État de l'art sur l'interopérabilité des systèmes

Lionel Seinturier

État de l'art sur l'interopérabilité des systèmes

Projet Hermès
Lot 1 : Connecteurs d'alimentation du hub
Livrable 1.1 : État de l'art sur l'interopérabilité des systèmes
Auteur : Inria ADAM (L. Seinturier)
Date : 22/7/2013

1. Introduction

Ce document consiste en un rapport d'état de l'état de l'art scientifique sur les techniques de connecteurs pour assurer l'interopérabilité des systèmes communicants hétérogènes.

L'interopérabilité est la « capacité que possède un produit ou un système, dont les interfaces sont intégralement connues, à fonctionner avec d'autres produits ou systèmes existants ou futurs et ce sans restriction d'accès ou de mise en œuvre. » [Interopérabilité, Wikipédia].

Dans ce document, nous nous intéressons à la façon dont cette interopérabilité est assurée à l'aide de connecteurs. En première approche, le domaine de l'architecture logicielle peut être vu comme comportant deux éléments principaux : des composants logiciels et des connecteurs logiciels[1]. Les composants fournissent des fonctions de calcul et de stockage, tandis que les connecteurs assurent les interactions entre les différents composants du système. Par raccourci, les connecteurs sont parfois désignés sous le terme de composants de communication pour signifier que leur rôle premier est de mettre en œuvre les communications au sein du système. Au delà des services de communication pure, les connecteurs peuvent donc être vus comme les entités mettant en œuvre l'interopérabilité, au sens large du terme, entre les différents constituants d'un système. Plus précisément, la définition suivante du terme connecteur fait référence : « *Connectors mediate interactions among components ; that is, they establish the rules that govern component interaction and specify any auxiliary mechanisms required.* » [Shaw & Garlan, 1996].

Dans la suite de ce document, nous présentons une taxonomie de référence pour la notion de connecteur (section 2). La section 3 présente quelques langages et *frameworks* qui proposent de façon native la notion de connecteur. La section 4 s'intéresse aux cas où les connecteurs peuvent évoluer dynamiquement à l'exécution. Finalement, la section 5 propose quelques recommandations pour la gestion des connecteurs du projet Hermès.

[1] Dans la suite de ce document, sauf mention explicite du contraire, nous omettrons l'adjectif logiciel pour les termes architecture, connecteur, composant, système, service, qui seront donc envisagés comme lui étant implicitement associés.

2. Taxonomie de référence

[Mehta et al., 2000] ont proposé une classification pour les différents connecteurs couramment rencontrés dans le domaine des systèmes informatiques qui est illustrée Figure 1. Cette classification se base sur les différents types de connecteurs existants et à aussi pour but de pouvoir accueillir de nouveaux connecteurs créés en fonction des besoins des applications.

Cette classification distingue les catégories de communication, coordination, conversion et facilitation. La communication fait référence à l'échange de données, la coordination au transfert du contrôle entre composants comme lors d'un appel de fonctions ou d'un appel de méthode, la conversion à la transformation de données, et la facilitation à la médiation qui peut être effectuée par un connecteur comme lorsque par exemple des mécanismes d'équilibrage de charge, d'ordonnancement ou de contrôle de concurrence sont mis en œuvre. Notons, que comme illustré, un même type de connecteur peut appartenir à plusieurs catégories.

Dans un deuxième temps, cette classification met en avant différents types de connecteurs. On retrouve ainsi les types appel de procédure, événement, accès aux données, lien, flux, arbitre, adaptateur, distributeur. L'appel de procédure correspond au mécanisme bien connu d'invocation requête réponse dans les systèmes client/serveur, l'événement à l'artefact de base des systèmes asynchrones, l'accès aux données à ce qui se pratique couramment dans le cadre des interactions avec les SGBD, le lien à la capacité à maintenir un lien logique entre différents composants d'un système, le flux au mode de diffusion utilisé par exemple dans les systèmes de diffusion d'audio et de vidéo, l'arbitre à la résolution de conflits et à la redirection de flots de contrôle, l'adaptateur à la mise en cohérence et à la transformation d'informations hétérogènes, le distributeur au routage et à la coordination d'informations.

Au delà de la classification des différents connecteurs, cette taxonomie permet également de les comparer et de fournir un cadre de référence sur lequel se baser pour qualifier et positionner les connecteurs.

