La composition de services dans le monde asynchrone
Formalisation et vérification en TLA+

Florent Chevrou Aurélie Hurault Philippe Mauran Meriem Ouederni
Philippe Quéinnec Xavier Thirioux

IRIT – Université de Toulouse

Résumé

Les architectures orientées services (SOA) permettent de répondre à deux défis importants du génie logiciel : la réutilisabilité et la décomposition. Néanmoins elles amènent de nouveaux problèmes, notamment liés à la répartition des services et la non-centralisation du contrôle. Les services étant indépendants et autonomes, il faut s'assurer que mis ensemble ils sont capables de communiquer et que leurs interactions n'introduisent pas de mauvais fonctionnement global. Dans le monde asynchrone, plus proche de la réalité, cette vérification devient non triviale, et cela d'autant plus qu'il existe de multiples modèles asynchrones, plus ou moins libéraux dans ce qu'ils autorisent. Nous exposons dans ce papier nos travaux en cours autour des modèles asynchrones et de la vérification des compositions de services paramétrées par ces modèles.

1 La composition de service...

Les architectures orientées services sont intéressantes de deux points de vue. Elles ont l'avantage de découper un problème en sous-problèmes plus simples et elles permettent de réutiliser les services. Ces architectures étant distribuées et si possible non contrôlées de façon centralisée, une des problématiques est la réalisation de l'assemblage [MP09] pour s'assurer que les services communiquent correctement ensemble et ont le comportement souhaité.

1.1 Obtention des compositions de services

Les systèmes distribués constitués d'un ensemble de processus sont généralement construits de manière ad-hoc. Mais la composition peut aussi découler d'une spécification. Dans ce cas, deux approches prédominent : l'orchestration de services et la chorégraphie de services [BGG+06]. L'orchestration conduit à la génération d'un contrôleur centralisé en charge de contrôler les échanges de messages entre les services. La distribution de ce contrôleur reste un problème majeur. Une chorégraphie spécifie, d'un point de vue global, les interactions (par messages) entre les participants d'une collaboration. Une chorégraphie ne peut pas être exécutée, elle est jouée quand les participants exécutent leur rôle. On dit qu'une chorégraphie est réalisable si, quand chaque participant agit indépendamment en fonction de son comportement, le comportement global joue la chorégraphie.

1.2 Vérification des compositions

De nombreux travaux s'intéressent à la vérification de compositions de services. Différents formalismes sont utilisés pour représenter les services : machine à états finis [DOS12, CLB08,

BCT04], algèbre de processus [BCPV04], réseaux de Petri [LFS+11, Mar03]. Différents critères sont utilisés pour représenter la compatibilité : absence d'interblocage [DOS12], réception non spécifiée [BZ83, DOS12], état terminal accessible [DOS12, BCT04, LFS+11], état terminal nécessairement atteint [BCT04, BCPV04], absence de famine [FUMK04], divergence [BCPV04]. Les modèles de communications utilisés sont principalement synchrones [DOS12, BCT04, FUMK04, BCPV04], plus rarement asynchrones [PS12].

Dans le cas asynchrone, chaque travail possède son propre modèle et propose une approche specifique pour ce modèle, en ignorant la diversité des modèles asynchrones. Par exemple, [PS12] propose une approche pour vérifier la réalisabilité d'une chorégraphie BPMN 2.0 dans le monde synchrone ou asynchrone et [BBO12] donne une condition nécessaire et suffisante pour prouver la réalisabilité d'une chorégraphie en asynchrone. Mais tous deux sont restreints à une forme particulière de communication asynchrone (FIFO n-1 dans la suite).

