la cohérence entre les sub-SLOs et les stratégies de reconfiguration à l'aide de l'algorithme 8. En sortie, notre outil retourne les éventuelles incohérences détectées à l'architecte cloud pour lui permettre de corriger ces incohérences. Ces deux algorithmes sont implémentés en Python et utilisent une base de connaissance contenant les informations nécessaires pour valider la cohérence de ces derniers.

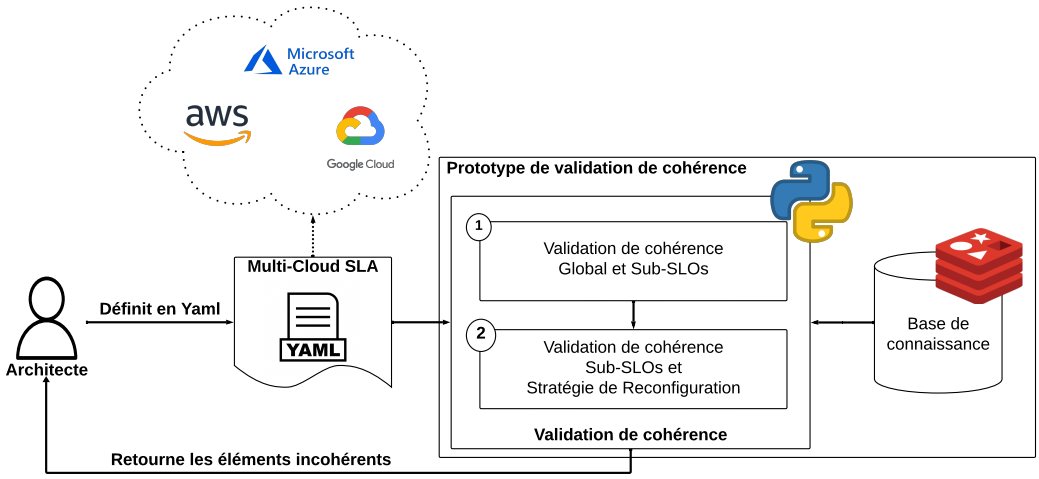


FIGURE 4.6 – Architecture du prototype de validation de la cohérence

Dans la suite de cette section, nous présenterons les différents composants nécessaires à la mise en œuvre de notre approche.

4.3.1.1 Base de connaissance

La base de connaissance, créée avec une base de données NoSQL, sert de base pour les deux autres composants de validation de cohérence entre Global et sub-SLOs, ainsi qu'entre Sub-SLOs et stratégie de reconfiguration.

Pour construire cette base, nous nous appuyons sur les besoins de nos algorithmes ainsi que sur les travaux sur la performance des brokers de messagerie proposés par Ramperez et al. dans [79]. La base est implémentée avec Redis [4], dans laquelle sont stockés les mécanismes d'agrégation nécessaires pour la validation de cohérence entre global-SLA et sub-SLA ainsi que sur les règles de traduction pour la validation de cohérence entre sub-SLA et stratégies de reconfiguration. Nous pouvons voir dans la Figure 4.7 un extrait de mécanismes d'agrégation stockés.

Également, on peut voir dans la Figure 4.8 un extrait des règles de traduction de SLOs en paramètres opérationnels.

4. `https://redis.io`

```
Mécanisme d'agrégation : (SLO, Opérateur d'agrégation, Calcul de seuil)
Key: Cout
Value: {'operateurAgregation': 'SumType', 'calculDeSeuil': 'SLO.v/nbc'}
Key: Debit
Value: {'operateurAgregation': 'MaxType', 'calculDeSeuil': 'SLO.v'}
Key: Disponibilité
Value: {'operateurAgregation': 'Disponibilite SumType', 'calculDeSeuil': '(SLO.v+(100*nbc-100)/nbc)'}
Key: Uptime
Value: {'operateurAgregation': 'ORType', 'calculDeSeuil': 'SLO.v'}
Key: Temps de reponse
Value: {'operateurAgregation': 'MaxType', 'calculDeSeuil': 'SLO.v'}
Key: Latence
Value: {'operateurAgregation': 'MaxType', 'calculDeSeuil': 'SLO.v'}
```

FIGURE 4.7 – Extrait depuis la base Redis des mécanismes d'agrégations associés aux caractéristiques de SLOs

```
#####
Traduction de SLOs : (SLO, Type d'application, Application, Paramètre opérationnel)
Key: (Debit,quantitatif,15000,notifications/s,>=)
Value: {'TypedApplication': 'MessageQueue', 'Application': 'ActiveMQ', 'ParametreOperationnel': '(Memory_Usage,80%,<=)'}
Key: (Debit,quantitatif,30000,notifications/s,>=)
Value: {'TypedApplication': 'MessageQueue', 'Application': 'RabbitMQ', 'ParametreOperationnel': '(CPU_Usage,80%,<=)'}
```

FIGURE 4.8 – Extrait depuis le base Redis des règles de traduction de SLOs en paramètres opérationnel

4.3.1.2 Validation de cohérence Global et sub-SLOs

Nous présentons l'implémentation en python des algorithmes 1 et 5 dans l'annexe A.1. Dans un premier temps, cette implémentation effectue, pour la validation ascendante (Algorithme 1), les calculs des agrégats à partir des sub-SLAs en utilisant les opérateurs d'agrégation définis dans la base de connaissance. Ensuite, nous comparons ces agrégats aux global-SLOs pour valider leur cohérence et ainsi identifier les global-SLOs incohérents.

Dans un second temps, cette implémentation effectue, pour la validation descendante (Algorithme 5), le calcul des seuils à partir des global-SLOs en utilisant les méthodes de calcul de seuils définies dans la base de connaissance. Ensuite, nous comparons ces seuils aux sub-SLOs pour valider leur cohérence, nous permettant ainsi d'identifier les sub-SLOs incohérents. Les résultats sont retournés pour signaler les problèmes détectés ou pour indiquer l'absence de problèmes.

4.3.1.3 Validation de cohérence Sub-SLOs et stratégie de reconfiguration

Nous présentons l'implémentation en python de l'algorithme 8 dans l'annexe A.2. Cette implémentation effectue la validation de cohérence entre les sub-SLOs et les machines à états formalisant les stratégies de reconfiguration. Pour ce faire, nous commençons par extraire les Sub-SLOs des différents sub-SLAs que nous traduisons à l'aide de la base de connaissance en paramètres opérationnels. Comme détaillé dans la section 4.2.3, nous comparons ensuite ces paramètres opérationnels traduits aux événements déclencheurs des machines à état. Cela permet ainsi d'identifier les stratégies de reconfiguration incohérente. Les résultats sont retournés pour signaler

les problèmes détectés ou pour indiquer l'absence de problèmes.

4.3.2 Évaluation

Dans cette section, nous évaluons notre approche en la soumettant à divers scénarios de test qui permettent d'évaluer sa capacité à identifier les incohérences. Ces scénarios englobent différentes situations, illustrant des incohérences qui peuvent se produire à différents niveaux d'un SLA multi-cloud dynamique (Global-SLA, Sub-SLA et stratégies de reconfiguration). Dans la première sous-section, nous présentons les différents scénarios d'évaluation, puis, dans la seconde les résultats de cette évaluation avec les sorties de notre prototype.

4.3.2.1 Scénarios d'évaluation

Nous décrivons 3 catégories de scénarios incohérents conçus pour mettre à l'épreuve notre approche de validation de la conformité des SLAs multi-cloud dynamiques. Chaque scénario est composé d'un global-SLA et de sub-SLAs avec chacun une machine à état associé telle que présentée dans les listings 3.1 à 3.5. Toutes les catégories de scénarios sont des variations d'un SLA multi-cloud dynamique de référence représentant notre exemple de motivation différenciée par le niveau auquel les incohérences sont introduites.

Pour chacune des catégories, nous définissons 10 scénarios différents avec des incohérences introduites aléatoirement. Dans la première catégorie, nous introduisons des incohérences au niveau des global-SLOs. Dans la seconde catégorie, nous introduisons des incohérences au niveau des sub-SLOs. Dans la troisième catégorie, nous introduisons des incohérences au niveau des stratégies de reconfiguration. La cohérence de chacun de ces scénarios sera évaluée avec notre approche et les résultats de ces derniers sont présentés dans la prochaine sous-section.

4.3.2.2 Résultat de l'évaluation

La sortie des exécutions de notre prototype avec un SLA multi-cloud ne présentant pas d'incohérence est présentée dans la Figure 4.9. La première ligne indique le SLA multi-cloud évalué. La sortie est décomposée en 2 grandes parties : (*i*) Validation entre Global-SLOs et Sub-SLOs (cadre **1**) et (*ii*) Validation entre Sub-SLOs et stratégie de reconfiguration (cadre **2**).

Dans la première partie, nous retrouvons l'exécution de l'étape de validation descendante avec le calcul des seuils de tolérance à partir des global-SLOs (cadre **1.1**), ainsi que l'étape de validation ascendante avec le calcul des agrégats à partir des sub-SLOs (cadre **1.2**).

La Figure 4.10 présente une validation faite sur un SLA multi-cloud contenant des incohérences de la première catégorie. Le format de la sortie est identique au précédent avec en plus les SLAs incohérent (cadre **1.3**). Dans ce scénario, l'objectif de niveau de service qui pose problème est le SLO de coût qui n'est pas cohérent. En effet, dans ce scénario, nous avons défini un global-SLO de coût incohérent avec

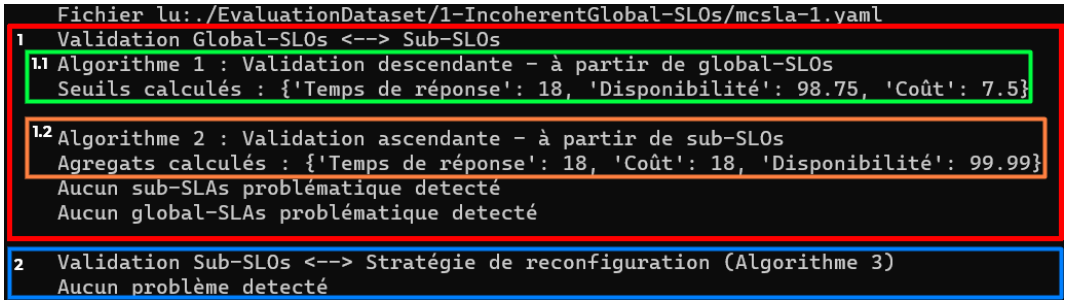


FIGURE 4.9 – Format de sortie de l'implémentation de l'algorithme de validation

les sub-SLOs, ce qui a induit un seuil de coût particulièrement bas impossible à respecter par les sub-SLAs. La sortie de l'implémentation identifie bien l'objectif incohérent comme étant le global-SLOs de coût.

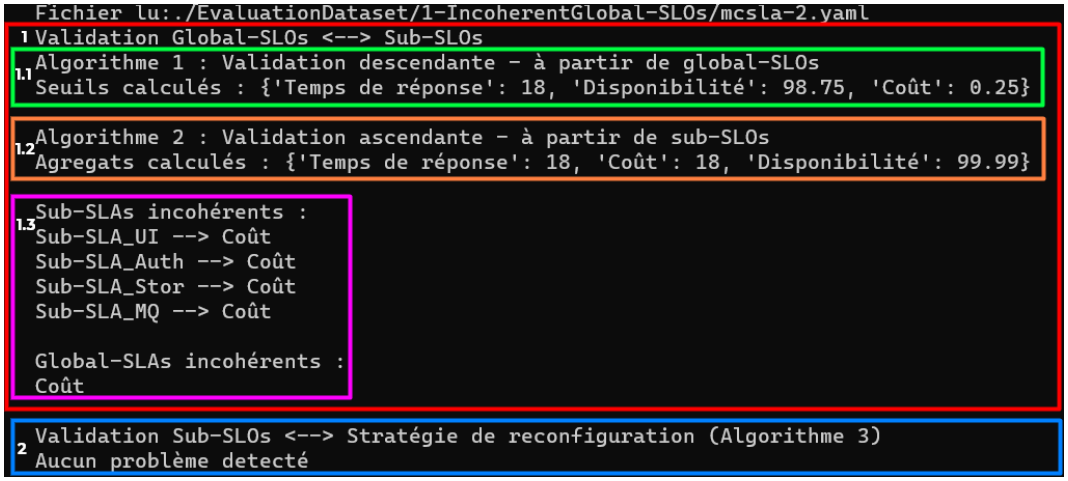


FIGURE 4.10 – Format de sortie de l'implémentation de l'algorithme de validation avec incohérences

L'issue des exécutions sur les différents scénarios incohérents est présentée dans le Tableau 4.3. Ce tableau résume les résultats obtenus pour chaque catégorie de scénarios évalués, le nombre d'incohérences que nous avons introduit et le nombre d'incohérences que notre approche a détectées.

TABLEAU 4.3 – Résultat de l'évaluation des quatre catégories de scénarios

Scénarios évalués	Incohérences introduites	Incohérences détectés
Global-SLOs incohérents	10	10
Sub-SLOs incohérents	60	57
Stratégies de reconfiguration incohérentes	43	43

Ces résultats montrent que pour chacun des scénarios notre approche a permis de détecter la majorité des incohérences introduites dans les différents scénarios de

SLA multi-cloud dynamique. Les global-SLOs et les stratégies de reconfiguration incohérentes ont tous pu être détectés. Pour les sub-SLOs incohérents, certaines incohérences n'ont pas pu être détectées à cause d'une limitation de notre méthode de définition de calcul de seuil de tolérance. En effet, pour un global-SLO donné les seuils de tolérance calculés sont les mêmes pour tous les sub-SLOs, ce qui peut être limitant pour certains types de SLO comme le coût. En effet, le coût entre les différents composants n'est pas forcément réparti uniformément. Cela nécessiterait de faire évoluer les mécanismes de calcul de seuil en leur permettant de considérer une certaine pondération en fonction du composant.

Dans la suite de nos travaux, nous considérons principalement deux évolutions : (*i*) permettre la définition de mécanismes de calcul de seuil plus complexe et (*ii*) étendre la base de connaissance pour de nouveaux SLOs et de nouvelles applications. Ceci nous permettra d'utiliser notre implémentation sur un plus grand nombre d'applications.

4.4 Conclusion

Dans ce chapitre, nous avons proposé une méthode de validation de cohérence pré-mise en œuvre des SLAs multi-cloud dynamique répondant à notre second objectif **O2.1** de validation pré-mise en œuvre des SLAs multi-cloud dynamique. Nous avons proposé une méthode pour la validation de cohérence dans un SLA multi-cloud dynamique entre : (*i*) les global-SLOs et les sub-SLOs et (*ii*) les sub-SLOs et leurs stratégies de reconfiguration associées.

Notre méthode de validation (*i*) entre global-SLOs et sub-SLOs basé sur une technique d'agrégation de SLO, décomposée en deux phases : descendante (Global-SLOs vers sub-SLOs) puis ascendante (Sub-SLOs vers Global-SLOs) pour permettre d'identifier les incohérences dans les deux sens. Cette double validation permet également de ne pas favoriser dans l'analyse ni les sub-SLOs, ni les global-SLOs. En effet, favoriser soit les sub-SLOs, soit les global-SLOs impliquerait une hiérarchie entre eux, ce qui sortirait du contexte de nos travaux de contrôle d'un SLA multi-cloud dynamique défini par un architecte cloud.

Notre méthode de validation (*ii*) entre sub-SLOs et stratégie de reconfiguration basée sur la traduction de SLO, nous a permis de faire une validation entre les Sub-SLOs et les stratégies de reconfiguration qui sont à deux niveaux d'abstraction différents : *fonctionnel* pour les caractéristiques des sub-SLOs et *opérationnel* pour les événements déclencheurs des stratégies de reconfiguration. Ainsi, de nous assurer que les stratégies de reconfiguration définies ne violeraient pas les sub-SLOs défini.

Par la suite, nous avons implémenté cette méthode sous forme d'un prototype en Python et évalué l'approche avec différents scénarios de SLA multi-cloud incohérents. Cette évaluation nous a permis de démontrer l'efficacité de notre méthode qui a permis d'identifier la majorité des incohérences introduites. Toutefois, pour améliorer cette efficacité, nous devons enrichir la base de connaissance avec de nouveaux mécanismes d'agrégation et règles de traduction de SLOs ce qui permettra

l'utilisation de notre approche sur un plus grand nombre de systèmes multi-cloud.

Une partie des travaux présentés dans ce chapitre ont été publiés dans : *Jeremy Mechouche, Roua Touihri, Mohamed Sellami, Walid Gaaloul : Towards higher-level description of SLA-aware reconfiguration strategies based on state machine, IEEE International Conference on E-Business Engineering, Guangzou, China, 2021, 1-8.*

Ces mécanismes de validation sont effectués uniquement avant la mise en œuvre du SLA multi-cloud et rien ne garantie que ce qui est modélisé sera bien exécuté par les fournisseurs. Par conséquent, dans le prochain chapitre, nous nous intéresserons à la validation post-mise en œuvre du SLA multi-cloud dynamique adressant notre dernier objectif **O2.2** de vérification post-mise en œuvre.

Chapitre 5

Reporting de SLA multi-cloud dynamique

Sommaire

Dans le chapitre précédent, nous avons détaillé notre approche pour la validation de la cohérence pré-mise en œuvre de SLA multi-cloud dynamique. La phase de mise en œuvre est, selon le cycle de vie du SLA (Section 2.2.1.1), la phase qui succède naturellement à celle de validation de cohérence. Pour un SLA multi-cloud dynamique, cette phase implique la fourniture de services respectant les niveaux de services convenus avec les différents fournisseurs et mettant en œuvre les stratégies de reconfiguration établies. La réalisation du reporting, c'est-à-dire l'évaluation rigoureuse post-mise en œuvre, est alors primordiale pour s'assurer que les niveaux de services fournis correspondent aux objectifs définis dans le SLA, tout en prenant en considération les stratégies de reconfiguration. Cependant, comme vu dans l'état de l'art (Section 2.3), nous trouvons principalement des travaux qui se concentrent uniquement sur le suivi des SLOs, mais qui ne prennent pas en compte le suivi des stratégies de reconfiguration permettant le maintien de ces SLOs.

Dans cette optique, nous proposons l'utilisation de méthodes issues de la fouille de processus métier [92] (*angl. Process Mining*), en particulier la technique de vérification de conformité [27] (*angl. conformance checking*), qui permet de s'assurer qu'un processus métier défini est en adéquation avec les activités réellement exécutées, telles que consignées dans des journaux d'événements. Dans ce chapitre, nous introduisons notre approche pour la vérification post-mise en œuvre des SLAs multi-cloud dynamique, en nous appuyant sur une technique de vérification de conformité. Cependant, pour permettre de tirer parti de cette technique, il est nécessaire de préparer les journaux d'événements (aussi appelés traces d'exécution), que nous collectons auprès des fournisseurs de services cloud et des services fournis, pour les rendre comparables avec la définition dans le SLA multi-cloud dynamique. Ainsi nous procédons en deux phases : (*i*) le pré-traitement des journaux d'événements et (*ii*) l'application d'une technique de vérification de conformité.