Remarques sur la taxonomie. Premièrement, dans le domaine des applications Internet, les connecteurs de communication et ceux d'accès aux données sont ceux qui sont le plus fréquemment rencontrés. Deuxièmement, les propriétés non fonctionnelles des connecteurs, comme par exemple les délais ou les échéances, ne sont pas vus comme des éléments de première classe de la taxonomie : elles restent des attributs internes des connecteurs mais ne rentrent pas dans la classification. Or il apparaît que dans le contexte du projet Hermès, ce type d'attribut est amené à jouer un rôle important dans les interactions entre les différents composants de l'architecture. Troisièmement, cette taxonomie ne s'intéresse pas à la façon dont les connecteurs sont générés ou sur la façon dont ils évoluent dans le temps, par exemple en réaction à des contraintes de temps de réponse imposées par l'environnement. En conséquence, il nous semble intéressant de compléter cette taxonomie par le point de vue présenté dans les Sections 4 et 5 dans lesquels nous nous abordons respectivement des langages et des *frameworks* qui permettent de définir statiquement des connecteurs et des approches qui permettent de les faire évoluer dynamiquement à l'exécution.

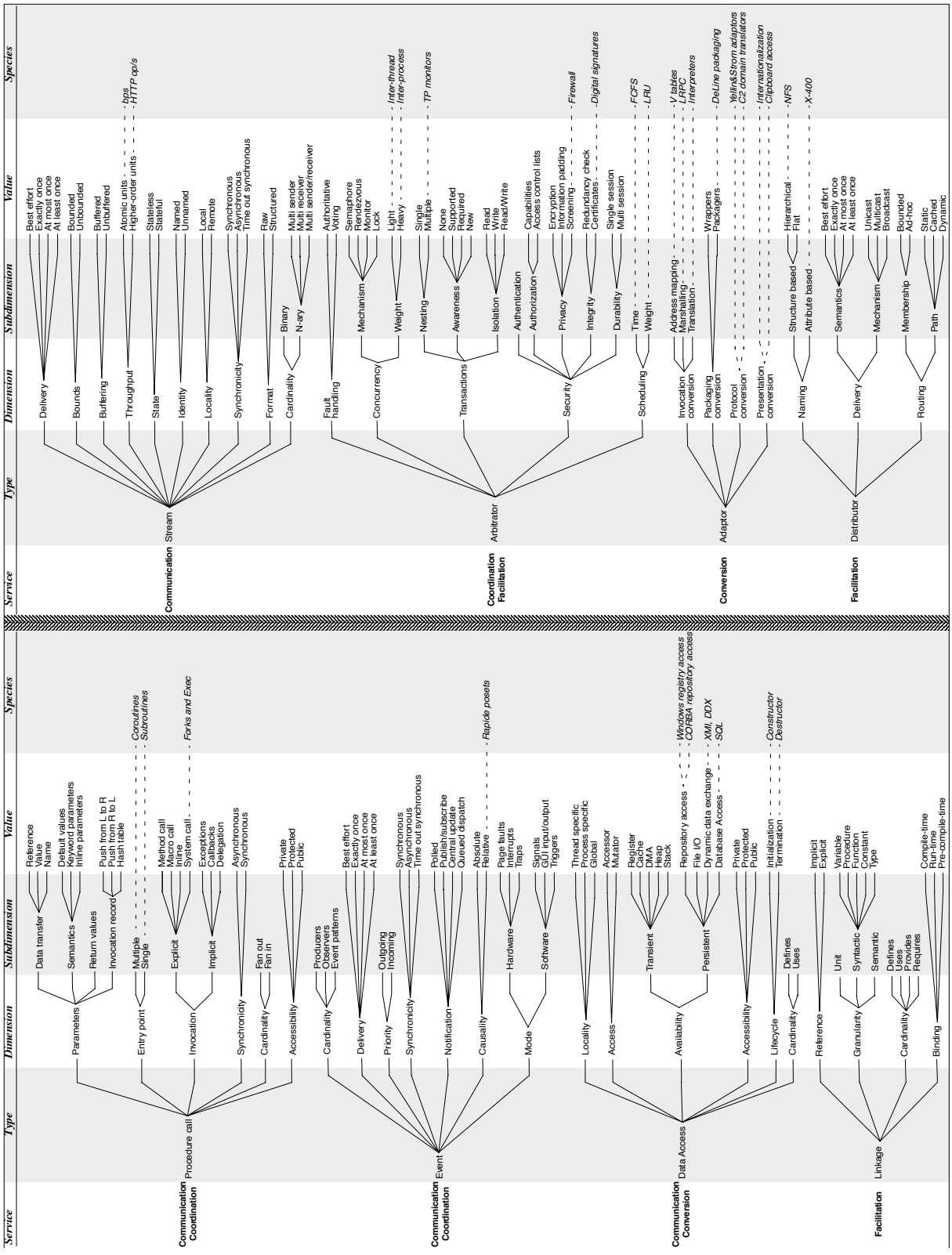

Figure 1. Taxonomie de la notion de connecteur d'après [Mehta et al., 2000]