1.3 Nos choix

Nous nous intéressons à la diversité des modèles asynchrones. Il est important de noter que dans le monde asynchrone, c'est le médium qui décide du message délivré : les applications spécifient les canaux qu'elles écoutent, mais elles ne peuvent pas imposer l'ordre de délivrance des messages. En ce sens, c'est le modèle de communication qui *pousse* les messages vers les services, en respectant certaines propriétés d'ordre. Par ailleurs, nous utilisons différents critères de compatibilité : la terminaison (tous les processus terminent dans un état d'acceptation), l'absence de service définitivement bloqué en attente de message, l'absence de message inattendu (le modèle de communication délivre un message à un service qui ne sait pas le traiter à ce point). Enfin, nous avons choisi de décrire les services par des systèmes de transitions qui découlent d'une description CCS (sans la règle de communication synchrone, voir section 3.1), mais les résultats sur la comparaison des modèles de communication ne sont pas restreints à ces descriptions.

2 . . . dans le monde asynchrone

Le comportement et la correction d'ensemble d'une composition de services reposent sur l'exécution et l'ordonnancement des interactions entre composants ou services. La mise en œuvre des interactions a donc un impact direct sur la viabilité de l'application globale. Or, dans un contexte réparti, il existe des écarts considérables entre les différents protocoles d'interaction, en termes d'efficacité et de faisabilité. Il est donc essentiel de situer les différents modèles d'interactions les uns par rapport aux autres au regard de leur facilité et économie de mise en œuvre. De plus il faut que le contrat définissant le service d'interaction lui-même soit complètement explicité, afin de permettre d'évaluer et vérifier la composition de services qu'il emploie.

2.1 Les modèles

Une application répartie est vue comme un ensemble de services (ou sites) séparés, communiquant par échange de messages au travers d'un réseau asynchrone. Chaque site dispose d'une mémoire locale, privée. La communication est *asynchrone*, c'est-à-dire qu'il n'existe pas de borne maximale sur le temps d'acheminement d'un message par le réseau. Cet asynchronisme, caractéristique des environnements répartis, induit de nombreux résultats d'impossibilité [FLP85, BZ83]. Par exemple, il est impossible de détecter l'arrêt d'un site distant, puisqu'il est impossible de savoir si un site muet est effectivement en panne, ou a émis des

messages non encore parvenus. Enfin, le système est ouvert et dynamique : à tout moment de l'exécution, des sites peuvent rejoindre ou quitter l'ensemble des sites connectés.

Les différents modèles asynchrones pratiquement utilisés et que nous avons étudiés sont :

FIFO n-n Les messages sont ordonnés globalement et sont consommés dans leur ordre d'émission. Ce modèle repose sur un objet centralisé/partagé (par exemple une unique FIFO). Il reste éloigné d'une implantation répartie efficace et ne constitue qu'une première étape pour découpler les événements d'émission et de réception et ainsi s'éloigner du modèle synchrone.

FIFO n-1 Pour un service donné, les messages sont consommés dans leur ordre global d'émission. Ce modèle est souvent confondu avec FIFO 1-1 alors qu'il induit un ordre plus fort : alors même que les messages émis peuvent être totalement indépendants, l'ordre de leur délivrance est leur ordre d'émission *dans le temps absolu*. Ainsi, un événement d'émission est ordonné implicitement par rapport à l'ensemble des émissions vers le même destinataire.

Causal Les messages sont consommés dans un ordre respectant la causalité de leurs émissions [Lam78] : si un message m_1 est émis causalement avant un message m_2 (c-à-d. qu'il existe un chemin causal d'une émission à l'autre), alors un même site ne peut délivrer m_2 avant m_1. La réalisation de ce modèle nécessite la capture de la relation de causalité, par des histoires causales ou des vecteurs/matrices d'horloges.

FIFO 1-1 Les messages d'un même émetteur pour un même récepteur sont consommés dans leur ordre d'émission. Il peut être implanté grâce à une file associée à chaque couple de sites.