Le reste de ce chapitre est organisé comme suit : dans la section 5.1, nous introduisons les techniques de fouille de processus. Puis, dans la section 5.2, nous donnons un aperçu sur notre approche de reporting post-mise en œuvre de SLA multi-cloud. Ensuite, dans la section 5.3, nous décrivons notre procédure de pré-traitement des journaux d'événements. Par la suite, dans la section 5.4, nous détaillons la manière dont nous mettons en œuvre une technique de vérification de conformité entre SLA multi-cloud dynamique et journaux d'événements. Finalement, dans la section 5.5, nous présentons la mise en œuvre de notre approche de vérification post-mise en œuvre de SLA Multi-Cloud dynamique.

5.1 Contexte : Fouille de processus

En gestion de processus métiers (*angl. Business Process Management, BPM*) [26], les techniques de fouille de processus permettent l'analyse de processus métier en se basant sur un modèle de processus et des journaux d'événements générés lors de son exécution [2]. La fouille de processus peut être considérée comme le lien entre la science des données (*angl. Data Science*) et la gestion des processus métiers. Les techniques de fouille de processus visent à découvrir, surveiller et améliorer les processus métier en extrayant des connaissances à partir de journaux d'événements. Un journal d'événements stocke des données sur les activités enregistrées par des systèmes d'information lors de l'exécution d'un processus.

Il existe quatre grandes catégories de techniques de *fouille de processus* comme illustré dans la Figure 5.1 :

1. Les techniques de **découverte automatisée de processus** (*angl. automated discovery*) produisent un modèle de processus à partir d'un ensemble donné de journaux d'événements.
2. Les techniques d'**analyse de performance** (*angl. performance mining*) produisent un modèle de processus amélioré à partir d'un modèle d'entrée et d'un journal d'événements. Ces techniques permettent de comprendre le comportement du processus et d'identifier des problèmes (e.g., goulots d'étranglement).

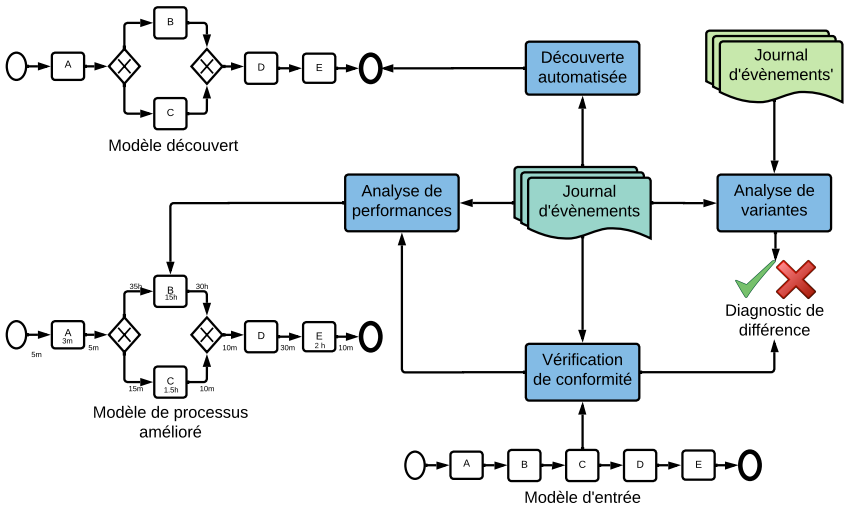


FIGURE 5.1 – Catégories de techniques de fouille de processus (tiré de [26]).

3. Les techniques d'**analyse de variante** (*angl. variant analysis*) produisent différents diagnostics à partir de deux journaux d'événements différents. Ces techniques comparent des journaux d'événements relatifs à des instances de processus se terminant positivement (c'est-à-dire que toutes les instances de processus réussissent) avec des journaux d'événements relatifs à des instances de processus se terminant négativement (c'est-à-dire que toutes les instances de processus échouent). Cela permet d'identifier les différences entre les instances se terminant positivement et les instances se terminant négativement.
4. Les techniques de **vérification de conformité** (*angl. conformance checking*) produisent différents diagnostics entre un modèle de processus et un journal d'événements pour identifier les similitudes et les écarts. Ces techniques permettent de calculer des mesures de conformité, c'est-à-dire, de trouver le niveau de conformité entre les modèles de processus et les journaux d'événements générés par les exécutions des modèles de processus.

Dans notre travail, nous visons à exploiter les techniques de vérification de conformité pour la génération de rapports de conformité entre les services fournis et attendus. Pour ce faire, nous considérons une machine à état, représentant une stratégie de reconfiguration, comme modèle d'entrée et les journaux d'événements collectés auprès des fournisseurs de service cloud, des orchestrateurs et des services fournis lors de la mise en œuvre des SLAs. Dans la suite de cette section, nous détaillons cette catégorie de méthodes de fouille de processus dans la Section 5.1.1, abordons les techniques de pré-traitement des journaux d'événements dans la Section 5.1.2 et concluons avec une discussion sur l'intérêt de la fouille de processus dans la vérification de conformité des SLAs multi-cloud dynamique dans la Section 5.1.3.

5.1.1 Vérification de conformité

La vérification de conformité est une catégorie de technique utilisée en fouille de processus métier pour comparer un modèle de processus préalablement défini avec les données réelles collectées sous forme de journaux d'événements. L'objectif de la vérification de conformité est de détecter les écarts entre le comportement attendu du processus, tel que défini dans le modèle et le comportement observé dans les événements enregistrés dans un journal. Pour ce faire, cette technique prend en entrée 2 paramètres : un modèle de processus et un journal d'événements (Section 5.1.1.1). On trouve principalement deux types de techniques : la Relecture [13] (*angl. token replay*)(Section 5.1.1.1) et l'Alignement [5] (Section 5.1.1.3).

5.1.1.1 Paramètres d'entrées pour la vérification de conformité

Comme illustré dans Figure 5.1, la vérification de conformité prend en entrée deux paramètres : un journal d'événements et un modèle de processus.

Un journal d'événements est un ensemble ordonné d'événements. Chaque événement dans le journal doit contenir (1) un identifiant unique pour une instance de processus particulière (appelé identifiant de cas), (2) une activité (description de l'événement qui se produit) et (3) un horodatage (*angl. timestamp*). Il peut y avoir des attributs d'événement supplémentaires se référant à des ressources, des coûts etc., mais ceux-ci sont optionnels. Chaque événement nous renseigne sur l'exécution d'une activité du processus (par exemple, le début ou la fin d'une activité). Dans le contexte de la fouille de processus, on parlera d'une trace pour définir une suite d'événement associé à une même instance de processus.

Un modèle de processus d'entrée peut être défini comme une représentation schématique (e.g. BPMN, réseau de pétri) ou un ensemble de règles et de paramètres décrivant les séquences d'activités, les chemins possibles, les conditions et les contraintes d'un processus métier. Ce modèle sert de référence pour comparer les événements enregistrés dans les journaux d'événements avec le déroulement théorique du processus. Il permet ainsi de détecter les écarts ou déviations dans l'exécution réelle du processus par rapport à sa conception prévue.

5.1.1.2 Vérification de conformité par relecture

La vérification de conformité basée sur la relecture consiste à rejouer les événements identifiés dans une trace de journal d'événements sur un processus. En d'autres termes, nous allons reprendre chaque événement l'un après l'autre et essayer de le rejouer en suivant le modèle du processus d'entrée. Cette méthode se base sur des modèles sous forme de réseau de Petri. Elle utilise des jetons (*angl. tokens*) pour rejouer l'exécution du processus selon le modèle et compare les traces d'événements avec les chemins possibles du modèle. En utilisant quatre compteurs de jetons : les jetons produits (p), les jetons consommés (c), les jetons manquants (m) et les jetons restants (r), qui permettent d'enregistrer les situations où une transition est forcée à se déclencher et les jetons restants après la fin de la relecture.

Sur la base des valeurs de chaque compteur, le taux de conformité entre la trace et le modèle est calculé avec l'équation 5.1.

$$\frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p}) \tag{5.1}$$

La Figure 5.2 illustre un exemple de réseau de Petri pour lequel nous effectuons une vérification de conformité par relecture.

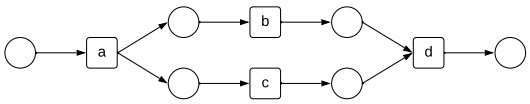


FIGURE 5.2 – Exemple de réseau de Petri

A travers cette relecture, nous tentons de rejouer la trace d'événements (a,b,d) enregistrés dans le journal d'événements sur le réseau de Petri. La Figure 5.3 illustre la relecture faite en 5 étapes :

1- 1 jeton est produit sur la place initiale ($p = 1$),
2- l'activité a consomme 1 jeton et en produit 2 ($p = 1 + 2 = 3$, $c = 1$),
3- l'activité b consomme et produit 1 jeton ($p = 3 + 1 = 4$, $c = 1 + 1 = 2$),
4- l'activité d nécessite 2 jetons pour être déclenchée, mais il n'y en a qu'un seul, un jeton artificiel est donc produit, l'activité d consomme 2 jetons et produit 1 jeton ($p = 4 + 1 = 5$, $c = 2 + 2 = 4$, $m = 1$)
5- le jeton dans la dernière place est consommé et il reste toujours un jeton dans le réseau [a,c] ($p = 5$, $c = 4 + 1 = 5$, $m = 1$ et $r = 1$)

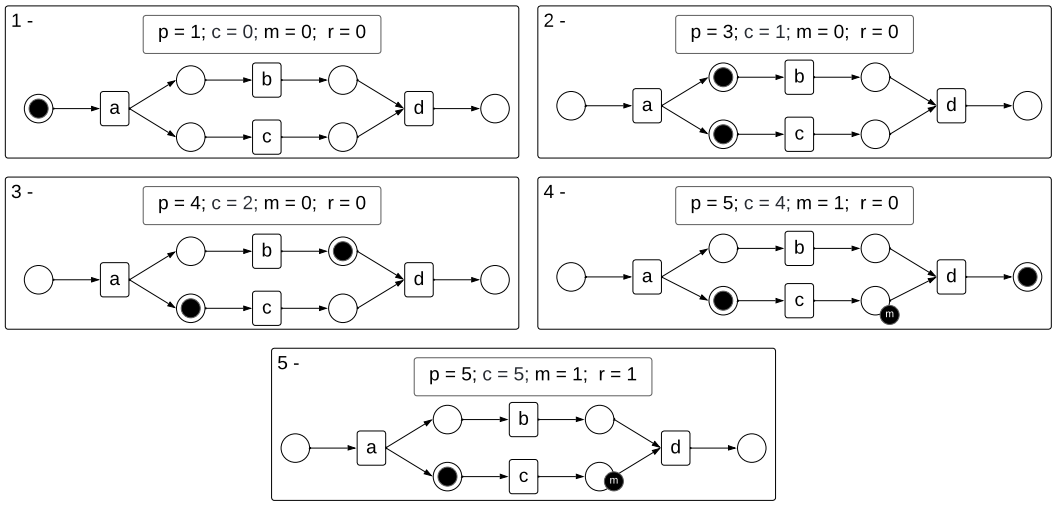


FIGURE 5.3 – Exemple de vérification de conformité par relecture

Les compteurs donnent $p = 5$, $c = 5$, $m = 1$ et $r = 1$. Ce qui donne une valeur de conformité de 0.8 en utilisant la formule 5.1 :

$$\frac{1}{2}(1 - \frac{1}{5}) + \frac{1}{2}(1 - \frac{1}{5}) = 0.8$$

5.1.1.3 Vérification de conformité par alignement

Bien que la technique de relecture de jetons soit efficace et facile à comprendre, cette approche est faite uniquement pour les réseaux de Petri et ne montre pas les cas fonctionnels non inclus dans le modèle de processus. En effet, certains cas peuvent ne pas être représentés dans le modèle, mais être tout de même fonctionnels. Par exemple dans une entreprise, le modèle de processus officiel pour les achats peut impliquer plusieurs étapes de validation pour garantir le contrôle des dépenses. Cependant, pour de petites dépenses jugées mineures, il se peut qu'un processus informel se développe parmi les employés, où ils effectuent des achats directement avec une approbation verbale de leurs supérieurs. Ce processus, bien que fonctionnel et courant, n'est généralement pas capturé dans le modèle de processus formel de l'organisation.

Les alignements ont alors été introduits pour résoudre ces limitations et consistent en la recherche du chemin le plus proche d'une trace en se basant sur ce qui est observé. L'algorithme effectue une recherche exhaustive pour trouver l'alignement optimal entre la trace observée et le modèle de processus. Ces derniers peuvent être appliqués à toute notation de modélisation de processus (e.g., BPMN).

La technique d'alignement vise à comparer les traces d'exécution d'un journal d'événements avec les chemins possibles du modèle de processus. L'alignement consiste à rechercher une séquence d'activités du modèle de processus qui correspond le mieux à chaque trace d'exécution réelle, en tenant compte des éventuelles déviations. Cette approche permet de mesurer le degré de conformité entre le modèle et les données réelles, identifiant ainsi les erreurs et les déviations (écarts) dans une instance de processus métiers.

Un alignement est composé de mouvements où un événement du journal et une activité du modèle de processus sont associés. Si l'événement et l'activité ne correspondent pas directement, le mouvement est considéré comme une déviation, symbolisée par "≫". Cela permet de détecter et d'analyser les écarts entre le comportement prévu et le comportement réel du processus.

Par exemple, en considérant le même réseau de Petri que la section précédente illustrée dans la Figure 5.2, on peut voir dans le Tableau 5.1 la ligne du haut correspond au modèle de processus et la ligne du bas correspond aux événements des journaux d'événements, deux alignements γ_1 avec la trace $\langle a, b, c, d\rangle$ sans déviations et γ_2 avec la trace $\langle a, b, d\rangle$ avec déviations.

TABLEAU 5.1 – Exemple d'alignement de traces (tiré de [27])

$$\gamma_1 = \frac{\begin{array}{c|c|c|c} \text{a} & \text{b} & \text{c} & \text{d} \end{array}}{\begin{array}{c|c|c|c} \text{a} & \text{b} & \text{c} & \text{d} \end{array}}, \quad \gamma_2 = \frac{\begin{array}{c|c|c|c} \text{a} & \text{b} & \text{c} & \text{d} \end{array}}{\begin{array}{c|c|c|c} \text{a} & \text{b} & \gg & \text{d} \end{array}}$$

Pour trouver l'alignement optimal, une fonction de coût associant chaque mouvement de l'alignement est définie et pondéré par un expert du domaine en fonction de la criticité du mouvement. Ce qui permet alors de calculer un taux de conformité

entre le modèle et le journal d'événement. Nous verrons plus en détail cette fonction ainsi que la méthode de recherche d'alignement dans la suite de ce chapitre.

Toutefois, ces méthodes supposent que chaque événement identifié dans un journal d'événements est directement associable à une activité dans un modèle de processus. Alors que dans les cas réels, il est souvent nécessaire d'associer plusieurs événements pour identifier une activité. On parlera alors d'abstraction d'événements en activité de plus haut niveau. C'est pourquoi il est primordial d'effectuer une étape de pré-traitement des journaux d'événements pour pouvoir utiliser ces méthodes de vérification de conformité. Dans la prochaine section, nous nous intéressons aux méthodes de pré-traitement de journaux d'événements et plus spécifiquement aux techniques d'abstraction d'événements.

5.1.2 Pré-traitement de journaux d'événements

Le pré-traitement des données est l'étape qui précède celle de l'analyse des données. Le pré-traitement est effectué sur des données dites *brutes* directement issues de la collecte. Il est constitué de plusieurs tâches effectuées sur les données comme le nettoyage [48], l'enrichissement [47] ou encore la fusion [94].

Dans le cadre de la fouille de processus, ce pré-traitement est effectué sur des journaux d'événements correspondants aux données brutes collectées. Le principal élément que l'on traite est l'association d'un ensemble d'événements à une activité du modèle de processus métiers. Dans la majorité des travaux liés à la fouille de processus, un événement collecté dans un journal d'événements est supposé correspondre à l'exécution d'une activité d'un processus. Nous parlons ici d'association un-pour-un (*angl. mapping one-to-one*) : un événement correspond à une activité. Toutefois, cela ne s'applique pas à tous les cas d'usage et il est nécessaire de grouper plusieurs événements pour identifier une activité. Nous parlons alors d'abstraction d'événements de bas niveau en activité [7].

Nous présentons dans la Figure 5.4 un exemple de plusieurs événements, de bas niveau, enregistrés correspondant à une activité de processus de haut niveau : la séquence d'événements ⟨reg_act_start ; opsi_pp_open ; reg_act_end⟩ correspondant à l'activité *register request*.

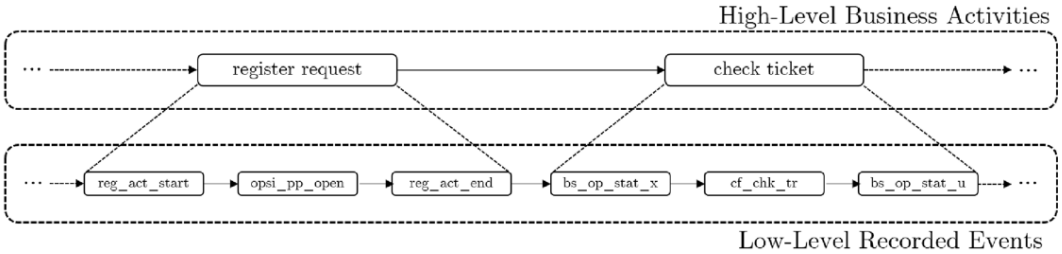


FIGURE 5.4 – Exemple de visualisation d'événements enregistrés à un niveau de granularité différent de celui d'une activité (tiré de [96])

Les méthodes d'abstraction d'événements sont classées en deux groupes : les méthodes *non supervisées* et les méthodes *supervisées*. Les méthodes non supervisées

permettent une association automatique des événements et se basent principalement sur des méthodes d'apprentissage automatique, tel que [6, 7]. Les méthodes supervisées quant à elles permettent une association des événements en se basant sur des connaissances métiers tels que l'utilisation de termes techniques similaires [60] ou l'identification de motifs spécifiques [66]. En conclusion, les méthodes non supervisées permettent de construire des approches moins spécifiques au domaine et applicables dans plusieurs cas d'utilisation. Les méthodes supervisées permettent une plus grande spécificité au domaine en raison de l'utilisation de connaissances spécifiques au domaine.

5.1.3 Discussion

Dans cette section, nous avons pu constater que la fouille de processus, grâce aux techniques de vérification de conformité, permet d'effectuer une comparaison entre ce qui s'est produit selon les événements enregistrés dans un journal d'événements et ce qui était supposé se produire selon un modèle de processus métier. L'utilisation de techniques de vérification de conformité s'aligne sur notre objectif de vérification post-mise en œuvre de SLA multi-cloud dynamique et tout particulièrement la vérification de conformité par alignement, car elle permet la comparaison entre n'importe quel modèle de processus et son exécution réelle, tel qu'enregistrée dans le journal d'événements. Cette technique constitue une analyse post-mortem visant à produire un rapport d'exécution a posteriori, plutôt qu'une analyse en temps réel généralement axée sur la correction d'incident.