3. Connecteurs statiquement définis

Plusieurs travaux de recherche se sont intéressés à la notion de connecteur et ont proposé des langages ou des *frameworks* de programmation intégrant cette notion. Une distinction importante peut être faite selon que le connecteur s'apparente à un élément immuable de l'architecture défini statiquement dans le programme, ou est une entité pouvant évoluer lors de l'exécution via des reconfigurations dynamiques (voir section suivante). Dans cette section, nous présentons trois approches qui mettent en œuvre la notion de connecteur défini statiquement.

ArchJava [Aldrich et al., 2002] est un travail pionnier pour la notion de connecteur. Il s'agit d'une extension du langage Java qui unifie les notions d'architecture et d'implémentation de façon à réduire la distance entre les deux et à garantir que l'implémentation reste conforme à l'architecture. ArchJava enrichit la notion de classe avec celle de connecteur (port dans la terminologie ArchJava) pour donner la notion de composant. Toute communication entre deux ou plusieurs composants doit passer par un connecteur, assurant ainsi à cette dernière notion un contrôle complet sur l'ensemble des interactions au sein d'une application. Les interfaces des ports ArchJava sont bidirectionnelles et comprennent des méthodes fournies par le composant et requises d'autres composants. Les communications via les ports sont 1-1 ou 1-n. Le langage ArchJava fournit un mot clé `connect` permettant de connecteur deux ports compatibles. L'architecture d'un programme ArchJava peut évoluer dynamiquement à l'exécution par connexions et déconnexions de composants. La structure des ports est néanmoins immuables : leur interface est fixe, les communications correspondent à de simples appels de méthode et ne permettent pas d'exprimer des comportements particuliers (traitement des données, gestion de la qualité de service, etc.). Au final, ArchJava est un langage pionnier, efficace et pragmatique permettant d'inclure facilement la notion de connecteur dans une application Java.

La notion de connecteur exogène (*exogenous connector*) [Lau et al., 2005] permet de séparer dans une architecture les entités qui ont un rôle purement fonctionnel de celles qui mettent en œuvre les communications, les connecteurs. Ceux-ci implémentent une interface générique de réification des invocations de méthodes et sont assemblés avec des entités fonctionnelles. Les communications sont initiées par les entités fonctionnelles et prises en charge par les connecteurs. Il est ainsi possible d'utiliser des protocoles réseaux, de mettre en œuvre des politiques de la gestion de la qualité de service, ou de programmer tout autre comportement nécessaire à la réalisation de la communication. Les connecteurs peuvent être assemblés afin d'obtenir par composition le comportement souhaité. Un prototype a été défini pour le langage Java. Au final, ce travail présente l'intérêt de permettre de programmer la communication entre entités fonctionnelles d'une application.

La plate-forme *middleware* FraSCAti [Seinturier et al., 2012] pour la mise en œuvre d'architectures réparties orientées services inclut le *framework* Binding Factory (BF) pour la création de liaisons entre composants de services. L'idée est de pouvoir séparer les invocations de services métier des technologies réseaux utilisées pour les mettre en œuvre. Les composants métiers définissent ainsi des services fournis et requis qui sont dans un second temps associé à une technologie réseau. Le *framework* BF se charge de créer les souches de communication correspondant au protocole choisi. Actuellement les protocoles suivants sont supportés : SOAP, REST, JSON-RPC, Java RMI, UPnP. Comme son nom le suggère, le *framework* BF est basé sur le *design pattern* Factory et peut être étendu pour supporter de nouveaux protocoles réseaux. Au delà il peut également être programmé pour définir tout type de liaison,

par exemple mettant en œuvre des propriétés de qualité de service. Finalement, bien que les connecteurs soient statiquement définis dans FraSCAti, ils offrent néanmoins un premier niveau de dynamicité via le paramétrage des adresses de communication. Ainsi les URL associées aux connecteurs sont des paramètres qui peuvent être manipulés et modifiés à l'exécution de façon à reconfigurer l'architecture.