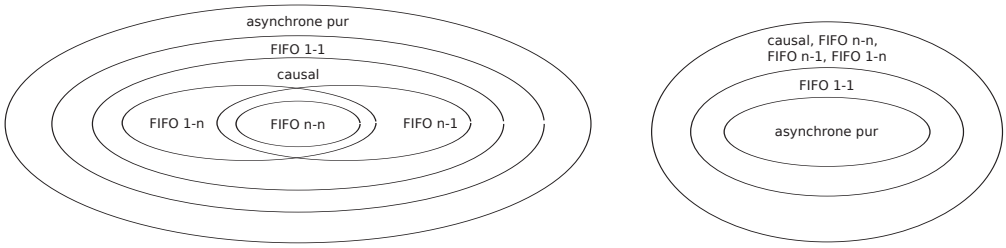
Asynchrone (pur) Il n'y a aucune contrainte sur l'ordre de consommation. Le réseau peut être vu comme un multi-ensemble où les sites déposent et d'où le réseau extrait arbitrairement des messages pour les délivrer.

FIFO 1-n les messages d'un même émetteur sont consommés dans leur ordre d'émission. Ce modèle est le pendant du FIFO n-1, avec, par exemple, sur chaque site, une file des messages émis.

2.2 Liens entre les modèles

Les modèles imposent plus ou moins de contraintes sur l'ordre de délivrance des messages. Si nous nous intéressons aux exécutions (suites d'émissions et consommations de messages), nous disons que pour un modèle de communication M et une exécution σ, $M \models \sigma$, si σ respecte l'ordre de consommation du modèle M. Pour deux modèles de communications M_1 et M_2, $M_1 \subseteq M_2 \triangleq \forall \sigma : M_1 \models \sigma \Rightarrow M_2 \models \sigma$. La figure 1(a) résume les liens entre les modèles. Certains résultats sont connus depuis longtemps (lien asynchrone, FIFO 1-1, causal, FIFO n-n), les preuves pour les modèles FIFO 1-n et FIFO n-1 ont été réalisées.

Nous nous intéressons aussi au comportement du système dans son ensemble. Pour deux modèles de communication M_1 et M_2, nous notons $M_1 \sqsubseteq M_2$ si tout système qui ne conduit pas à un état d'erreur (délivrance d'un message que le système ne sait pas traiter) dans M_1 ne conduit pas à un état d'erreur dans M_2. La figure 1(b) montre la hiérarchie des modèles dans le cas de système où tout message finit par être consommé. On remarque que les modèles causal, FIFO n-n, FIFO n-1 et FIFO n-1 se confondent. En effet si une exécution causale mène à un état d'erreur, il est également possible d'en trouver une similaire (au réordonnancement des émissions près), conforme au modèle FIFO n-n, qui mène également à l'état d'erreur. Les preuves sont en cours de réalisation.



(a) $\subseteq$ (b) $\sqsubseteq$

FIGURE 1 – Inclusion des modèles de communication

3 Formalisation et implantation

Nous avons défini formellement les modèles de communication et développé un outil de vérification de compatibilité. L'outil permet de vérifier automatiquement des compositions pour différents modèles de communication. Cet outil peut également servir à la vérification de chorégraphie selon la démarche de [PS12], qui lui ne gère qu'un seul modèle de communication.

3.1 Formalisation

Soit $\mathcal{C}$ un ensemble énumérable de **canaux**. Un canal n'est pas restreint à un unique émetteur ni à un unique récepteur. Les canaux peuvent être partitionnés en groupes auxquels sont associés différents modèles de communication.

Un **service** est défini par un système de transition étiqueté dont les étiquettes sont :
— $c!$: envoi d'un message sur le canal c ($c \in \mathcal{C}$) ;
— $c?$: réception de message sur le canal c ($c \in \mathcal{C}$) ;
— τ : action interne.