Dans notre contexte, la machine à état présente dans nos SLA multi-cloud dynamique sera assimilée aux modèles de processus dont la conformité est à vérifier. Les journaux d'événements sont collectés après la mise en œuvre d'un SLA multi-cloud dynamique auprès de différents fournisseurs de services cloud. Toutefois, ces événements collectés ne peuvent pas être associés directement aux composants d'une machine à état et nécessiteront d'être abstraits.

Comme nous avons pu le voir dans l'état de l'art (Section 2.3.2), il n'existe pas de méthode permettant le suivi des SLAs multi-cloud dynamique. Dans la suite de ce chapitre, nous nous baserons sur les méthodes de fouille de processus pour le reporting de SLA multi-cloud dynamique et plus particulièrement sur les techniques de vérification de conformité par alignement.

5.2 Vérification post-mise en œuvre de SLA multi-cloud dynamique

Dans cette section, nous présentons une vision globale de notre approche de vérification post-mise en œuvre de SLA multi-cloud dynamique basée sur la fouille de processus, illustrée dans la Figure 5.5. Notre approche se décompose donc en deux étapes : ① le pré-traitement des journaux d'événements (Section 5.3) et ② la vérification de conformité des SLAs multi-cloud dynamique (Section 5.4).

Comme expliqué dans la discussion précédente, nous avons besoin, pour comparer les journaux d'événements collectés aux machines à état du SLA multi-cloud dynamique, d'effectuer une abstraction des événements collectés en éléments de machine à état. Pour ce faire, nous adaptons une méthode supervisée d'abstraction d'événements [60] qui détecte des motifs pré-définis dans un journal d'événements. Les événements étant collectés auprès de différents fournisseurs de service cloud les événements sont très hétérogènes. Nous commençons donc par annoter les journaux d'événements pour permettre une détection de motifs agnostique des spécificités des fournisseurs et ainsi adresser ce problème d'hétérogénéité.

La première étape ① de pré-traitement consiste en deux phases : ①.① l'annotation des journaux d'événements collectés résultant de la mise en œuvre des SLAs multi-cloud en se basant sur une ontologie définissant le lien entre les événements et les composants d'une machine à état pour relever le défi d'hétérogénéité entre les journaux d'événements issus de différents fournisseurs de service cloud et permettre la phase suivante d'abstraction ; puis en ①.②, l'abstraction des événements en éléments de machine à états basés sur la reconnaissance de motifs dans les journaux d'événements retourne des journaux pré-traités. La seconde étape ② consiste en la vérification de la cohérence entre le SLA multi-cloud dynamique et les journaux d'événements pré-traités en utilisant une technique de vérification de conformité basée sur l'alignement. Puis, retourne à l'architecte un rapport détaillant les éventuelles déviations identifiées. Nous présentons dans la suite de ce chapitre en détail chacun des composants de notre approche.

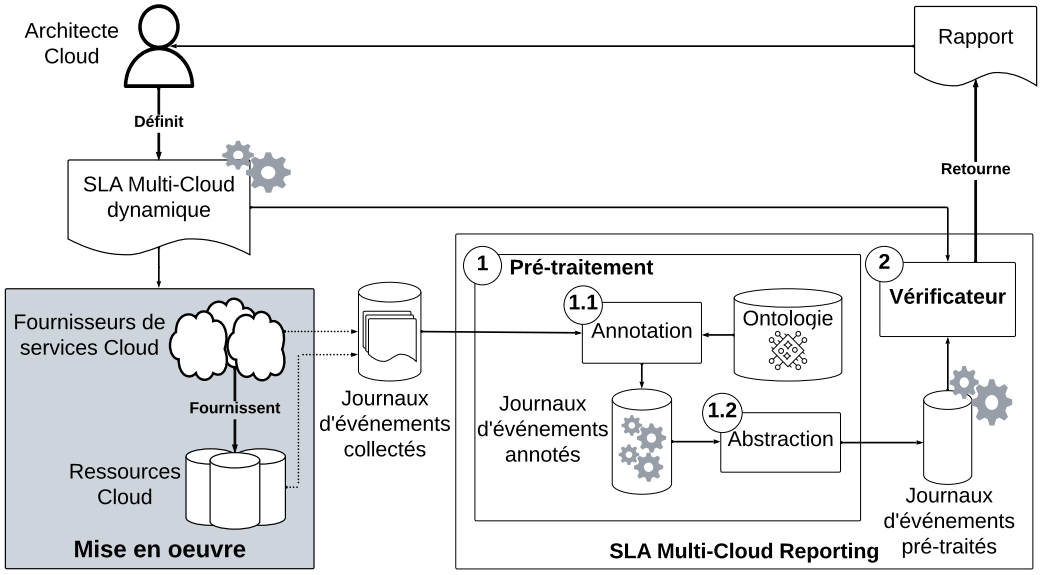


FIGURE 5.5 – Approche de vérification de conformité post-mise en œuvre de SLA Multi-Cloud dynamique

5.3 Pré-traitement de journaux d'événements collectés

Un événement est l'enregistrement d'une action par un système d'information lors de l'exécution d'un processus. Un journal d'événements consiste en une collection ordonnée d'événements evt_i. Ces événements sont collectés depuis différentes sources, telles que, les fournisseurs de services cloud, les orchestrateurs de ressources et les ressources elles-mêmes. Nous définissons un événement comme suit :

Définition 5.1 (événement) *Un **événement** evt est un 6-uplet (ts, src, src_{name}, t, r, M) où ts est l'horodatage (angl. timestamp) de l'événement, src est le type de source* $\in \{orchestrateur, fournisseur, ressource\}$*, src_{name} est le nom de la source, t est le type de l'événement collecté (e.g., dans le contexte docker swarm* $t \in \{Service\_Create, Container\_Create, Container\_Start\}$*), r est le nom de la ressource associée à l'événement et M est un ensemble de couples (métrique, valeur) fournis par la source de données (e.g. (Utilisation du CPU, 15)).*
Pour illustrer cette définition considérons : $evt1 = \langle ts = 00:01, src = orchestrateur, src_{name} = UI, t = Service\_Create, r = UI, M = [CpuUsage : 95\%]\rangle$

Toutefois, comme présenté dans la section précédente, les journaux d'événements ne peuvent pas être traités en l'état par les algorithmes de vérification de conformité, car ils ne sont pas comparables aux machines à état décrivant les stratégies de reconfiguration présentes dans les SLAs multi-cloud dynamique. C'est pourquoi, il est nécessaire de pré-traiter les journaux d'événements pour les rendre comparables (étape ①). Pour ce faire, nous adaptons une technique d'abstraction d'événement supervisée proposé par *Leonardi et al.* dans [60]. Cette technique utilise une base de connaissance représentant les connaissances métier du domaine du processus étudié formalisé dans une ontologie permettant l'abstraction d'événements issus des journaux d'événements. Nous adaptons donc cette technique à l'association des événements issus des journaux d'événements collectés auprès de différents fournisseurs de service cloud aux éléments de machine à état.

Nous effectuons ce pré-traitement en deux phases : ①.① tout d'abord, une phase d'*annotation* qui permettra d'inférer à partir d'une ontologie le lien des événements du journal avec une étape du cycle de vie d'un élément de machine à état et d'ajouter cette information dans l'événement permettant ainsi de résoudre la problématique d'hétérogénéité et à ①.② la phase d'*abstraction* d'associer à partir des motifs prédéfinis plusieurs de ces événements à un des éléments de machine à état (état ou transition). Dans la suite, nous décrivons la phase ①.① d'annotation dans la Section 5.3.1 et la phase ①.② d'abstraction dans la Section 5.3.2.

5.3.1 Annotation de journaux d'événements

Lors de cette première phase, notre objectif est d'annoter les événements *evt* (Définition 5.1) issue des journaux d'événement brut collecté (*Logs*) en ajoutant des informations à chaque événement en fonction de leur type. A l'issue de cette phase,

nous obtenons des journaux d'événements annotés ($Logs_a$), composés d'événements annotés (evt_a) tels que :

Définition 5.2 (événement annoté) *Un* ***événement annoté*** evt_a *est un 8-uplet (ts, src, src_{name}, t, r, M, smElt, lcStep) où ts, src, src_{name}, t, r, M sont équivalent à la Définition 5.1, smElt correspond à l'élément de machine à état auquel cet événement pourra être associé tel que* $smElt \in \{state, transition\}$ *et lcStep correspond à l'étape du cycle de vie de cet élément tel que* $lcStep \in \{Start, Execute, Complete\}$.

Pour effectuer cette annotation, nous définissons une ontologie, décrite dans la section 5.3.1.1. Cette ontologie définit les annotations à réaliser et sera appliquée dans notre algorithme d'annotation automatique des journaux d'événements, expliqué dans la section 5.3.1.2.

5.3.1.1 Ontologie pour l'annotation d'événements de machines à états

Pour accomplir efficacement cette étape d'annotation, l'utilisation de connaissances spécialisées métier est indispensable. La première étape consiste donc en la modélisation des connaissances d'un expert du domaine sous la forme d'une ontologie liant les éléments de la machine à état à des types d'événements t (Définition 5.1). Cette étape nous permet de formaliser dans cette ontologie l'annotation pour des événements recueillis auprès de différents fournisseurs de service cloud (tel que AWS, Azure ou GCP). La structure de cette ontologie est illustrée dans la Figure 5.6. Cette ontologie sera utilisée pour annoter les événements collectés dans un journal d'événements (Log) en fonction de leur type. Nous leur associons l'attribut $smElt$ ($smElt \in \{State, Transition\}$) dont la valeur correspond à l'un des concepts $State$ ou $Transition$ représentant l'élément de la machine à état et définis dans l'ontologie (Figure 5.6).

La classe *state* est composée de 3 sous-classes capturant les étapes de son cycle de vie ($lcStep \in \{Start, Execute, Complete\}$) qui peuvent être $Start$, $Execute$ ou $Complete$. Ces 3 sous-classes sont donc des descendants de *State*. La classe *transition* n'a quant à elle pas de descendant décrivant les étapes du cycle de vie ($lcStep$), cela découle du motif permettant l'identification des transitions qui se déclenche entre 2 états. La propriété d'objet *isRelatedTo* avec le domaine *eventType* et la plage *machine à état* permettent d'associer un type d'événement et un élément de la machine à état.

Par exemple, en considérant un log brut issue de Docker Swarm, la propriété d'objet *isRelatedTo* nous permet de spécifier que $Service\_Create$ *isRelatedTo Start*. Nous pourrons alors dire que l'ascendant du type d'événement $Service\_Create$ est *Start*, qui a lui même comme ascendant *State*. Ce qui nous permet de conclure que cet événement doit être annoté avec l'étape du cycle de vie ($lcStep$) *Start* et l'élément de machine à états ($smElt$) *State*. En d'autres termes, pour $evt1 = \langle ts = 00:01, src_{name} = UI, t = Service\_Create, ...\rangle$, nous annotons $evt1_a = \langle ts = 00:01, src_{name} = UI, t = Service\_Create, ..., smElt = State, lcStep = Start\rangle$.

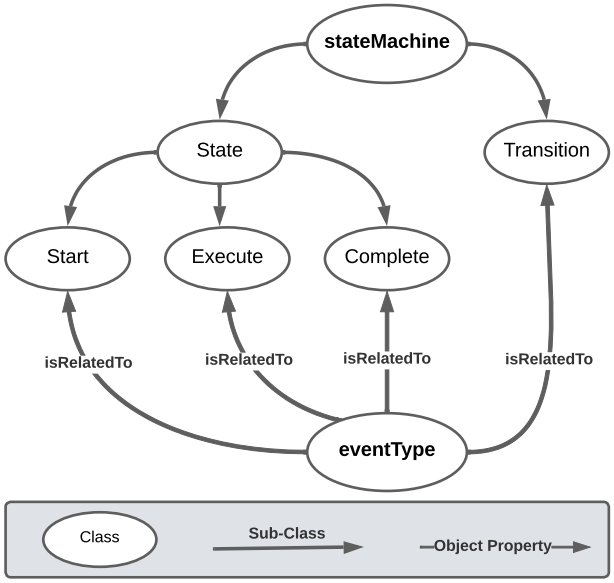


FIGURE 5.6 – Ontologie du domaine des machines à états

5.3.1.2 Annotation automatique de journaux d'événements

En nous basant sur l'ontologie de domaine des machines à états (Section 5.3.1.1), nous présentons dans cette section notre technique d'annotation automatique de journaux d'évènement brut collectés ($Logs$). Notre technique décrite dans l'algorithme 9 prend en entrée un journal d'événement brut collecté ($Logs$) et une ontologie du domaine des machines à état (O_{SM}). Nous initialisons un ensemble vide qui contiendra le journal d'événements annotés ($Logs_a$) (ligne 1). Ensuite, nous itérons pour chaque événement (evt) du journal d'événements brut collecté (ligne 2) et extrayons ses ascendants ($lcStep$ et $smElt$) depuis l'ontologie ($\mathcal{O}_{SM}$) à partir du type d'événement ($evt.t$) (ligne 3). Si ce dernier a des ascendants alors nous annotons l'événement avec ses ascendants qui correspondent d'abord à $lcStep$ puis à $smElt$ (ligne 4-7) sinon l'événement n'est pas annoté. L'événement annoté est ensuite ajouté au journal d'événement annoté $Logs_a$ (ligne 8). Une fois tous les événements parcourus le journal d'événements annotés est retourné et utilisé pour la phase d'abstraction.

Algorithme 9 Annotation de journal d'événement

Entrée: $Logs$: Journal d'événement brut collecté
$\mathcal{O}_{SM}$: Ontologie du domaine des machines à état
Sortie: $Logs_a$: Journal d'événement annoté

$Logs_a \leftarrow \emptyset$
▷ Itère à travers le journal d'événements brut collecté
for each $evt \in Logs$ **do**
▷ Récupère les ascendants de ce type d'événement ($evt.t$) depuis $\mathcal{O}_{SM}$
  $ascendant \leftarrow ascendants(evt.t,\ \mathcal{O}_{SM})$
  if $ascendant \neq null$ **then**
  ▷ Associe les ascendants du type d'événement ($evt.t$) avec les événements
    $evt_a \leftarrow evt$
    $evt_a.lcStep \leftarrow ascendant[0]$
    $evt_a.smElt \leftarrow ascendant[1]$
    $Logs_a \leftarrow Logs_a \cup evt_a$
  end if
end for
Return $Logs_a$

Pour illustrer une exécution de cet algorithme, nous considérons le journal d'événement brut collecté ($Logs$) présenté dans le Tableau 5.2. Ce journal est composé de 3 événements suivant la Définition 5.1. En exécutant, l'algorithme avec en entrée ce journal et l'exemple d'ontologie de la section précédente, nous obtenons un journal d'événements annoté présenté dans le Tableau 5.3. Ce journal est composé des 3 mêmes événements mais cette fois annoté avec les informations de l'ontologie en suivant la Définition 5.2. Par exemple, $evt_1 = \langle ts = 00:01, src = orchestrateur, src_{name} = UI, t = Service\_Create, r = UI\rangle$ est annoté tel que $evta_1 = \langle ts = 00:01, src = orchestrateur, src_{name} = UI, t = Service\_Create, r = UI, smElt = State, lcStep = Start\rangle$.

Tableau 5.2 – Journal d'événements brut ($Logs$)

	Timestamp	**Source**	src_{name}	**Type**	**Resource**
evt_1	00 : 01	orchestrateur	UI	Service_Create	UI
evt_2	00 : 02	orchestrateur	UI	Container_Create	UI
evt_3	00 : 03	orchestrateur	UI	Container_Start	UI

Tableau 5.3 – Journal d'événements annoté ($Logs_a$)

	Timestamp	**Source**	src_{name}	**Type**	**Resource**	**SmElt**	**lcStep**
$evta_1$	00 : 01	orchestrateur	UI	Service_Create	UI	State	Start
$evta_2$	00 : 02	orchestrateur	UI	Container_Create	UI	State	Execute
$evta_3$	00 : 03	orchestrateur	UI	Container_Start	UI	State	Complete

5.3.2 Identification d'éléments de machine à états

Une fois les journaux d'événements annotés, nous procédons à l'abstraction d'éléments de machines à état, dénoté $\mathcal{SM}^{abst} = \langle \mathcal{S}, \mathcal{T}\rangle$. $\mathcal{SM}^{abst}$ représente

l'exécution observée d'une stratégie de reconfiguration, comme décrit dans les journaux d'événements. On pourra la comparer à l'exécution prévue, décrite par une machine à état, afin de vérifier la conformité post-exécution. Pour générer une machine à état abstraite $\mathcal{SM}^{abst}$, nous considérons 3 éléments identifiables depuis les journaux d'événements : les *états* (Définition. 3.5), les *événements déclencheurs* (Définition 3.8) et les *actions de reconfiguration* (Définition. 3.7). Ces éléments sont identifiés grâce à la détection de motifs spécifiques d'événements dans un journal d'événements annotés ($Logs_a$). Un motif d'événement consiste en une suite d'événements spécifiques caractérisés par leur type, leur horodatage ou tout autre attribut complémentaire de l'événement.

Dans la suite, nous présentons d'abord notre algorithme pour l'abstraction des journaux d'événements en machines à états (Section 5.3.2.1), puis les 4 motifs que nous avons définis (Section 5.3.2.2 à 5.3.2.5) et un exemple (Section 5.3.2.6).

5.3.2.1 Algorithme

Notre approche pour identifier une machine à états abstraite depuis un journal d'événements annotés $Logs_a$ est décrite dans l'algorithme 10. Cet algorithme prend en entrée un journal d'événements annoté ($Logs_a$) et un seuil de délai entre les événements ($delay_{th}$), spécifié par un expert, représentant une fenêtre temporelle dans laquelle les événements qui s'y déroulent peuvent être associés à un même élément de machine à état (i.e. état, action de reconfiguration et événement déclencheur).

Nous itérons pour chaque événement annoté (evt_a) du journal d'événements annoté (Log_a) (ligne 1-5) et appliquons consécutivement les différents motifs définis précédemment pour identifier les états abstraits (S^{abst}) avec les motifs 5.3.1 et 5.3.2, les événements déclencheurs (TE^{abst}) avec le motif 5.3.3 et les actions de reconfiguration (RA^{abst}) avec le motif 5.3.4. Ensuite, nous parcourons tous les éléments composant les transitions abstraites pour les combiner (ligne 6-11) en fonction de leur identifiant (id). Enfin, la machine à état abstraite ($\mathcal{SM}^{abst}$) combinant les états abstraits (S^{abst}) et les transitions abstraites (T^{abst}) est retournée par l'algorithme(ligne 12).

Dans la suite de cette section, nous présentons chacun des différents motifs que nous avons définis pour l'abstraction d'éléments de machine à état.

5.3.2.2 Identification d'état abstrait

Un groupe d'événements représentant un état abstrait (s_i^{abst}) est identifié à travers une séquence spécifique d'événements, définie comme étant un motif d'événement, avec des valeurs d'attributs composant ces événements tels que l'horodatage, ou les métriques. Ce motif est décrit dans le motif 5.3.1 et illustré à travers un schéma dans la Figure 5.7.

Algorithme 10 Abstraction de journal d'événement

Entrée: $Logs_a$: journal d'événements annoté
$delay_{th}$: seuil de fenêtre de temps
Sortie: $\mathcal{SM}^{abst} = \langle \mathcal{S}^{abst}, \mathcal{T}^{abst} \rangle$: machine à état abstraite
▷ Application des motifs sur $Logs_a$
for each $evt_a \in Logs_a$ **do**
  $S^{abst} \leftarrow$ Application du motif 5.3.1 et du motif 5.3.2
  $TE^{abst} \leftarrow$ Application du motif 5.3.3
  $RA^{abst} \leftarrow$ Application du motif 5.3.4
end for
▷ Combine les éléments abstrait pour construire $\mathcal{SM}^{abst}$
for each ($te^{abst} \in TE^{abst}$ and $ra^{abst} \in RA^{abst}$) **do**
  if ($te^{abst}.id == ra^{abst}.id$) **then**
    $t^{abst} \leftarrow te^{abst} \cup ra^{abst}$
    $T^{abst} \leftarrow T^{abst} \cup t^{abst}$
  end if
end for
$\mathcal{SM}^{abst} \leftarrow \langle S^{abst}, T^{abst} \rangle$
Return $\mathcal{SM}^{abst}$

À partir d'un journal d'événements annoté ($Logs_a$), si nous identifions un ensemble d'événements (noté A) associés à la même ressource, relatifs à un état ($smElt = state$) commençant par l'étape du cycle de vie Start ($lcStep = Start$), se terminant par l'étape du cycle de vie Complete ($lcStep = Complete$) et dans une fenêtre de temps définie, alors on considère que ce groupe de ressources peut-être abstrait en un état. On lui affecte un numéro d'état noté i que l'on retrouve dans le nom de l'état l_s, on extrait les ressources composant l'état (noté R) depuis les métriques présentées dans l'ensemble d'événements. Toutefois, le type d'état ne pouvant pas être identifié immédiatement, car il doit être comparé aux autres états qui l'entourent, est noté $\perp$ pour vide et sera déterminé à postériori avec le motif 5.3.2.

Motif 5.3.1 *(Abstraction d'état)*
$\forall\ evt \in A \subset Logs_a,$
$(evt_m.src_{name} == evt_{(m+1)}.src_{name}) \wedge (evt_m.smElt == \texttt{State}) \wedge$
$((evt_{|A|}.ts - evt_m.ts) \leq delay_{th}) \wedge (evt_m.lcStep == \texttt{Start}) \wedge$
$(evt_{(m,m\neq 0,m\neq |A|)}.lcStep == \texttt{Execute}) \wedge (evt_{|A|}.lcStep == \texttt{Complete})$
$\Longrightarrow s_i^{abst} = (l_s, l_t, R), l_s =$ "s_i^{abst}", $l_t = \perp, R = R \cup evt_{(m...n)}.M$

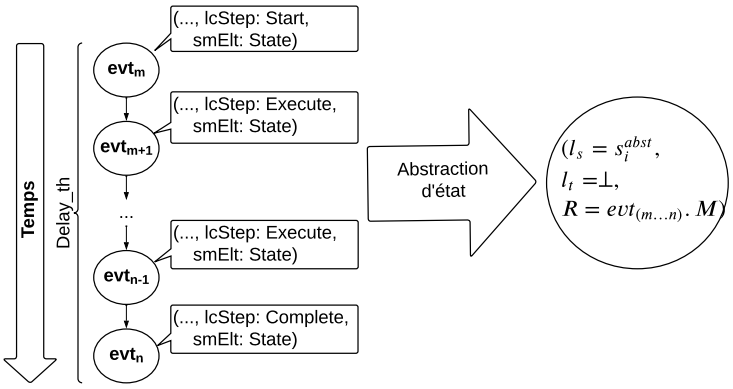


FIGURE 5.7 – Schéma descriptif du motif d'abstraction d'état

Par exemple, dans le Tableau 5.4, nous présentons différents événements $evt_{(m...n)}$ représentant un état selon le motif 5.3.1.

En effet, les événements sont (i) tous liés à la même ressource ($evt_{m...n}.src_{name}$ $=UI$), (ii) dans une fenêtre de temps inférieure ou égale au seuil de la fenêtre définie ($(evt_n.ts - evt_m.ts) \leq delay_{th}$), (iii) liés au même élément de la machine à état ($evt_{(m...n)}.smElt$ = `State`) et (iv) l'étape du cycle de vie ($evt_m.lcStep$) du premier événement est égale à `Start`, celui du dernier événement ($evt_n.lcStep$) est égal à `Complete` et tous les événements entre $(evt_i.lcStep, m > i < n)$ sont égaux à `Execute`. De plus, les caractéristiques des états abstraits ($s_i^{abst}.R$) sont également identifiés depuis ces événements, i.e. depuis leurs métriques ($evt_{(m...n)}.M$). En conclusion, nous obtenons l'état abstrait suivant $\langle l_s = S_i^{abst}, l_t = \perp, R = [r] \rangle$.

TABLEAU 5.4 – Evénements identifié en utilisant le motif 5.3.1

	Timestamp	...	src_{name}	M	lcStep	smElt
m	00 :03	...	UI	$[replicas.new = 8,$ $replicas.old = 0]$	**Start**	**State**
m+1	00 :04	...	UI	/	**Execute**	**State**
				...		
n-1	01 :01	...	UI	/	**Execute**	**State**
n	01 :02	...	UI	/	**Complete**	**State**

5.3.2.3 Identification des types d'état abstrait

Comme défini dans le dernier motif, le type d'état ($isInitial$, $isNormal$, $isFinal$) ne peut pas être identifié sans être comparé aux autres états. Ainsi, identifier un type d'état abstrait l_t ne peut se faire qu'une fois tous les états abstraits identifiés. En effet, le type de l'état dépend du positionnement de l'état par rapport aux autres. Il existe 3 types possibles d'état abstrait ($s_i^{abst}.l_t$) : (i) s'il n'existe aucun prédécesseur

alors son type est *initial* ($s_i^{abst}.l_t$ = `isInitial`), (*ii*) s'il existe au moins un successeur et un prédécesseur alors son type est $Normal$ ($s_i^{abst}.l_t$ = `isNormal`) et (*iii*) s'il n'existe aucun successeur alors son type est $final$ ($s_i^{abst}.l_t$ = `isFinal`). Le motif d'identification est décrit dans le motif 5.3.2.

Motif 5.3.2 *(Abstraction de type d'état)*
$\forall\ s_i^{abst} \in S^{Abst}$,

$$s_i^{abst}.l_t = \begin{cases} isInitial & if \nexists\ s_{(i-1)}^{abst} \in S^{Abst} \\ isNormal & if \exists\ s_{(i-1)}^{abst}, s_{(i+1)}^{abst} \in S^{Abst} \\ isFinal & if \nexists\ s_{(i+1)}^{abst} \in S^{Abst} \end{cases}$$

Par exemple, considérons 3 états identifiés présenté dans la Figure 5.8 : S_1^{abst}, S_2^{abst} et S_3^{abst}. En utilisant le motif 5.3.2, nous cherchons à identifier le type de ces

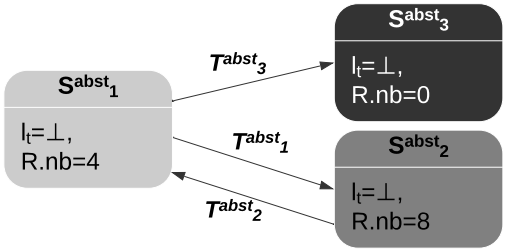


FIGURE 5.8 – Exemple de 3 états identifiés

trois états abstraits. L'état S_1^{abst} n'a pas d'état qui le précéde, son type est donc initial : $s_1^{abst}.l_t$ = `isInitial`. L'état S_2^{abst} a un d'état qui le précéde et un état qui le succède, son type est donc normal : $s_2^{abst}.l_t$ = `isNormal`. L'état S_3^{abst} n'a pas d'état qui le succède, son type est donc final : $s_3^{abst}.l_t$ = `isFinal`.

5.3.2.4 Identification d'événements déclencheurs abstrait

Afin d'identifier les événements déclencheurs (te^{abst}) associés à une machine à état abstrait, nous définissons un motif d'identification d'événements déclencheurs des transitions d'une machine à état analysant la séquence d'événements précédant un état abstrait. Cette identification consiste en un ensemble d'événements (noté A) associés à la même ressource, relative à une transition ($smElt = Transition$), dans une fenêtre de temps définie ($Delay\_th$), alors on considère que ce groupe de ressources peut-être abstrait en une transition et précédant un état détecté. Le motif d'identification est décrit dans le motif 5.3.2.4 et sous forme de schéma dans la Figure 5.9. Cette figure présente 3 evénéments (evt_j, evt_{m-1} et evt_m) associé à l'élément de machine à état T̈ransition(̈$smElt = Transition$) se déroulant dans une même fenêtre de temps ($delay\_th$) qui permettent d'indentifier une événement déclencheur (te_i^{abst}).

Motif 5.3.3 *(Abstraction d'événement déclencheur)*
$\forall\ evt \in A \subset Logs_a$,
$(evt_j.src_{name} == evt_{(j+1)}.src_{name}) \wedge (evt_j.smElt == \texttt{Transition}) \wedge$
$(evt_{m-1}.smElt == \texttt{Transition}) \wedge (evt_m.smElt == \texttt{State}) \wedge$
$(evt_m.lcStep == \texttt{Start}) \wedge ((evt_{|A|}.ts - evt_0.ts) \leq delay_{th})$
$\Longrightarrow te^{abst} = (id_{te} = \text{“}te_i^{abst}\text{”}, t_{te} = \perp, p = evt_{(j...n)}.M)$

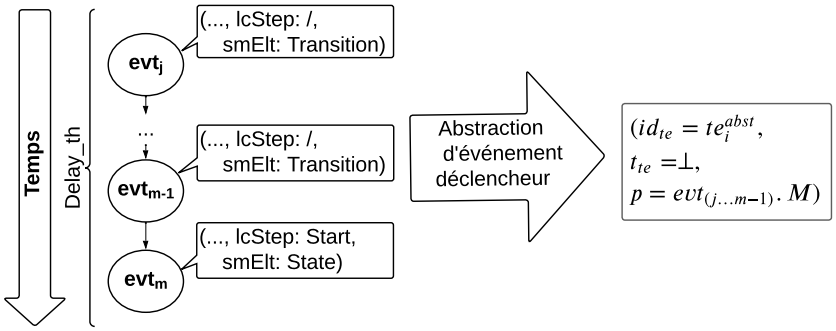


FIGURE 5.9 – Schéma descriptif du motif d'abstraction d'événement déclencheur

Par exemple, le Tableau 5.5 présente une liste d'événement identifié ($evt_{(j...m)}$) représentant un événement déclencheur conformément au motif 5.3.3 où les événements sont : (*i*) liés à la même ressource ($evt_{j...m}.src_{name} = UI$), (*ii*) dans la même fenêtre temporelle qui est inférieur ou égale au seuil défini ($evt_j.ts - evt_m.ts \leq delay_{th}$), (*iii*) associés au même élément de la machine à état ($evt_{(j,...,m-1)}.smElt = \texttt{Transition}$) et (*iv*) précédant un état détecté ($evt_m.smElt = \texttt{State}$, $evt_m.lcStep = \texttt{Start}$). Ce qui donne en suivant la définition 3.9 : $te_1^{abst} = (id_{te} = \text{“}te_1^{abst}\text{”}, t_{te} = HorizontalScaling, p = [UI, scale-out, 8]$.

TABLEAU 5.5 – Evénements identifiés en utilisant le motif 5.3.3

	Timestamp	...	src_{name}	Metrics	lcStep	smElt
j	01 :00	...	UI	[CpuUsage = 15%]	/	Transition
				...		
m-1	02 :00	...	UI	[CpuUsage = 15%]	/	Transition
m	02 :01	...	UI	[$replicas.new = 8$, $replicas.old = 0$]	Start	State

À noter que le type de l'événement déclencheur (t_{te}) va dépendre du type de prédicat (p), qui peut être {*temporal*, *resourceRelated*, *userDefined* ou *composite*}, déduit depuis $evt_{(i...n)}.M$ comme suit, si un événement est lié à un horodatage spécifique initié par le fournisseur de service cloud alors le type d'événement sera temporel $te^{abst}.t_{te} = temporal$, si les événements sont liés à une action utilisateur alors le type d'événement sera défini par un utilisateur

$te^{abst}.t_{te} = userDefined$, si les événements suivent un pic de consommation alors le type d'événement sera liée à la ressource $te^{abst}.t_{te} = resourceRelated$. Si plusieurs événements sont identifiés par le motif alors le type d'événement sera $te^{abst}.t_{te} = composite$.

5.3.2.5 Identification d'action de reconfiguration abstrait

L'identification d'action de reconfiguration (ra^{abst}),voir définition 3.9, est basée sur l'analyse des états abstraits (S^{abst}). Cette identification consiste en (i) l'extraction de deux états consécutif et (ii) la comparaison de leurs caractéristiques (R) pour identifier quel type d'action ($ra^{abst}.t_{ra}$) et quel attribut d'action ($ra^{abst}.AA$) sont requis pour passer d'un état à un autre, tel que, nombre d'instances à ajouter ou supprimer.

Par exemple, pour 2 états abstraits s_1^{abst} avec 4 instances de taille M et s_2^{abst} grâce au motif 5.3.1, avec $s_1^{abst}.R = [(\text{“}UI\text{”}, \text{“}CSP1\text{”}, \text{“}M\text{”}, 4)]$ et $s_2^{abst}.R = [(\text{“}UI\text{”}, \text{“}CSP1\text{”}, \text{“}M\text{”}, 10)]$ avec 10 instances de taille M. La comparaison de ces caractéristiques nous permet de conclure que l'action (ra^{abst}) requise pour passer d'un état à un autre est de passer à l'échelle horizontalement en ajoutant 6 instances. A noter que le type d'action de reconfiguration ($ra^{abst}.t_{ra}$) dépend des attributs d'action ($ra^{abst}.AA$). En effet, nous considérons 2 actions potentiels $\in \{HorizontalScaling\ and\ VerticalScaling\}$, si le nombre d'instance change entre les états alors $t_{ra} = HorizontalScaling$ et si la taille des instances varie alors $t_{ra} = VerticalScaling$.

Motif 5.3.4 *(Abstraction d'action de reconfiguration)*

$$\forall\ s_i^{abst} \in S^{abst}, \exists s_{i+1}^{abst} \in S^{abst} \Longrightarrow$$
$$ra^{abst} = (id_{ra}, t_{ra}, AA), id_{ra} = \text{“}ra_i^{abst}\text{”},$$
$$t_{ra} = \begin{cases} VerticalScaling & if\ s_{(i+1)}^{abst}.R.s \neq s_i^{abst}.R.s \\ HorizontalScaling & if\ s_{(i+1)}^{abst}.R.nb \neq s_i^{abst}.R.nb \end{cases},$$
$$AA = s_{(i+1)}^{abst}.R - s_i^{abst}.R$$

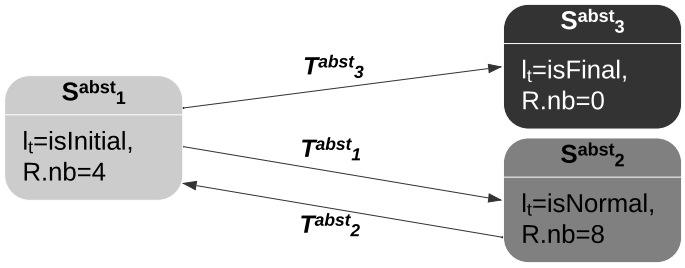
5.3.2.6 Exemple

Pour illustrer l'application des motifs définis, le Tableau 5.6 illustre un journal d'événements pré-traité avec 20 événements annotés. Grâce aux différents motifs, nous pouvons identifier depuis le journal d'événements les états abstraits : S_1^{abst} dans les lignes 1 à 3 et 13 à 15, S_2^{abst} dans les lignes 6 à 8 et S_3^{abst} dans les lignes 18 à 20 ; et les transitions : T_1^{abst} dans les lignes 4 à 5, T_2^{abst} dans les lignes 9 à 12 et T_3^{abst} dans les lignes 16 à 17. La Figure. 5.10 illustre la machine à état abstraite associée avec 3 états (S_1^{abst}, S_2^{abst} et S_3^{abst}) et 3 transitions (T_1^{abst}, T_2^{abst} et T_3^{abst}) décrite en détail dans le tableau.

À l'issue de cette première étape de pré-traitement, nous obtenons à l'issue de la première phase d'annotation des journaux d'événements pré-traités (Tableau 5.6)

TABLEAU 5.6 – Journal d'événements pré-traité

	Timestamp	...	src_{name}	Metrics	lcStep	smElt
1	00 : 01	...	UI	[$replicas.new = 4$, $replicas.old = 0$]	Start	State
2	00 : 03	...	UI	/	Execute	State
3	00 : 05	...	UI	/	Complete	State
4	00 : 17	...	UI	[CpuUsage = 95%]	/	Transition
				...		
5	01 : 17	...	UI	[CpuUsage = 95%]	/	Transition
6	01 : 18	...	UI	[$replicas.new = 8$, $replicas.old = 4$]	Start	State
7	01 : 19	...	UI	/	Execute	State
8	01 : 20	...	UI	/	Complete	State
9	01 : 21	...	UI	[CpuUsage = 15%]	/	Transition
10	01 : 26	...	UI	[CpuUsage = 6%]	/	Transition
11	01 : 31	...	UI	[CpuUsage = 9%]	/	Transition
				...		
12	02 : 31	...	UI	[CpuUsage = 10%]	/	Transition
13	02 : 31	...	UI	[$replicas.new = 4$, $replicas.old = 8$]	Start	State
14	02 : 32	...	UI	/	Execute	State
15	02 : 33	...	UI	/	Complete	State
16	10 : 00	...	UI	[CpuUsage = 25%]	/	Transition
			...			
17	12 : 00	...	UI	[CpuUsage = 25%]	/	Transition
18	12 : 01	...	UI	[$replicas.new = 0$, $replicas.old = 4$]	Start	State
19	12 : 02	...	UI	/	Execute	State
20	12 : 03	...	UI	/	Complete	State



l_t	s_s	s_t	*E*	*A*
T^{abst}_1	S^{abst}_1	S^{abst}_2	resourceRelated,{cpuUsage, average, >=,80,%,60s}	HorizontalScalingAction, {scale-out, 4}
T^{abst}_2	S^{abst}_2	S^{abst}_1	resourceRelated,{cpuUsage, average,<=,15,%,60s}	HorizontalScalingAction, {scale-in, 4}
T^{abst}_3	S^{abst}_1	S^{abst}_3	TP-Event,{12:00}	HorizontalScalingAction, {scale-in, 4}

FIGURE 5.10 – Machine à état abstraite

pouvant être utilisés pas la seconde phase d'abstraction qui donne une machine à état abstraite (Figure 5.10) pouvant être utilisé par le module vérificateur utilisant les algorithmes de vérification de conformité. Dans la prochaine section, nous présentons notre méthode de vérification de conformité de SLA multi-cloud dynamique.