Un **modèle de communication** est élémentaire, comme fifo 1-1 pour un ensemble des canaux, ou composite, associant des modèles élémentaires différents à des groupes de canaux. Il est défini par un système de transition étiqueté dont les étiquettes sont :
— un sous-ensemble de $\mathbb{N} \times \bigcup_{c \in \mathcal{C}} \{c!\}$: envoi par le service i d'un message sur le canal c ;
— un sous-ensemble de $\mathbb{N} \times \bigcup_{c \in \mathcal{C}} \{c?\} \times \mathcal{P}(\mathcal{C})$: réception par le service i d'un message sur le canal c alors que le service est en écoute sur un ensemble de canaux ;
— τ : action interne.

Un **système** est composé d'un ensemble de services et d'un modèle de communication. Il s'agit d'un produit synchronisé entre les services et le modèle de communication, où une transition est la synchronisation entre une émission (resp. réception) d'un service, et une action d'émission (resp. de réception) du modèle de communication. Enfin plusieurs **critères de compatibilité** ont été définis : la terminaison, l'absence d'interblocage, l'absence de message jamais consommé, la délivrance d'un message inattendu... Ces notions de système, services, critères de compatibilité et modèles de communication ont été formalisées en TLA+ [Lam03], dans le but d'en donner une description non ambiguë et de bénéficier des outils associés. Le choix s'est porté sur TLA+ car un modèle de spécification de haut niveau et expressif était nécessaire et que nous bénéficions des outils associés : le vérificateur de modèle TLC et l'assistant de preuve TLAPS.

3.2 Implantation

Un outil a été développé pour vérifier la composition de service, paramétrée par les modèles de communication et les critères de compatibilité considérés. Partant d'une description CCS

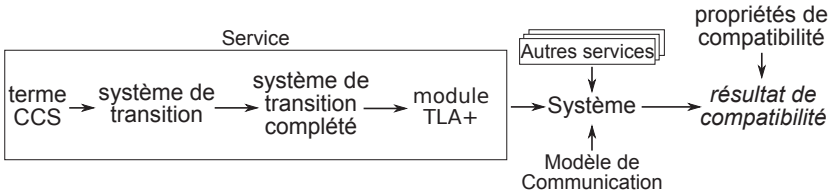


FIGURE 2 – Étapes principales de l'implantation

des services, l'outil génère le système de transitions correspondant à chaque service (sans la règle de communication synchrone de CCS), le traduit en une spécification TLA+, et construit le produit avec les modèles de communication étudiés (cf figure 2). L'outil appelle ensuite TLC, le vérificateur de modèle de TLA+, pour vérifier si les services sont compatibles pour un critère de compatibilité donné. Une étape importante de la conversion en TLA+ est la complétion du service : pour détecter la situation où un message inattendu est délivré au service, chaque état est complété pour assurer qu'il a une transition de réception de messages pour tous les canaux écoutés. Ces transitions ajoutées conduisent à un état d'erreur.

3.3 Exemple

Considérons un système faisant intervenir quatre processus (un étudiant, un responsable de formation, un secrétaire et un enseignant) et représentant les interactions lorsqu'un étudiant doit passer une seconde session d'un examen. Voici les termes CCS décrivant ces processus :

$$\begin{aligned}
\textbf{Responsable} &\triangleq nom! \cdot nom! \cdot session2! \cdot (ok? \cdot 0 + ko? \cdot annule! \cdot note! \cdot 0) \\
\textbf{Secretaire} &\triangleq nom? \cdot note? \cdot 0 \\
\textbf{Etudiant} &\triangleq session2? \cdot (\tau \cdot ko! \cdot 0 + \tau \cdot EtudiantOK) \\
EtudiantOK &\triangleq ok! \cdot examreq! \cdot documents? \cdot exam? \cdot reponses! \cdot 0 \\
\textbf{Enseignant} &\triangleq nom? \cdot (annule? \cdot 0 + examreq? \cdot EnseignantExam) \\
EnseignantExam &\triangleq documents! \cdot exam! \cdot reponses? \cdot note! \cdot 0
\end{aligned}$$

En analysant cette composition avec notre outil, nous obtenons les résultats suivants :

	fifo n-n	fifo n-1	fifo 1-n	causal	fifo 1-1	async.
Terminaison avec un réseau vide	✔	✔	✔	✔	×	×
Terminaison partielle (secrétaire)	✔	✔	✔	✔	×	×
Pas de message inattendu	✔	✔	✔	✔	×	×
Pas de blocage en attente de msg	✔	✔	✔	✔	×	×

La délivrance causale est nécessaire car il y a un lien de causalité entre *nom* (*Responsable* → *Enseignant*) et *examreq* (*Etudiant* → *Enseignant*) via *session2* (*Responsable* → *Etudiant*). Par ailleurs, la terminaison de la secrétaire n'est pas triviale, la note pouvant lui parvenir par deux chemins (via le responsable si annulation de l'étudiant ou via l'enseignant si l'étudiant décide de repasser l'examen).

4 Conclusion et perspectives

Ces travaux visent à éclairer la diversité du monde asynchrone et à fournir des outils pratiques adaptés à cette multiplicité. Ainsi nous pouvons vérifier automatiquement la correction de la composition de services, paramétrée par une combinaison de modèles de communication. L'utilisation du vérificateur de modèle TLC limite les vérifications aux systèmes ayant un nombre fini d'états. Des travaux sont en cours pour réaliser des preuves avec TLAPS (TLA Proof System) de façon à traiter des systèmes plus complexes, notamment des systèmes

paramétrés par le nombre de sites et qui présentent une certaine symétrie. Par ailleurs, les travaux en cours finalisent les relations entre les modèles et l'impact des opérateurs de communication sur ces relations, afin de clarifier la diversité du monde asynchrone et les spécificités de chaque modèle.

Références

[BBO12] Samik Basu, Tevfik Bultan, and Meriem Ouederni. Deciding choreography realizability. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 191–202, January 2012.

[BCPV04] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electron. Notes Theor. Comput. Sci.*, 105 :73–94, December 2004.

[BCT04] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Conceptual Modeling – ER 2004*, volume 3288 of *LNCS*, pages 524–541. Springer, 2004.

[BGG+06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *Coordination Models and Languages*, volume 4038 of *LNCS*, pages 63–81. 2006.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2) :323–342, April 1983.

[CLB08] Heung Seok Chae, Joon-Sang Lee, and Jung Ho Bae. An approach to checking behavioral compatibility between web services. *International Journal of Software Engineering and Knowledge Engineering*, 18(2) :223–241, 2008.

[DOS12] Francisco Durán, Meriem Ouederni, and Gwen Salaün. A generic framework for n-protocol compatibility checking. *Science of Computer Programming*, 77(7-8) :870–886, July 2012.

[FLP85] Michael Fischer, Nancy Lynch, and Michael Paterson. Impossibility of distributed consensus with one faulty process. *J. of the ACM*, 32(2) :374–382, April 1985.

[FUMK04] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Compatibility verification for web service choreography. In *IEEE International Conference on Web Services*, pages 738–, 2004.

[Lam78] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, July 1978.

[Lam03] Leslie Lamport. *Specifying Systems*. Addison Wesley, 2003.

[LFS+11] Xitong Li, Yushun Fan, Q. Z. Sheng, Z. Maamar, and Hongwei Zhu. A Petri net approach to analyzing behavioral compatibility and similarity of web services. *Trans. Sys. Man Cyber. Part A*, 41(3) :510–521, May 2011.

[Mar03] Axel Martens. On compatibility of web services. *Petri Net Newsletter*, pages 12–20, 2003.

[MP09] Annapaola Marconi and Marco Pistore. Synthesis and composition of web services. In *Formal Methods for Web Services*, volume 5569 of *LNCS*, pages 89–157. 2009.

[PS12] Pascal Poizat and Gwen Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *27th Symposium On Applied Computing (SAC 2012)*, pages 1927–1934, March 2012.