5.4 Vérification de conformité de SLA multi-cloud dynamique

La vérification de conformité est effectuée à travers le module *Vérificateur* (Figure 5.5). Ce dernier reçoit 2 entrées : la machine à état abstraite ($\mathcal{SM}^{abst}$) et la machine à état définie ($\mathcal{SM}^{defined}$) dans le SLA décrivant les stratégies de reconfiguration convenues. Ces entrées sont utilisées pour identifier si des déviations entre les deux existent. Pour ce faire, nous adoptons une technique de vérification de conformité basée sur l'alignement. Traditionnellement, ces techniques se limitent au flux de contrôle, qui consiste au suivi de l'ordre d'exécution des activités de processus.

Définition 5.3 (Mouvement) *Un mouvement est défini par un couple (evt, act) où : evt est un événement issue d'un journal d'événements et act est activité du modèle de processus. Si les deux éléments correspondent directement, il s'agit d'un mouvement synchrone tel que : (evt=a, act=a). Sinon, il s'agit d'un mouvement asynchrone, symbolisé par "≫", tel que : (evt=a, ≫).*

Définition 5.4 (Alignement) *Un alignement est défini par une séquence de mouvements, défini dans la Définition 5.3, align où : align est un ensemble de mouvements tel que align* $= \langle (evt, act), (evt, act), ... \rangle$.

Toutefois, pour nous, il est également nécessaire de s'assurer que les données de chaque activité sont correctes, c'est-à-dire que les caractéristiques des états et transitions parcourus s'alignent. On parlera alors de perspectives de données. Nous considérons la perspective des données représentant les variables caractérisant les états (Définition. 3.5) et les événements déclencheurs composant les transitions (Définition. 3.8). Dans nos travaux, nous adaptons la technique d'alignement prenant en compte les données dans les processus proposés par Mannhardt et al. [65] en appliquant cette technique à notre définition des stratégies de reconfiguration sous forme de machine à état. L'alignement (Définition 5.4) appliqué à notre contexte consiste à comparer des groupes d'événements dans les journaux pré-traités, représentant les éléments d'une machine à état abstraite $\mathcal{SM}^{abst}$, avec une machine à état définie $\mathcal{SM}^{defined}$ désigné comme étant un *mouvement* (Définition 5.3).

Par exemple, dans le Tableau 5.7, en appliquant les concepts de la fouille de processus présenté dans la section 5.1.1 dans notre contexte un mouvement peut être un état abstrait s_1^{abst} lié à un état défini dans le motif s_1^{def}, qui correspond à un mouvement synchrone $\gamma_0 = \{s_1^{abst}, s_1^{def}\}$. Toutefois, quand un état s_2^{abst} est identifié mais ne correspond pas à un état défini dans $\mathcal{SM}^{defined}$, ceci correspond à un mouvement asynchrone $\gamma_1 = \{s_2^{abst}, \gg\}$.

TABLEAU 5.7 – Exemple de mouvements sans perspective de données

$$\gamma_0 = \frac{s_1^{abst} \mid ...}{s_1^{def} \mid ...}, \quad \gamma_1 = \frac{s_2^{abst} \mid ...}{\gg \mid ...}$$

La perspective de données inclut des variables correspondant aux caractéristiques des états, aux événements ou aux actions de reconfiguration. Par exemple dans le Tableau 5.8, un mouvement synchrone serait $\gamma_2 = \{s_3^{abst}\{R.nb = 4\}, s_3^{def}\{R.nb = 4\}\}$.

TABLEAU 5.8 – Exemple de mouvements avec perspective de données

$$\gamma_2 = \frac{s_3^{abst}\{R.nb = 4\} \mid ...}{s_3^{def}\{R.nb = 4\} \mid ...}$$

Nous considérons, les mouvements suivants comme étant possibles dans un alignement : mouvement synchrone avec des données correctes ou incorrectes, mouvement asynchrone dans les logs seulement ou dans le modèle seulement. Pour sélectionner l'alignement le moins coûteux, chacun est associé à un poids. Ces poids sont définis à travers une fonction de coût κ tel que :
$\forall\ elt^{abst} \in \langle \mathcal{SM}.\mathcal{S}^{abst} \cup \mathcal{SM}.\mathcal{T}^{abst} \rangle\ \ \&\ \ elt^{defined} \in \langle \mathcal{SM}.\mathcal{S}^{defined} \cup \mathcal{SM}.\mathcal{T}^{defined} \rangle$,

$$\kappa(elt^{abst}, elt^{defined}) = \begin{cases} 0 & \text{si mouvement synchrone avec des données correctes} \\ 3 & \text{si mouvement synchrone avec des données incorrectes} \\ 5 & \text{si mouvement asynchrone} \end{cases}$$

Étant donné que plusieurs alignements peuvent exister entre une $\mathcal{SM}^{abst}$ et un $\mathcal{SM}^{defined}$, une comparaison doit être effectuée entre les différents alignements possibles à partir d'une valeur de $fitness$. Cette valeur d'adéquation (*angl.* $fitness$) d'un alignement est calculée avec l'équation 5.2, où : $AlignCost$ est le coût d'alignement calculé sur la base de la fonction de coût κ comme la somme des coûts de tous ses mouvements. Et, $WorstAlignCost$ est la valeur du coût associé au "pire" alignement possible, c'est-à-dire celui qui ne contient que des mouvements asynchrones.

$$fitnessValue = 1 - \frac{AlignCost}{WorstAlignCost} \tag{5.2}$$

Dans notre contexte, nous cherchons à identifier l'alignement optimal, i.e. l'alignement ayant le coût global le plus bas possible après avoir calculé les valeurs de fitness pour plusieurs alignements. Identifier un *alignement optimal* est généralement formulé comme un problème de recherche dans un graphe pondéré dirigé représentant tous les alignements possibles avec des poids sur chaque nœud représentant un mouvement possible [5]. Nous utilisons l'algorithme A* [20] pour identifier les chemins possibles avec le coût le plus bas possible entre deux nœuds. En utilisant A*, le coût

pour chaque nœud v est déterminé par la fonction d'évaluation suivante présentée (équation 5.3) :

$$f(v) = g(v) + h(v) \tag{5.3}$$

où g renvoie le coût du chemin le plus court entre le nœud racine et un nœud courant (v) et h renvoie une estimation du coût du chemin le plus court entre un nœud courant (v) et l'un des nœuds cibles (finaux).

Pour que l'algorithme A* identifie le chemin avec le coût global le plus faible, l'estimation renvoyée par h doit être sous-estimée, c'est-à-dire qu'elle doit être inférieure à l'estimation réelle. Pour ce faire, nous définissons la fonction g en se basant sur la fonction de coût κ et ϵ une petite valeur négligeable permettant de garantir la terminaison de l'algorithme [65] comme suit (équation 5.4) :

$$g(\gamma) = \kappa(\gamma) + \epsilon|\gamma| \tag{5.4}$$

Pour ce faire, nous définissons la fonction h qui retourne $+\infty$ si aucun chemin n'est trouvé et une valeur inférieure au poids du chemin le plus court trouvé (où σ représente un chemin possible et δ une fonction de calcul du poids de chemin) en suivant la définition suivante [4] :

$$h \begin{cases} = +\infty & \text{si aucun chemin n'est trouvé} \\ \leq \delta(\sigma) & \text{si au moins un chemin existe} \end{cases}$$

L'algorithme 11 décrit notre technique de vérification de conformité adaptée à notre stratégie de reconfiguration. Nous construisons d'abord l'espace de recherche (SS), c'est-à-dire le graphe pondéré orienté représentant chaque alignement possible en itérant et en comparant les éléments $elt_x \in \langle \mathcal{SM}.\mathcal{S}^{abst} \cup \mathcal{SM}.\mathcal{T}^{abst} \rangle$ et $elt_y \in \langle \mathcal{SM}.\mathcal{S}^{defined} \cup \mathcal{SM}.\mathcal{T}^{defined} \rangle$ (lignes 2-8). S'ils correspondent, nous ajoutons un mouvement synchrone (elt_x,elt_y) à SS ; sinon, nous ajoutons tous les autres mouvements asynchrones possibles, pondérés à l'aide de la fonction de coût κ. Une fois SS entièrement construit, nous utilisons l'algorithme A* comme décrit précédemment pour identifier l'alignement optimal $\gamma_{optimal}$ (ligne 9). Pour finir, nous renvoyons un rapport de conformité (Rep) contenant cet alignement avec les éventuels déviations ($\gamma_{optimal}$) et sa valeur de fitness associée (ligne 10).

La complexité de l'algorithme est principalement déterminée par la construction de l'espace de recherche, qui nécessite une comparaison de tous les éléments de la machine à état abstraite avec ceux de la machine à état définie. Cette partie de l'algorithme a une complexité temporelle de $O(n \times m)$, où n est le nombre d'éléments dans $\mathcal{SM}.\mathcal{S}^{abst} \cup \mathcal{SM}.\mathcal{T}^{abst}$ et m est le nombre d'éléments dans $\mathcal{SM}.\mathcal{S}^{defined} \cup \mathcal{SM}.\mathcal{T}^{defined}$. Ensuite, l'algorithme utilise A*, dont la complexité dépend du nombre de nœuds et d'arêtes dans l'espace de recherche, généralement exponentielle dans le pire des cas. Cependant, le temps d'exécution n'est pas un problème car l'algorithme est exécuté après la mise en oeuvre du SLA, permettant une vérification asynchrone et détaillée des conformités.

Algorithme 11 Algorithme de vérification de conformité

Entrée: $\mathcal{SM}^{abst}$: Machine à état abstraite
$\mathcal{SM}^{defined}$: Machine à état défini
Sortie: *Rep* : Rapport

▷ Initialisation de l'espace de recherche SS
1: $SS \leftarrow (V, E), V = E = \emptyset$
▷ Construction de l'espace de recherche
2: **for each** $elt_x \in \langle \mathcal{SM}.\mathcal{S}^{abst} \cup \mathcal{SM}.\mathcal{T}^{abst} \rangle$ & $elt_y \in \langle \mathcal{SM}.\mathcal{S}^{defined} \cup \mathcal{SM}.\mathcal{T}^{defined} \rangle$ **do**
▷ Construction de SS en comparant les valeurs de $elt_x.v$ et $elt_y.v$
3: **if** $elt_x.v == elt_y.v$ **then**
4: $SS \leftarrow \{\kappa(elt_x, elt_y) + \epsilon\}$
5: **else**
6: $SS \leftarrow \{\kappa(elt_x, \gg) + \epsilon, \kappa(\gg, elt_y) + \epsilon\}$
7: **end if**
8: **end for**
▷ Recherche de l'alignement optimal entre le nœud initial et les nœuds finaux
9: $\gamma_{optimal} \leftarrow A\_Star(SS.initial, SS.final, h)$
▷ Retourne la valeur de fitness conformément à l'Equation (5.2)
10: $Rep \leftarrow \{1 - (\gamma_{optimal}/\gamma_{worst})\}$ & $\{\gamma_{optimal}\}$
Return *Rep*

Exemple : Nous nous référons à nouveau à notre exemple de motivation (Section 1.4). Nous considérons $\mathcal{SM}^{defined}$ comme décrivant la stratégie de reconfiguration définie composée de quatre états et de sept transitions, ainsi que la machine à états $\mathcal{SM}^{abst}$ abstraite à partir des journaux d'événements collectés, décrite dans la Fig. 5.10. En appliquant l'algorithme 11 afin d'identifier un alignement optimal, nous construisons d'abord l'espace de recherche (SS Fig. 5.11).

Par exemple, considérons la Figure 5.11, le nœud associé au mouvement $\{S_1^{abst}(R.nb = 4), high - demand(R.nb = 4)\}$ la fonction κ retourne 0 pour le mouvement précédent $+ \epsilon$ et le nœud associés au mouvement $\{T\_1\{E = (cpu \geq 80\%, 60s), A = (scale - out, 4)\}, \gg\}$ correspond à la sortie de la fonction κ est $5 + \epsilon$. Notons que ϵ est une valeur négligeable garantissant la fin de la recherche de l'algorithme A*.

Ensuite, nous appliquons l'algorithme A* à l'espace de recherche SS pour trouver un alignement optimal. L'alignement identifié pour notre exemple figure dans le Tableau 5.9.

Enfin, nous renvoyons *Rep* contenant l'alignement optimal et sa valeur de $fitness$ calculée à l'aide de l'équation (5.2) où le coût d'alignement optimal 15 et pire cas 70 comme suit :

$$fitnessValue = 1 - \frac{AlignCost}{WorstAlignCost} = 1 - \frac{15}{70} = 0,8$$

5.5 Mise en œuvre et évaluation

Dans cette section, nous présentons dans un premier temps la mise en œuvre

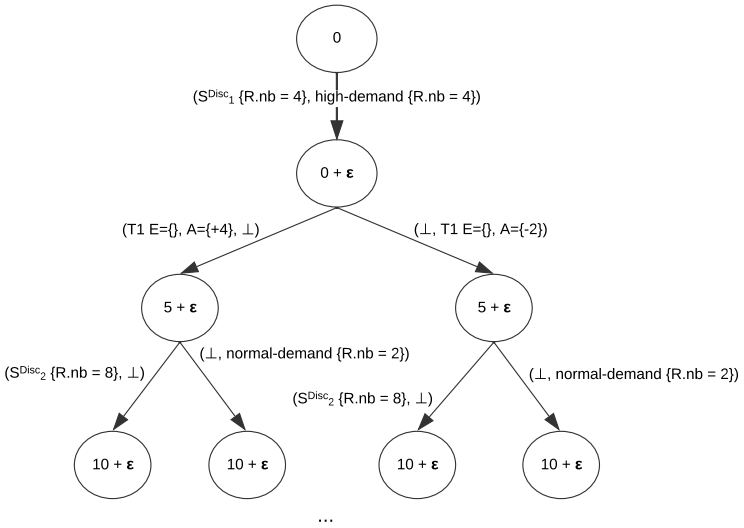


FIGURE 5.11 – Extrait de l'espace de recherche SS pour $\mathcal{SM}^{defined}$ et $\mathcal{SM}^{abst}$

TABLEAU 5.9 – Alignement optimal pour $\mathcal{SM}^{defined}$ et $\mathcal{SM}^{abst}$

Abstrait	**Définie**
S_1^{abst}{R.nb=4}	high-demand{R.nb=4}
T_1{E={cpu $\geq$ 80%,60s}, A={scale-out,4}}	$\gg$
S_2^{abst}{R.nb=8}	$\gg$
T_2{E={cpu $\geq$ 80%,60s}, A={scale-in,4}}	$\gg$
S_1^{abst}{R.nb=4}	high-demand{R.nb=4}
T_3{E={TP-Event{12 :00}}, A={scale-in,4}}	T_6{E={TP-Event{12 :00}} A={scale-in,4}}
S_3^{abst}{R.nb=0}	end{R.nb=0}

du prototype de vérification de conformité puis l'évaluation des deux étapes de l'approche : pré-traitement des journaux d'événements et vérification de conformité.

5.5.1 Mise en œuvre

Cette section est décomposée en deux parties : (*i*) l'architecture du prototype composé de la collecte des journaux d'événements, le pré-traitement des journaux d'événement et l'analyse de conformité ; et (*ii*) un exemple d'exécution du prototype.

5.5.1.1 Architecture du prototype

L'architecture de notre prototype de vérification de conformité de SLA multi-cloud est illustrée dans la Figure 5.12. Le premier composant de ce prototype permet la collecte de journaux d'événements depuis différentes sources (e.g. fournisseurs de services cloud, orchestrateur de ressources et ressources cloud). Puis, le second composant, implémentant les algorithmes 9 et 10 en python, permet un pré-traitement des journaux d'événements collectés en se basant sur notre ontologie modélisée avec Protégé [73]. Enfin, le dernier composant, implémentant en python l'algorithme 11, vérifie la conformité en prenant en entrée les journaux d'événements pré-traités du composant précédent et un SLA multi-cloud dynamique défini en YAML.

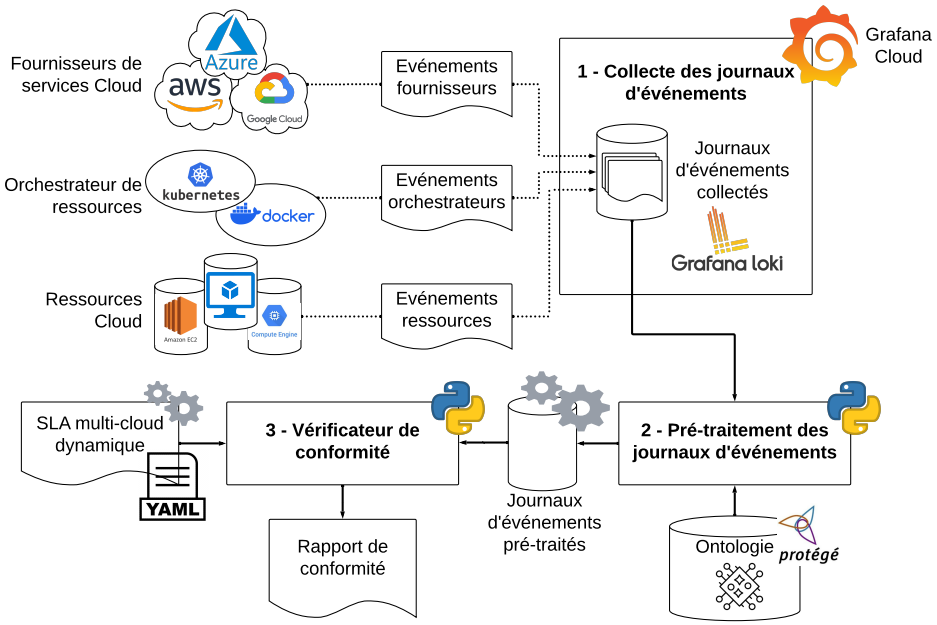


FIGURE 5.12 – Architecture du prototype de vérification de conformité de SLA multi-cloud dynamique

5.5.1.1.1 Collecte des journaux d'événements Nous effectuons la collecte avec Grafana[1], une plateforme open source d'observabilité permettant de fournir

1. `https://grafana.com`

des visualisations dynamiques à partir de données variées, facilitant ainsi la surveillance des performances et la gestion des journaux d'événements. Cet outil permet l'intégration avec une multitude de sources de données, rendant la consolidation des données pour la surveillance à travers des environnements diversifiés possible grâce à des agents de collecte. Nous avons utilisé sa fonctionnalité de collecte des journaux d'événements effectuée en utilisant une solution hébergée avec des agents de collection de journaux d'événements. Ces derniers sont stockés dans une base Grafana Loki [2] depuis laquelle nous pouvons extraire des journaux d'événements. Cette méthode nous permet de collecter les journaux d'événements depuis différents types de sources et d'obtenir un journal d'événement brut combinant ces différentes sources de données présentées préalablement.

5.5.1.1.2 Pré-traitement des journaux d'événements Une fois les journaux d'événements collectés, nous avons développé le composant de pré-traitement en trois étapes. Premièrement, nous avons développé l'ontologie proposée dans la section 5.3.1.1 en utilisant l'éditeur Protégé [73], comme présenté dans la Figure 5.13. Cette ontologie permet de représenter les connaissances du domaine requises pour lier les types d'événements aux éléments des machines à état.

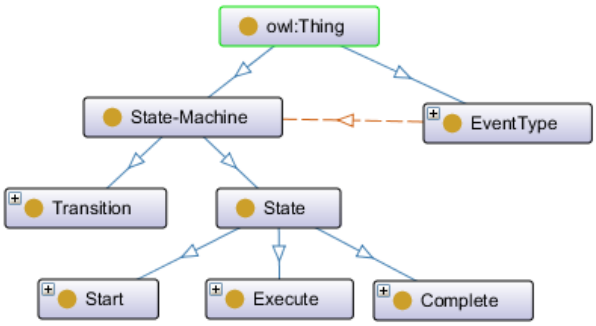


FIGURE 5.13 – Ontologie lié à la représentation de machine à état

Deuxièmement, pour permettre l'identification de machine à état abstraite SM^{abst}, nous avons implémenté l'algorithme 9 permettant d'associer les événements aux concepts définis dans l'ontologie en Python. Cette fonction, comme décrit dans la section 5.3.1, permet d'annoter automatiquement les événements des journaux d'événements collectés en utilisant la librairie *Owlready2* [59]. Cette annotation est faite en fonction du type de l'événement avec sa relation correspondante à un élément de la machine à état ($smElt$) et à son étape dans le cycle de vie de l'élément ($lcStep$).

Troisièmement, nous avons implémenté l'algorithme 10 d'abstraction de machine à état depuis les journaux d'événements annotés. Cet algorithme est basé sur les motifs définis dans la section 5.3.2. Notre implémentation utilise la bibliothèque python *pandas* [3] pour l'analyse des journaux d'événements. L'identification de motif sur plu-

2. `https://grafana.com/docs/loki/latest/`
3. `https://pandas.pydata.org/`

sieurs entrées consécutives n'est pas nativement supporter par cette librairie. Nous avons implémenté une fonction permettant cette identification de motif à travers plusieurs entrées consécutives dans un Dataframe [4] sur le format des motifs présenté dans la section 5.3.2.

5.5.1.1.3 Analyse de conformité Nous avons implémenté la technique d'analyse de la conformité présentée dans la Section 5.1.1.3 (l'algorithme 11) en Python en utilisant la librairie *pm4py* [14]. Le composant implémenté analyse la conformité entre une SM^{Def} et une SM^{abst}. Tout d'abord, nous effectuons la construction d'un espace de recherche contenant tous les alignements possibles entre la SM^{Def} et la SM^{abst}. Puis pour trouver l'alignement optimal dans cet espace de recherche, nous tirons parti de la librairie *pm4py* pour l'implémentation de A*. Enfin, nous retournons un rapport de conformité comportant l'alignement optimal contenant les potentielles déviations, comme décrit dans la Section 5.4.

5.5.1.2 Exemple d'exécution du prototype

Comme décris dans la Section 5.5.1.1.1, la première étape consiste à configurer la collecte des journaux d'événements depuis les différents fournisseurs. Pour cet exemple, nous considérons une extraction effectuée depuis Docker Swarm dont le journal d'événement brut est présenté dans la Figure 5.14.

Raw event-logs

	Timestamp	Source	Resource Name	Event-Type	Metric	Value
0	2023-03-25 00:00:03+00:00	Provider	Auth	Service_Create	replicas	2
1	2023-03-25 00:00:04+00:00	Provider	Auth	Container_Create	/	/
2	2023-03-25 00:00:05+00:00	Provider	Auth	Container_Start	/	/
3	2023-03-25 00:00:05+00:00	Ressource	Auth	Ressource_Usage	Cpu Usage	15
4	2023-03-25 00:01:05+00:00	Ressource	Auth	Ressource_Usage	Cpu Usage	15
..	...	...	...	...	...	...
73	2023-03-25 00:03:05+00:00	Ressource	UI	Ressource_Usage	Cpu Usage	15
74	2023-03-25 00:03:15+00:00	Ressource	UI	Ressource_Usage	Cpu Usage	15
75	2023-03-25 00:03:45+00:00	Provider	UI	Service_Update	replicas	0
76	2023-03-25 00:03:46+00:00	Provider	UI	Container_Stop	/	/
77	2023-03-25 00:03:47+00:00	Provider	UI	Container_Destroy	/	/

FIGURE 5.14 – Extrait de journal d'événement avant annotation

La seconde étape est de remplir l'ontologie correspondante aux différents fournisseurs. Comme décris ultérieurement, nous avons peuplé l'ontologie présentée dans la section 5.3.1.1 avec les différentes informations nécessaires à l'annotation des journaux d'événements issue de Docker Swarm illustré dans la Figure 5.15.

Une fois l'ontologie peuplée, nous pouvons ensuite exécuter l'algorithme d'annotation. Nous exécutons ce dernier sur le journal d'événements brut collecté dans le Figure 5.14 ce qui nous permet d'obtenir un journal d'événements annoté présenté dans la Figure 5.16.

En utilisant le journal d'événements annoté nous allons pouvoir exécuter l'algorithme d'abstraction permettant d'identifier des machines à état depuis les journaux d'événements annotés. Nous avons abstrait une machine à état par service

4. `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`

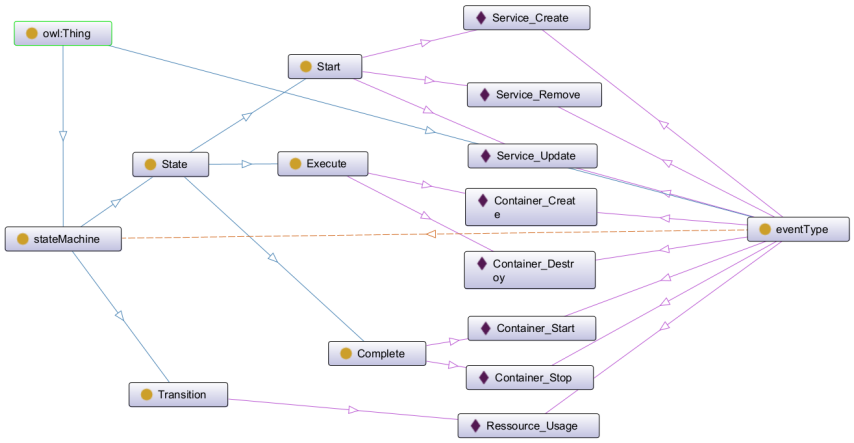


FIGURE 5.15 – Ontologie pour annotation de journaux d'événements collecté depuis Docker Swarm

Annotated logs with state-machine element (smElt) and lifecycle step (lcStep)

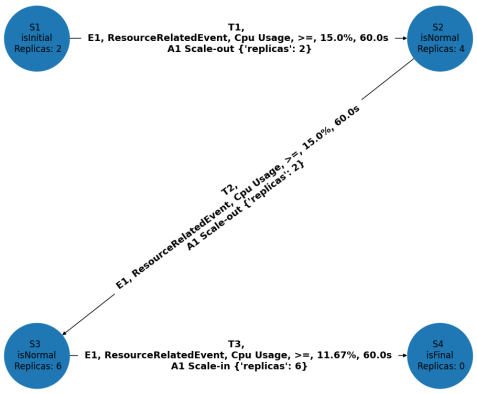
	Timestamp	Source	Resource Name	Event-Type	Metric	Value	smElt	lcStep
0	2023-03-25 00:00:03+00:00	Provider	Auth	Service_Create	replicas	2	State	Start
1	2023-03-25 00:00:04+00:00	Provider	Auth	Container_Create	/	/	State	Execute
2	2023-03-25 00:00:05+00:00	Provider	Auth	Container_Start	/	/	State	Complete
3	2023-03-25 00:00:05+00:00	Ressource	Auth	Ressource_Usage	Cpu Usage	15	Transition	N/A
4	2023-03-25 00:01:05+00:00	Ressource	Auth	Ressource_Usage	Cpu Usage	15	Transition	N/A
..	...	...	...	...	...	...	...	...
73	2023-03-25 00:03:05+00:00	Ressource	UI	Ressource_Usage	Cpu Usage	15	Transition	N/A
74	2023-03-25 00:03:15+00:00	Ressource	UI	Ressource_Usage	Cpu Usage	15	Transition	N/A
75	2023-03-25 00:03:45+00:00	Provider	UI	Service_Update	replicas	0	State	Start
76	2023-03-25 00:03:46+00:00	Provider	UI	Container_Stop	/	/	State	Complete
77	2023-03-25 00:03:47+00:00	Provider	UI	Container_Destroy	/	/	State	Execute

FIGURE 5.16 – Extrait de journal d'événement après annotation

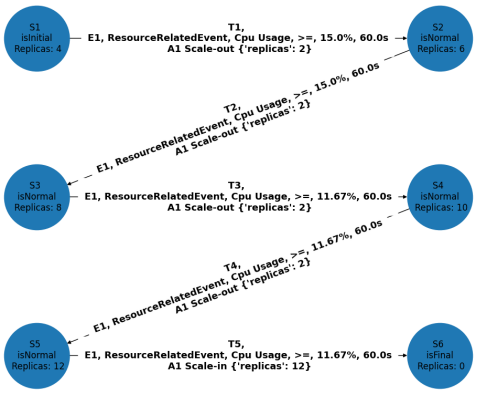
représentant ce qui a été exécuté : UI (Figure 5.17a), Stor (Figure 5.17b) et Auth (Figure 5.17c).

Pour valider le bon fonctionnement de cette implémentation, nous l'avons testé avec une machine à état défini et les machines à état abstraites identifiées dans la section précédente. En analysant ces rapports, nous pouvons voir que les rapports liés aux composants *UI* et *Stor* aboutissent en une exécution quasi similaire avec la machine à état défini. Toutefois, l'exécution du composant *Auth* est totalement différente de la machine à état défini. De plus, ces rapports permettent d'identifier les déviations (incohérence entre les états défini et identifié) de ce qui a été déployé, mais nécessite une étude approfondie pour identifier la cause des déviations. Cette exécution retourne les trois rapports présentés dans la Figure 5.18. Ces rapports sont composés des trois éléments suivants : (*i*) l'Alignment qui présente l'alignement optimal sélectionné, (*ii*) Y_Optimal correspond au coût de l'alignement optimal et (*iii*) le résultat de la valeur de fitness comme décrit dans l'Equation. 5.2.

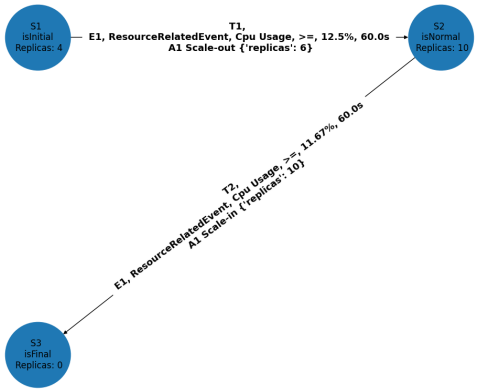
Dans la section suivante, nous évaluons la capacité de notre approche à analyser les journaux d'événements collectés depuis différents fournisseurs de service cloud. Puis, d'évaluer la conformité de ces machines à état abstraite par rapport à une machine à état défini.



(a) Machine à état abstraite pour le composant UI



(b) Machine à état abstraite pour le composant Stor



(c) Machine à état abstraite pour le composant Auth

FIGURE 5.17 – Machines à état abstraite depuis les journaux d'événements collectés

```
Report :
Alignment :
 Defined    Discovered
---------  ------------
S1         S1
T1         T1
S2         S2
T2         T2
S3         S3
Y_Optimal : 5.4
FitnessValue : 0.82
```

(a) Rapport du composant Interface Utilisateur

```
Report :
Alignment :
 Defined    Discovered
---------  ------------
S1         S1
T1         T1
S2         S2
T2         T2
S3         S3
Y_Optimal : 5.4
FitnessValue : 0.82
```

(b) Rapport du composant Stockage

```
Report :
Alignment :
 Defined    Discovered
---------  ------------
S1         >>
T1         >>
S2         >>
Y_Optimal : 15.1
FitnessValue : 0.5
```

(c) Rapport du composant Authentification

FIGURE 5.18 – Rapports résultant de l'exécution

5.5.2 Évaluation

Dans cette section, nous présentons l'évaluation de notre approche sur des journaux d'événements collectés chez différents fournisseurs de service cloud de *AWS*, *Azure* et *GCP*. Nous effectuons tout d'abord l'évaluation du composant de Pré-traitement des journaux d'événements que nous évaluons dans sa capacité à abstraire des éléments de machine à état depuis les journaux d'événements de différents fournisseurs de services cloud. Pour ce faire, nous avons construit un Dataset de journaux d'événements collectés auprès de AWS, Azure et GCP en suivant la même méthodologie que proposé dans [25] qui consiste en la création et la suppression aléatoire de ressource pendant une période de temps donné. Dans la même démarche, nous avons implémenté un script en python similaire permettant de créer et supprimer des ressources chez les différents fournisseurs de services cloud.

Ensuite, nous effectuons l'évaluation du composant de vérification de conformité dans sa capacité à identifier des déviations entre une machine à état abstraite et une machine à état définie entre elles. Pour ce faire, nous avons construit 2 types de scénarios d'évaluation de machines à états dans lesquels nous avons introduit (i) des incohérences aux niveaux des états et (ii) des incohérences au niveau des transitions.

5.5.2.1 Abstraction de machine à état de fournisseurs de service cloud

Dans un premier temps, nous évaluons le composant de pré-traitement des journaux d'événements de notre système dans sa capacité d'abstraction de machine à état à partir des journaux d'événements issus de plusieurs fournisseurs de service cloud. Pour ce faire, nous nous basons sur les journaux d'événements collectés dans Grafana depuis AWS, GCP et Azure. Dans cette expérimentation, nous avons considéré les services de machines virtuelles (IaaS) des trois fournisseurs.

Tableau 5.10 – Résultat de l'évaluation de l'abstraction des journaux d'événements

CSP	Événements collectés	Événements annotés	Éléments de machine à état abstrait
AWS	665	462	227
Azure	1230	258	39
GCP	984	292	146

Le résultat de cette évaluation est résumé dans le Tableau 5.10. Nous avons collecté pour AWS 665 événements, 462 évènements ont été annotés et nous avons pu identifier 227 éléments de machine à état (états et transitions). Pour AWS, nous avons collecté 665 événements, annotés 462 évènements et identifié 227 éléments de machine à état. Pour Azure, nous avons collecté 1230 événements, annotés 258 évènements et identifié 39 éléments de machine à état. Pour GCP, nous avons collecté 984 événements, annotés 292 évènements et identifié 146 éléments de machine à état.

Pour évaluer la pertinence des résultats d'évaluation, une analyse effectuée par

un expert du domaine a examiné les étapes d'annotation et d'abstraction, comme indiqué dans le Tableau 5.11. Nous avons considéré trois métriques principales : le taux d'annotation correcte d'événements, le taux de non-annotation correcte et le taux d'abstraction correcte d'éléments de machines à états. Il en ressort que tous les événements qui devaient être annotés l'ont été, ceux qui ne devaient pas être annotés ne l'ont pas été et tous les éléments de machines à états qui devaient être abstraits l'ont été correctement.

Tableau 5.11 – Analyse de la pertinence des résultats de l'évaluation

	Taux d'annotation correcte	Taux de non-annotation correcte	Taux d'abstraction correcte
AWS	100%	100%	100%
Azure	100%	100%	100%
GCP	100%	100%	100%

Dans cette première étape d'évaluation, nous avons pu voir que notre approche permet l'abstraction de machine à état à partir de journaux d'événements collectés auprès de différents fournisseurs de service cloud.

5.5.2.2 Vérification de la conformité

Dans un second temps, nous évaluons le composant de vérification de conformité. L'objectif est d'évaluer la capacité de notre approche à identifier les déviations entre machines à états définies et les machines à état abstraites. Ce qui reviendra à identifier une différence entre le comportement attendu du système et le comportement réellement observé du système. Pour effectuer cette évaluation, nous définissons un ensemble de machines à états avec des déviations au niveau des états, des événements déclencheurs des transitions et des actions des transitions. Puis, nous analysons les différents résultats de cette évaluation présentée dans le Tableau 5.12. Nous pouvons donc constater que toutes les déviations introduites ont été correctement détectées.

Tableau 5.12 – Résultat de l'évaluation de vérification de conformité

Type de déviation défini	Déviations introduite	Déviations détecté	Taux de détections correcte
États avec caractéristiques incorrectes	30	30	100%
Transitions avec événements et actions incorrectes	15	15	100%

Dans cette dernière partie, nous avons pu étudier la capacité de notre approche à identifier les différents types de déviations pouvant arriver dans le cadre de la mise en œuvre de SLA multi-cloud dynamique.

5.6 Conclusion

Dans ce chapitre, nous avons proposé une méthode de vérification de la conformité post-mise en œuvre des SLAs multi-cloud dynamique, répondant à notre troisième objectif de validation post-mise en œuvre **O2.2**. Nous avons examiné en détail la vérification de conformité entre un SLA multi-cloud dynamique et des journaux d'événements collectés auprès de fournisseurs de service cloud.

Nous avons proposé une approche de vérification de la conformité de SLA multi-cloud dynamique basée sur une technique d'alignement issue de la fouille de processus. Notre approche est décomposée en deux étapes : (*i*) le pré-traitement des journaux d'événements issue de l'environnement multi-cloud et (*ii*) la vérification de la conformité. Le pré-traitement permet de préparer les journaux d'événement et d'en extraire une machine à état qui représentera ce qui s'est réellement déroulé pendant la mise en œuvre. Puis, cela permettra de comparer cette dernière machine à état extraite à la machine à état défini dans le SLA mis en œuvre. Nous avons implémenté un prototype mettant en œuvre ces propositions en Python et évalué les deux aspects de notre approche. Premièrement, le pré-traitement est évalué dans sa capacité à abstraire des journaux d'événements issus des services IaaS des fournisseurs de service cloud : AWS, GCP et Azure. Deuxièmement, l'étape de vérification de la conformité est évaluée dans sa capacité à identifier des déviations entre des machines à états abstraites et des machines à états définies. Les travaux présentés dans ce chapitre ont été publiés dans l'article de conférence suivant : *Jeremy Mechouche, Mohamed Sellami, Zakaria Maamar, Roua Touihri, Walid Gaaloul : Process mining approach for Multi-Cloud SLA Reporting, IEEE International Conference on Big Data, Dec 15-18, 2023, Sorrento, Italy.*

Dans la suite de ces travaux, nous prévoyons de prouvé formellement l'exactitude de nos algorithmes proposé et d'étendre la capacité de détection à un cas en temps réel puis d'analyser les déviations identifiées pour proposer une résolution des déviations détectées.

Chapitre 6

Conclusion et Perspectives

Dans ce chapitre, nous dressons le bilan des travaux réalisés dans le cadre de cette thèse. Nous présentons, d'abord, un rappel des motivations et une synthèse des contributions. Ensuite, nous développons les perspectives à court, moyen et long terme de nos travaux.

6.1 Réponse aux objectifs

À mesure que le cloud computing se développe, de nouveaux besoins clients naissent, surpassant les capacités d'un unique fournisseur. Naturellement, les clients se sont intéressés à la consommation de services de plusieurs fournisseurs, paradigme nommé multi-cloud [77, 79]. Ce paradigme apporte un certain nombre d'avantages, par exemple, l'optimisation des coûts en utilisant les services avec le meilleur rapport qualité/prix ou l'augmentation de la résilience en assurant un plan de continuité/reprise d'activités réparties sur plusieurs fournisseurs. Cependant, bien que ces avantages soient conséquents, le multi-cloud soulève des verrous inhérents à la multiplicité de fournisseurs et à l'hétérogénéité de leurs services. En effet, chaque fournisseur a ses propres types de services et ses propres modèles de description de ces derniers. En outre, la gestion de la qualité de service, le maintien et le suivi des objectifs de niveau de service est rendue d'autant plus complexe en raison de la nature distribuée du contexte multi-cloud et de la dépendance qui existe entre les différents composants [88].

Dans les environnements multi-cloud, la définition et le suivi des objectifs de niveau de service (*angl. Service Level Objective, SLO*) revêtent une importance cruciale afin de fournir un service de qualité aux utilisateurs finaux et de garantir que les services fournis par les différents fournisseurs correspondent aux attentes clients. Les SLOs définissent les attentes en termes de performance, de disponibilité et d'autres critères de qualité pour chaque service cloud. Les SLOs composent les accords de niveaux de services (*angl. Service Level Agreement, SLA*) qui engagent le fournisseur sur un niveau de service à fournir. En raison de la diversité des fournisseurs, des infrastructures et du caractère distribué d'un environnement multi-cloud, la définition et le suivi des SLAs est complexe. Également, pour garantir le respect des SLOs dans

le contexte cloud un élément essentiel est la reconfiguration dynamique des services sous forme de stratégies. Ces stratégies de reconfiguration déterminent comment les ressources sont allouées, ajustées ou libérées en fonction des variations de charge, des pannes et d'autres facteurs. Ces derniers sont d'autant plus complexes à mettre en œuvre dans un contexte multi-cloud. À l'issue de la mise en œuvre des SLAs, il est essentiel de les vérifier pour assurer que les services fournis répondent aux attentes et aux exigences définies. La mise en œuvre de mécanismes de vérification permet non seulement de s'assurer que les SLAs sont respectés, mais aussi d'identifier précisément tout écart potentiel. Cette vérification est d'autant plus complexe dans un contexte de SLA multi-cloud. Par conséquent, l'adoption d'une approche systématique et automatisée pour la validation et le suivi des SLAs est cruciale pour appréhender efficacement la complexité du multi-cloud et pour assurer un service cloud cohérent et fiable.

Dans cette thèse, nous nous sommes donc intéressés à la gestion des SLAs dans le contexte multi-cloud. Plus précisément, nous cherchions à traiter les questions de recherche suivantes : **(RQ1)** Comment modéliser les SLAs multi-cloud considérant la dynamicité des services ? **(RQ2)** Comment vérifier et suivre la mise en œuvre des SLAs multi-cloud considérant la dynamicité des services ?

Pour adresser ces 2 questions de recherche, nous avons défini trois objectifs permettant de répondre à ces problématiques : (1) proposer un modèle de description de SLA multi-cloud dynamique, (2) proposer un processus de validation de la cohérence préalable à la mise en œuvre des SLAs multi-cloud dynamique et (3) proposer un processus de vérification post-mise en œuvre des SLAs multi-cloud dynamique.

Pour atteindre le premier objectif, présenté dans le chapitre 3, nous avons proposé notre modèle de description de SLA Multi-cloud enrichi par des stratégies de reconfiguration représentées sous forme de machine à états. Pour adresser l'aspect multi-cloud, ce modèle est composé d'un *global-SLA* permettant de représenter des objectifs globaux au système multi-cloud et de *sub-SLAs* permettant de représenter des objectifs locaux au niveau des composants d'un système multi-cloud. Nous proposons pour adresser l'aspect dynamique d'intégrer une machine à état permettant la formalisation de stratégie de reconfiguration des services cloud. Nous avons également présenté en détail les 3 éléments composant notre SLA multi-cloud dynamique : le Global-SLA, les Sub-SLAs et les stratégies de reconfiguration. Ensuite, notre approche a été proposée en tirant parti du format YAML. Pour finir, l'expressivité du modèle a pu être évalué en le comparant aux concepts couverts par les autres langages de description de SLA de l'état-de-l'art.

Pour atteindre le second objectif, présenté dans le chapitre 4, nous avons proposé une méthode de validation de cohérence pré-mise en œuvre des SLAs multi-cloud dynamique répondant à notre second objectif de validation pré-mise en œuvre des SLAs multi-cloud dynamique. Nous avons proposé une méthode pour la validation de cohérence dans un SLA multi-cloud dynamique entre : (*i*) les global-SLOs et les sub-SLOs et (*ii*) les sub-SLOs et leurs stratégies de reconfiguration associées. Notre méthode de validation (*i*) entre global-SLOs et sub-SLOs basé sur une tech-

nique d'agrégation de SLO, est décomposée en deux phases : descendante (Global-SLOs vers sub-SLOs) puis ascendante (Sub-SLOs vers Global-SLOs) pour permettre d'identifier les incohérences dans les deux sens. Cette double validation permet également de ne pas favoriser dans l'analyse ni les sub-SLOs, ni les global-SLOs. Notre méthode de validation (*ii*) entre sub-SLOs et stratégie de reconfiguration basée sur la traduction de SLO, permet une validation à deux niveaux d'abstraction différents : *fonctionnel* pour les caractéristiques des sub-SLOs et *opérationnel* pour les événements déclencheurs des stratégies de reconfiguration. Par la suite, nous avons implémenté cette méthode sous forme d'un prototype en Python que nous avons évalué avec différents scénarios de SLA multi-cloud incohérents à différents niveaux de ce dernier : Global-SLA, Sub-SLA et stratégie de reconfiguration. Cette évaluation nous a permis de démontrer l'efficacité de notre méthode qui a permis d'identifier la majorité des incohérences introduites.

Pour atteindre le troisième objectif, présenté dans le chapitre 5, nous avons proposé une méthode de vérification de la conformité post-mise en œuvre des SLAs multi-cloud dynamique, répondant à notre troisième objectif de validation post-mise en œuvre **O2.2**. Nous avons examiné en détail la vérification de conformité entre un SLA multi-cloud dynamique et des journaux d'événements collectés auprès de différents fournisseurs de service cloud. Nous avons proposé une approche de vérification de la conformité basée sur une technique d'alignement issue de la fouille de processus. Notre approche est décomposée en deux étapes : (*i*) le pré-traitement des journaux d'événements et (*ii*) la vérification de la conformité. Le pré-traitement permet de préparer les journaux d'événement et d'en extraire une machine à état qui représentera ce qui s'est réellement déroulé pendant la mise en œuvre. Puis, cela permettra de comparer cette dernière machine à état extraite à la machine à état défini dans le SLA mis en œuvre. Nous avons implémenté un prototype mettant en œuvre ces propositions en Python et évalué les deux aspects de notre approche. Premièrement, le pré-traitement est évalué dans sa capacité à abstraire des journaux d'événements en machine à état issue des services IaaS des fournisseurs de service cloud : AWS, GCP et Azure. Deuxièmement, l'étape de vérification de la conformité est évaluée dans sa capacité à identifier des déviations entre des machines à états abstraites et des machines à états définis.

6.2 Perspectives

Notre travail ouvre plusieurs perspectives de recherches à court, moyen et long terme. Nous nous concentrerons sur l'extension des propositions faites dans les 3 chapitres présentant les contributions de cette thèse, tout en examinant également d'autres problématiques de recherche connexes. Dans la section 6.2.1, nous commençons par la construction d'un assistant de modélisation de SLA multi-cloud dynamique. Ensuite, dans la section 6.2.2, nous présentons comment nous pourrions étendre notre approche de vérification de conformité. Puis, dans la section 6.2.3, nous traiterons une extension de la brique d'abstraction du chapitre 5. Pour finir, dans

la section 6.2.4, nous proposons d'étendre nos approches pour les SLAs multi-cloud dynamique au contexte continuum computing.

6.2.1 Assistant de modélisation de SLA multi-cloud dynamique

À court terme, nous prévoyons d'utiliser les différents concepts proposés notamment pour la vérification pré-mise en œuvre de SLA multi-cloud pour construire un assistant intelligent pour la modélisation de SLA multi-cloud dynamique. Dans la lignée initiale de nos travaux, l'objectif est d'assister les architectes cloud et de leur mettre à disposition des outils efficients pour la construction des SLOs de leur système multi-cloud qui seront garantis à leurs clients. Le principal verrou de cette perspective sera dans la population des différentes bases de connaissance de nos travaux. Nous considérons la mise en place de solutions automatique de collecte de données pour faciliter la collecte et limiter l'intervention humaine [38, 69].

6.2.2 Extension de la vérification de conformité

À moyen terme, nous prévoyons d'étendre les travaux à un contexte plus large de différents fournisseurs de service cloud et d'investiguer la mise en place de la vérification de conformité en temps réel des SLAs multi-cloud. En effet, dans ces travaux, nous nous sommes principalement intéressés à l'analyse à froid (ou post-mortem) des journaux d'évènements, dans la suite de nos travaux, nous prévoyons d'intégrer une vérification de conformité en temps réel. Cela nécessitera de faire évoluer la brique de pré-traitement avec des techniques d'analyse de flux de données [19] (angl. stream processing) pour identifier des éléments de machines à état en temps réel. Egalement, de faire évoluer la brique de vérification de conformité pour lui permettre de supporter la vérification en temps réel avec toujours une technique issue de la fouille de processus : la vérification de conformité en temps réel [17] (angl. online conformance checking).

6.2.3 Extension de la brique d'abstraction

À long terme, nous prévoyons d'étendre les travaux de la dernière contribution de découverte de machine à états dans les journaux d'événements collectés auprès des fournisseurs de service cloud. L'objectif serait d'étendre nos travaux à la découverte de système cloud complet à partir de différentes sources de données. En effet, dans nos travaux nous nous sommes concentrés sur l'identification de machines à état dans des journaux d'événements. Toutefois, cette approche pourrait être utilisée pour identifier d'autres modèles de description de système cloud et à partir d'autres sources de données sans se limiter aux journaux d'événements. Cela permettrait entre autres de faire de la découverte de système cloud pour générer des schémas d'architecture automatique d'un instant donnée du système [84] ou de faire de la reconnaissance dans des scénarios d'audit de sécurité cloud [67]. Une première piste envisagée serait d'adapter des techniques de modèle learning [6] qui désigne

un groupe d'algorithmes conçus pour l'apprentissage des modèles (sous forme de diagrammes d'état) de systèmes à partir des données d'entrées/sorties observées.

6.2.4 Gestion de SLA multi-cloud appliqué au continuum computing

À long terme, nous envisageons d'appliquer nos approches au continuum computing, intégrant cloud, fog et edge computing pour une gestion fluide des charges de travail. Modélisation de SLA dans le continuum computing : Notre modèle de SLA multi-cloud dynamique sera étendu pour inclure les ressources fog et edge, intégrant des objectifs globaux et locaux et des stratégies de reconfiguration pour déplacer les charges de travail selon les besoins. Validation pré-mise en œuvre : La validation de cohérence devra gérer les ressources limitées des environnements fog et edge, en améliorant les algorithmes d'agrégation et de traduction des SLA pour traiter cette complexité accrue. Vérification post-mise en œuvre : La vérification de conformité inclura des données provenant des dispositifs edge, des nœuds fog et des infrastructures cloud, en utilisant des techniques de fouille de processus en temps réel pour assurer une surveillance proactive des SLA. En conclusion, l'intégration des ressources du continuum computing permettra de fournir des services plus flexibles et résilients, répondant aux besoins évolutifs des utilisateurs finaux. Ces développements contribueront à la création de systèmes de gestion de SLA plus robustes et intelligents, adaptés aux environnements de calcul distribués de nouvelle génération.

Bibliographie

[1] Christoph "Fehling et al. *"Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications"*. "Springer", 2014.

[2] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016, p. 3-452. isbn : 978-3-662-49851-4.

[3] Arya Adriansyah, Boudewijn F van Dongen et Wil MP van der Aalst. « Conformance checking using cost-based fitness analysis ». In : *2011 ieee 15th international enterprise distributed object computing conference* (2011), p. 55-64.

[4] Arya Adriansyah, Boudewijn F van Dongen et Wil MP van der Aalst. « Memory-efficient alignment of observed and modeled behavior ». In : (2013).

[5] Arya Adriansyah et al. « Alignment based precision checking ». In : *International conference on business process management*. Springer. 2012, p. 137-149.

[6] Simone Agostinelli et al. « Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis ». In : *Information Systems* 114 (2023), p. 102180.

[7] Hamda Al-Ali et al. « A composite machine-learning-based framework for supporting low-level event logs to high-level business process model activities mappings enhanced by flexible BPMN model translation ». In : *Soft Computing* 24 (2020), p. 7557-7578.

[8] Eman Aljoumah et al. « SLA in cloud computing architectures: A comprehensive study ». In : *Int. J. Grid Distrib. Comput* 8.5 (2015), p. 7-32.

[9] Ali Alzubaidi, Karan Mitra et Ellis Solaiman. « A blockchain-based SLA monitoring and compliance assessment for IoT ecosystems ». In : *Journal of Cloud Computing* 12.1 (2023), p. 50.

[10] Alain Andrieux et al. « Web services agreement specification (WS-Agreement) ». In : *Open grid forum*. Citeseer. 2007.

[11] Danilo Ardagna et al. « MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds ». In : *2012 4th International Workshop on Modeling in Software Engineering (MISE)* (2012). url : `https://ieeexplore.ieee.org/document/6226014/`.

[12] Nicolò Bartelucci et Paolo Bellavista. « A Practical Guide to Autoscaling Solutions for Next Generation Internet Applications ». In : *2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom)*. 2023, p. 627-631. doi : 10.1109/MetaCom57706.2023.00110.

[13] Alessandro Berti et Wil MP van der Aalst. « Reviving Token-based Replay: Increasing Speed While Improving Diagnostics. » In : *ATAED@ Petri Nets/ACSD* 2371 (2019), p. 87-103.

[14] Alessandro Berti, Sebastiaan J van Zelst et Wil van der Aalst. « Process mining for python (PM4Py): bridging the gap between process-and data science ». In : *arXiv preprint arXiv:1905.06169* (2019).

[15] Beyer Betsy et al. *The Site Reliability Workbook. Practical Ways to Implement SRE*. O'Reilly Media, Inc, 2018.

[16] Antonio Brogi et al. « Adaptive management of applications across multiple clouds: The SeaClouds Approach. » In : *CLEI Electron. J.* 18 (2015).

[17] Andrea Burattin et Josep Carmona. « A framework for online conformance checking ». In : *Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers 15*. Springer. 2018, p. 165-177.

[18] Erik Christensen et al. *Web services description language (WSDL) 1.1*. 2001.

[19] Gianpaolo Cugola et Alessandro Margara. « Processing flows of information: From data stream to complex event processing ». In : *ACM Computing Surveys (CSUR)* 44.3 (2012), p. 1-62.

[20] Rina Dechter et Judea Pearl. « Generalized best-first search strategies and the optimality of A ». In : *Journal of the ACM (JACM)* 32.3 (1985), p. 505-536.

[21] *DECIDE - Multicloud Application Towards the Digital Single Market*. 2020. url : https://decide-h2020.eu/.

[22] Sharmi Dev Gupta, Begüm Genç et Barry O'Sullivan. « Explanation in constraint satisfaction: A survey ». In : International Joint Conferences on Artificial Intelligence Organization. 2021.

[23] Yahya Al-Dhuraibi et al. « Elasticity in cloud computing: state of the art and research challenges ». In : *IEEE Transactions on services computing* 11.2 (2017), p. 430-447.

[24] Jianru Ding et al. « Characterizing service level objectives for cloud services: Realities and myths ». In : *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019, p. 200-206.

[25] Min Du et al. « Deeplog: Anomaly detection and diagnosis from system logs through deep learning ». In : *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2017, p. 1285-1298.

[26] Marlon Dumas et al. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018, p. 1-527. ISBN : 978-3-662-56508-7.

[27] Sebastian Dunzer et al. « Conformance checking: a state-of-the-art literature review ». In : *Proceedings of the 11th international conference on subject-oriented business process management* (2019), p. 1-10.

[28] Simon Dupont et al. « Experimental analysis on autonomic strategies for cloud elasticity ». In : *2015 International Conference on Cloud and Autonomic Computing*. IEEE. 2015, p. 81-92.

[29] Simon Dutkowski. *Deliverable D3.15: Final multi-cloud native application composite CSLA definition*. 2019. URL : `https://www.decide-h2020.eu/content/deliverables`.

[30] Abdessalam Elhabbash et al. « Slo-ml: A language for service level objective modelling in multi-cloud applications ». In : *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 2019, p. 241-250.

[31] Vincent C Emeakaroha et al. « Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments ». In : *2010 International Conference on High Performance Computing & Simulation*. IEEE. 2010, p. 48-54.

[32] Robert Engel et al. « ysla: reusable and configurable SLAs for large-scale SLA management ». In : *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)* (2018), p. 317-325.

[33] Funmilade Faniyi et Rami Bahsoon. « A systematic review of service level management in the cloud ». In : *ACM Computing Surveys (CSUR)* 48.3 (2015), p. 1-27.

[34] Mostafa Vakili Fard et al. « Resource allocation mechanisms in cloud computing: a systematic literature review ». In : *IET Software* 14.6 (2020), p. 638-653.

[35] Soodeh Farokhi. « Towards an SLA-based service allocation in multi-cloud environments ». In : *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE. 2014, p. 591-594.

[36] Soodeh Farokhi et al. « Hierarchical SLA-based service selection for multi-cloud environments ». In : *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (2014).

[37] Nicolas Ferry et al. « Managing multi-cloud systems with CloudMF ». In : *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*. 2013, p. 38-45.

[38] Divya Natolana Ganapathy et Karuna Pande Joshi. « A semantically rich framework to automate cloud service level agreements ». In : *IEEE Transactions on Services Computing* 16.1 (2022), p. 53-64.

[39] Svetlana Girs et al. « A systematic literature study on definition and modeling of service-level agreements for cloud services in IoT ». In : *IEEE Access* 8 (2020), p. 134498-134513.

[40] Brabra Hayet et al. « Toward higher-level abstractions based on state machine for cloud resources elasticity ». In : *Information Systems* (2020).

[41] Leonard Heilig, Eduardo Lalla-Ruiz et Stefan Voss. « A cloud brokerage approach for solving the resource management problem in multi-cloud environments ». In : *Computers & Industrial Engineering* 95 (2016), p. 16-26.

[42] Nikolas Roman Herbst, Samuel Kounev et Ralf Reussner. « Elasticity in cloud computing: What it is, and what it is not ». In : *10th international conference on autonomic computing (ICAC 13)*. 2013, p. 23-27.

[43] Jennifer Horkoff et al. « Evaluating modeling languages: an example from the requirements domain ». In : *Conceptual Modeling: 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings 33*. Springer. 2014, p. 260-274.

[44] ISO. *Information technology — Cloud computing — Service level agreement (SLA) framework — Part 1: Overview and concepts*. Standard. International Organization for Standardization, 2016.

[45] ISO. *Information technology — Cloud computing — Service level agreement (SLA) framework — Part 2: Metric model*. Standard. International Organization for Standardization, 2018.

[46] Pooyan Jamshidi, Claus Pahl et Nabor C. Mendonça. « Managing Uncertainty in Autonomic Cloud Elasticity Controllers ». In : *IEEE Cloud Computing* 3.3 (2016), p. 50-60. doi : 10.1109/MCC.2016.66.

[47] Bernard J Jansen et al. « Data Preprocessing ». In : *Understanding Audiences, Customers, and Users via Analytics: An Introduction to the Employment of Web, Social, and Other Types of Digital People Data*. Springer, 2023, p. 65-75.

[48] ASHISH P Joshi et BIRAJ V Patel. « Data preprocessing: the techniques for preparing clean and quality data for data analytics process ». In : *Orient. J. Comput. Sci. Technol* 13.0203 (2021), p. 78-81.

[49] Foued Jrad et al. « SLA enactment for large-scale healthcare workflows on multi-cloud ». In : *Future Generation Computer Systems* 43 (2015), p. 135-148.

[50] Keven T Kearney, Francesco Torelli et Constantinos Kotsokalis. « SLA*: An abstract syntax for Service Level Agreements ». In : *2010 11th IEEE/ACM International Conference on Grid Computing* (2010), p. 217-224.

[51] Alexander Keller et Heiko Ludwig. « The WSLA framework: Specifying and monitoring service level agreements for web services ». In : *Journal of Network and Systems Management* 11 (2003).

[52] Yousri KOUKI, Thomas LEDOUX et al. « CSLA: A Language for Improving Cloud SLA Management. » In : *CLOSER* (2012), p. 586-591.

[53] Yousri KOUKI et Thomas LEDOUX. « RightCapacity: SLA-driven Cross-Layer Cloud Elasticity Management. » In : *International Journal of Next-Generation Computing* 4.3 (2013).

[54] Kyriakos KRITIKOS et Dimitris PLEXOUSAKIS. « Multi-cloud application design through cloud service composition ». In : *2015 IEEE 8th international conference on cloud computing*. IEEE. 2015, p. 686-693.

[55] Kyriakos KRITIKOS et al. « A Survey on Service Quality Description ». In : *ACM Comput. Surv.* 46.1 (juill. 2013). ISSN : 0360-0300. DOI : 10.1145/2522968.2522969. URL : https://doi.org/10.1145/2522968.2522969.

[56] Kyriakos KRITIKOS et al. « Multi-cloud provisioning of business processes ». In : *Journal of Cloud Computing* 8.1 (2019), p. 1-29.

[57] Kyriakos KRITIKOS et al. « Towards Multi-cloud SLO Evaluation. » In : *CLOSER*. 2018, p. 409-417.

[58] Fatma LAHMAR et Haithem MEZNI. « Multicloud service composition: a survey of current approaches and issues ». In : *Journal of Software: Evolution and Process* 30.10 (2018), e1947.

[59] Jean-Baptiste LAMY. « Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies ». In : *Artificial intelligence in medicine* 80 (2017), p. 11-28.

[60] Giorgio LEONARDI et al. « Towards semantic process mining through knowledge-based trace abstraction ». In : *International Symposium on Data-Driven Process Discovery and Analysis*. Springer. 2017, p. 45-64.

[61] Hui LI, Wolfgang THEILMANN et Jens HAPPE. *Sla translation in multi-layered service oriented architectures: Status and challenges*. Universität Karlsruhe, Fakultät für Informatik Interner Bericht, 2009.

[62] Heiko LUDWIG et al. « rSLA: Monitoring SLAs in dynamic service environments ». In : *International Conference on Service-Oriented Computing* (2015), p. 139-153.

[63] Adil MAAROUF, Abderrahim MARZOUK et Abdelkrim HAQIQ. « A review of SLA specification languages in the cloud computing. » In : *SITA* (2015), p. 1-6.

[64] Christian MAGER. « Analysis of Service Level Agreements using Process Mining techniques ». In : *FHWS Science Journal* (2014).

[65] Felix MANNHARDT et al. « Balanced multi-perspective checking of process conformance ». In : *Computing* 98.4 (2016), p. 407-437.

[66] Felix MANNHARDT et al. « From low-level events to activities-a pattern-based approach ». In : *International conference on business process management* (2016), p. 125-141.

[67] Christos-Minas Mathas et Costas Vassilakis. « Reconnaissance ». In : *Cyber-Security Threats, Actors, and Dynamic Mitigation*. CRC Press, 2021, p. 27-80.

[68] Peter M. Mell et Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Rapp. tech. Gaithersburg, MD, USA : NIST, 2011.

[69] Elena Molino-Peña et José María García. « Towards a Systematic Comparison Framework for Cloud Services Customer Agreements ». In : *International Conference on Service-Oriented Computing*. Springer. 2023, p. 241-252.

[70] Carlos Müller, Antonio Ruiz-Cortés et Manuel Resinas. « An initial approach to explaining sla inconsistencies ». In : *Service-Oriented Computing–ICSOC 2008: 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings 6*. Springer. 2008, p. 394-406.

[71] Carlos Müller et al. « Automated validation of compensable SLAs ». In : *IEEE Transactions on Services Computing* 14.5 (2018), p. 1306-1319.

[72] Carlos Müller et al. « Comprehensive explanation of SLA violations at runtime ». In : *IEEE Transactions on Services Computing* 7.2 (2013), p. 168-183.

[73] Mark A. Musen. « The Protégé Project: A Look Back and a Look Forward ». In : *AI Matters* 1.4 (juin 2015), p. 4-12.

[74] Athanasios Naskos, Anastasios Gounaris et Spyros Sioutas. « Cloud elasticity: a survey ». In : *Algorithmic Aspects of Cloud Computing: First International Workshop, ALGOCLOUD 2015, Patras, Greece, September 14-15, 2015. Revised Selected Papers*. Springer. 2016, p. 151-167.

[75] Stefan Nastic et al. « Sloc: Service level objectives for next generation cloud computing ». In : *IEEE Internet Computing* 24.3 (2020), p. 39-50.

[76] Pankesh Patel, Ajith H Ranabahu et Amit P Sheth. « Service level agreement in cloud computing ». In : (2009).

[77] Dana Petcu. « Multi-cloud: expectations and current approaches ». In : *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. 2013, p. 1-6.

[78] Thomas Pusztai et al. « Slo script: A novel language for implementing complex cloud-native elasticity-driven slos ». In : *2021 IEEE International Conference on Web Services (ICWS)*. IEEE. 2021, p. 21-31.

[79] Víctor Rampérez et al. « From SLA to vendor-neutral metrics: An intelligent knowledge-based approach for multi-cloud SLA-based broker ». In : *International Journal of Intelligent Systems* 37.12 (2022), p. 10533-10575.

[80] Alessandro Rossini et al. « The cloud application modelling and execution language (CAMEL) ». In : (2017).

[81] Mohammad Sadoghi et Hans-Arno Jacobsen. « Be-tree: an index structure to efficiently match boolean expressions over high-dimensional discrete space ». In : *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011, p. 637-648.

[82] Valdivino Santiago et al. « An environment for automated test case generation from statechart-based and finite state machine-based behavioral models ». In : *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*. IEEE. 2008, p. 63-72.

[83] *Service Level Agreement in the Data Center. (April 2002).* `http://www.sun.com/blueprints`. Accessed: 2010-03-28.

[84] Zhiming Shen et al. « Identifying resources for cloud garbage collection ». In : *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE. 2016, p. 248-252.

[85] Ron Sprenkels et Aiko Pras. *Service Level Agreements : Internet NG Deliverable D2.7*. English. Netherlands : Centre for Telematics et Information Technology (CTIT), 2001.

[86] *State of the cloud report from Flexera.* `https://www.flexera.com/about-us/press-center/flexera-2024-state-of-the-cloud-managing-spending-top-challenge`. Accessed: 2024-06-23.

[87] Ahmed Taha, Salman Manzoor et Neeraj Suri. « SLA-based service selection for multi-cloud environments ». In : *2017 IEEE International Conference on Edge Computing (EDGE)* (2017), p. 65-72.

[88] Irfan Ul Haq et Erich Schikuta. « Aggregation patterns of service level agreements ». In : *Proceedings of the 8th International Conference on Frontiers of Information Technology*. 2010.

[89] Rafael Brundo Uriarte, Rocco De Nicola et Kyriakos Kritikos. « Towards distributed sla management with smart contracts and blockchain ». In : *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2018, p. 266-271.

[90] Rafael Brundo Uriarte, Francesco Tiezzi et Rocco De Nicola. « Slac: A formal service-level-agreement language for cloud computing ». In : *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing* (2014), p. 419-426.

[91] Rafael Brundo Uriarte et al. « Distributed service-level agreement management with smart contracts and blockchain ». In : *Concurrency and Computation: Practice and Experience* 33.14 (2021), e5800.

[92] Wil Van Der Aalst. « Process mining: Overview and opportunities ». In : *ACM Transactions on Management Information Systems (TMIS)* 3.2 (2012), p. 1-17.

[93] Sergio Vavassori, Javier Soriano et Rafael Fernández. « Enabling large-scale IoT-based services through elastic publish/subscribe ». In : *Sensors* 17.9 (2017), p. 2148.

[94] Shuihua Wang et al. « Advances in data preprocessing for biomedical data fusion: An overview of the methods, challenges, and prospects ». In : *Information Fusion* 76 (2021), p. 376-421.

[95] Amir Teshome Wonjiga et al. « Blockchain as a trusted component in cloud SLA verification ». In : *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 2019, p. 93-100.

[96] Sebastiaan J. van Zelst et al. « Event abstraction in process mining: literature review and taxonomy ». In : *Granular Computing* (2020). doi : 10.1007/s41066-020-00226-2.

[97] Huan Zhou et al. « A blockchain based witness model for trustworthy cloud service level agreement enforcement ». In : *IEEE INFOCOM 2019-IEEE conference on computer Communications*. IEEE. 2019, p. 1567-1575.

Annexe A

Annexes

A.1 Code source de validation de la cohérence

FIGURE A.1 – Code source Python de la vérification de cohérence entre global-SLOs et sub-SLOs

```python
print("Algo 1 : Evaluation Global-SLOs <--> Sub-SLOs\nEtape 1 : Validation `a partir de global-SLOs ")
thresholds, problematic_sub_slas, nbc, sub_slos, agregats = {}, [], len(sub_slas), {}, {}
for obj_name, obj_data in global_sla_objectives.items():
    thresholds[obj_name] = round(eval((kb_obj[2].replace("SLO.v", str(obj_data['value'])).replace("nbc", str(nbc))) for kb_obj in KB_Agreg if kb_obj[0] == obj_name).pop(), 3)
print(f"Seuils calculés : {thresholds}\n")

for sub_sla in sub_slas:
    for obj_name, obj_data in sub_sla['objectives'].items():
        if obj_name in global_sla_objectives:
            threshold_value = eval((kb_obj[2].replace("SLO.v", str(global_sla_objectives[obj_name]['value'])).replace("nbc", str(nbc)) for kb_obj in KB_Agreg if kb_obj[0] == obj_name).pop())
            if (obj_data['condition'] == "<=" and obj_data['value'] > threshold_value) or (obj_data['condition'] == ">=" and obj_data['value'] < threshold_value):
                problematic_sub_slas.append([sub_sla['name'], obj_name])
    sub_slos.update({obj_name: (sub_slos.get(obj_name, []) + [obj_data]) for obj_name, obj_data in sub_sla['objectives'].items()})

for slo, slos in sub_slos.items():
    agreg = next((eval(f"{('max' if kb_obj[1]=='MaxType' else 'sum')}(d['value'] for d in slos)") for kb_obj in KB_Agreg if kb_obj[0] == slo), None)
    if kb_obj[1] == "AvailSumType": agreg = sum(round((d['value'] / agreg) * 100, 2) for d in slos)
    agregats[slo] = agreg
print(f"Agregats Calculé : {agregats}")

problematic_global_slas = [obj_name for obj_name, obj_data in global_sla_objectives.items() if (obj_data['condition'] == "<=" and obj_data['value'] < agregats.get(obj_name, 0)) or (obj_data['condition'] == ">=" and obj_data['value'] > agregats.get(obj_name, 0))]

print(f"\nSub-SLAs incohérents :{['\n'.join(f'{obj[0]} --> {obj[1]}' for obj in problematic_sub_slas)]}" if problematic_sub_slas else "Aucun sub-SLAs problématique detecté")
print(f"\nGlobal-SLAs incohérents :{['\n'.join(problematic_global_slas)]}" if problematic_global_slas else "Aucun global-SLAs problématique detecté")
```

FIGURE A.2 – Code source Python de la vérification de cohérence entre global-SLOs et stratégies de reconfiguration

```python
print("Algo 2 : Vérification Sub-SLOs <--> Stratégie de reconfiguration")
problematic_transitions = []
def check_in_kb_trans(sub_slo_obj_name): return any(kb_entry[0] == sub_slo_obj_name for kb_entry in KB_Trans)
def get_kb_trans_entry_for_metric(metric): return next((kb_entry for kb_entry in KB_Trans if kb_entry[0][0] == metric), None)
for sub_sla in (sub_sla for sub_sla in sub_slas if 'StateMachine' in sub_sla):
    resource_related_transitions = filter(lambda t: any(e['type'] == "ResourceRelatedEvent" for e in t['events']), sub_sla['StateMachine']['Transitions'])
    for transition in resource_related_transitions:
        kb_trans_entries = [get_kb_trans_entry_for_metric(obj) for obj in sub_sla['objectives'] if get_kb_trans_entry_for_metric(obj)]
        if not kb_trans_entries: continue
        for i in kb_trans_entries:
            if transition['actions'][0]['type'] == 'Scale-in': continue
            if event['predicate']['operator'] == "<" and float(event['predicate']['refValue'].rstrip('%')) < float(i[3][1].rstrip('%')) or event['predicate']['operator'] == ">" and float(event['predicate']['refValue'].rstrip('%')) > float(i[3][1].rstrip('%')): problematic_transitions.append([sub_sla['name'], transition['id']])

print("\nTransitions incohérentes : " + "\n".join(f"{obj[0]} --> {obj[1]}" for obj in problematic_transitions) if problematic_transitions else "Aucun problème detecté")
```

Titre : Gérer et assurer la qualité de services de ressources dans un environnement multi-cloud

Mots clés : Multi-cloud, Accord de niveau de service, Vérification de la conformité, Journal d'événements

Résumé : A mesure que le cloud computing se développe, de nouveaux besoins clients naissent, surpassant les capacités d'un unique fournisseur. Naturellement, les clients se sont intéressés à la consommation de services de plusieurs fournisseurs. Ce paradigme apportent un certains nombre d'avantages. Cependant, le multi-cloud soulève des verrous inhérents à la multiplicité de fournisseurs et à l'hétérogénéité de leurs services. En outre, la gestion de la qualité de service, le maintien et le suivi des objectifs de niveau de service, est rendue d'autant plus complexe en raison de la nature distribuée du contexte multi-cloud et de la dépendance qui existe entre les différents composants.
Afin de surmonter ces difficultés, nous visons dans cette thèse à (1) proposer un modèle de description de SLA multi-cloud dynamique, (2) proposer un processus de validation de la cohérence préalable à la mise en œuvre des SLAs multi-cloud dynamique et (3) proposer un processus de vérification post-mise en œuvre des SLAs multi-cloud dynamique. Pour le premier objectif, nous proposons un modèle de description de SLA multi-cloud dynamique composé d'un *global-SLA* permettant de représenter des objectifs globaux au système multi-cloud, de *sub-SLAs* permettant de représenter des objectifs locaux au niveau des composants du système multi-cloud et une machine à état permettant la formalisation de reconfiguration des services cloud pour adresser l'aspect dynamique. Pour le second objectif, nous proposons une vérification de la cohérence préalable à la mise en oeuvre des SLA multi-cloud dynamiques, identifiant et rapportant les SLOs incohérents. Nous proposons une vérification en 2 étapes: (1) entre les global-SLAs et les sub-SLAs en se basant sur une méthode d'agrégation de SLOs et (2) entre les sub-SLAs et stratégies de reconfiguration en se basant sur une technique de traduction de SLA. Pour le troisième objectif, portant sur le reporting du SLA multi-cloud dynamique après leur mise en œuvre par les fournisseurs de service cloud. Cette contribution, basée sur des techniques de fouille de processus comprenant la collecte et le pré-traitement des journaux d'événements produits durant la mise en œuvre, afin de les comparer avec le SLA multi-cloud dynamique établi et de signaler les éventuelles violations de SLA aux architectes cloud. Nous évaluons cette étape avec des journaux d'événements collectés auprès des 3 plus grands fournisseurs de services cloud: AWS, GCP et Azure.

Title : Managing and Ensuring the Quality of Resource Services in a Multi-Cloud Environment

Keywords : Multi-cloud, Service level Agreement, conformance checking, event-log

Abstract : As cloud computing continues to grow, new client needs arise, surpassing the capabilities of a single provider. Naturally, clients have become interested in consuming services from multiple providers. This paradigm brings a number of advantages. However, multi-cloud raises challenges inherent to the multiplicity of providers and the heterogeneity of their services. Moreover, managing the quality of service, maintaining and monitoring service level objectives, is made more complex due to the distributed nature of the multi-cloud context and the dependencies between different components.
To overcome these difficulties, in this thesis, we aim to (1) propose a model for dynamic multi-cloud SLA description, (2) propose a process for pre-implementation consistency validation of dynamic multi-cloud SLAs, and (3) propose a process for post-implementation verification of dynamic multi-cloud SLAs. To achieve the first objective, we propose a model for dynamic multi-cloud SLA description composed of a *global-SLA* to represent global objectives for the multi-cloud system, *sub-SLAs* to represent local objectives at the component level of the multi-cloud system, and a state machine to formalize the reconfiguration of cloud services to address the dynamic aspect. To achieve the second objective, we propose a pre-implementation consistency verification of dynamic multi-cloud SLAs, identifying and reporting inconsistent SLOs. We propose a two-step verification: (1) between the global-SLAs and the sub-SLAs based on an SLO aggregation method, and (2) between the sub-SLAs and reconfiguration strategies based on an SLA translation technique. To achieve the third objective, focused on reporting the dynamic multi-cloud SLA after implementation by cloud service providers, this contribution is based on process mining techniques, including the collection and pre-processing of event logs produced during the implementation, to compare them with the established dynamic multi-cloud SLA and report any SLA violations to cloud architects. We evaluate this step with event logs collected from the 3 largest cloud service providers: AWS, GCP, and Azure.

Institut Polytechnique de Paris
91120 Palaiseau, France