4. Connecteurs adaptables dynamiquement

Récemment, un certain nombre de travaux de recherche ont émergé pour étendre la notion de connecteur telle que nous l'avons présenté dans la section précédente. L'idée part du constat que les interactions entre entités communicantes ne sont pas nécessairement figées et peuvent évoluer dans le temps. De ce fait, le connecteur doit pouvoir évoluer pour supporter les nouvelles formes prises par l'interaction. Dans cette section, nous présentons deux approches qui adressent cet objectif.

[Bencomo et al., 2013] propose la notion de connecteur émergent (*emergent connector*). L'idée consiste à pouvoir apprendre à l'exécution le comportement de l'interaction entre entités communicantes et générer à la volée l'infrastructure qui permet de mettre en œuvre cette interaction. Ainsi, un cas d'étude ciblé par l'approche consiste à pouvoir faire communiquer un sous-système utilisant le protocole SOAP avec un sous-système utilisant le protocole HTTP Live Stream, chaque sous-système ayant par ailleurs sa propre API de service. Il s'agit donc de pouvoir générer un connecteur qui assure à la fois la traduction entre les protocoles réseaux et les API de service. Pour ce faire, au niveau protocole, l'approche utilise un langage XML de description de langage (MDL *Message Description Language*) dans lequel les primitives de base permettant d'utiliser le protocole sont décrites. Le comportement des API de services est quant à lui décrit à l'aide d'automates états-transitions. En partant de la liste des méthodes de l'API, l'automate décrivant leurs enchaînements légaux est construit pas à pas par essais-erreurs (voir p177 de [Bencomo et al., 2013]). Chaque erreur détectée est considérée comme dénotant une séquence erronée dans l'utilisation de l'API et l'automate est modifié de façon à interdire cet enchaînement. Au final, l'intérêt de l'approche réside dans la possibilité de découvrir dynamiquement le comportement de l'API de service et de pouvoir générer un connecteur en fonction de ce comportement appris.

Plus les systèmes interagissant sont hétérogènes et dynamiques, plus il devient difficile de définir les mécanismes leur permettant d'interopérer. Une réflexion à haut niveau d'abstraction s'avère alors nécessaire. Pour cela, un certain nombre de solutions, comme par exemple [Blair et al., 2011], proposent la notion d'ontologie associée à des langages de description comme OWL. Une ontologie permet de décrire comment des concepts peuvent être groupés, mis en relation au travers d'une hiérarchie, et subdivisés en fonction de similarités et de différences [Ontology, Wikipedia]. Il s'agit alors de raisonner sur les concepts de haut niveau manipulés par les entités qui interagissent plutôt que sur les détails de leur mise en œuvre. En faisant cela on espère pouvoir traiter plus facilement des cas plus complexes et automatiser la gestion des détails d'interaction. Les ontologies des entités communicantes peuvent alors être comparées et rapprochées à l'aide de langages de règles comme SWRL pour déduire les traductions de concepts qu'il va être nécessaire de mettre en œuvre. Ainsi, [Blair et al., 2011] applique cela à la traduction de format de paquets réseaux entre les protocoles BBR et Broadcom du domaines des réseaux véhiculaires ad hoc (VANET). Au final, ce genre d'approches, très puissantes, s'applique aux situations dans lesquelles il y a très peu de connaissances communes initiales entre les entités communicantes.

5. Vers une approche auto-descriptive des connecteurs

En se basant sur l'état des pratiques en terme de recherche et sur les cibles applicatives du projet Hermès, une recommandation pour les aspects recherche de la gestion des connecteurs *middleware* peut être d'adopter une approche MDE (*Model Driven Engineering*). Le but est d'aller vers une auto-description des connecteurs et une génération, autant automatique que possible, de leur code afin de pouvoir capitaliser et rationnaliser les bonnes pratiques et les connaissances métiers liées à la production de connecteurs *middleware* dans les architectures visées par le projet Hermès.

L'adoption d'une approche MDE consistera à définir un méta-modèle de connecteur qui capture les propriétés des interactions que l'on veut mettre en place en termes de protocoles de communications, propriétés non fonctionnelles (par exemple, en fonction des besoins, temps de réponse, débit, échéance, voire taille des données, nombre de requêtes simultanées, nombre de connexions simultanées), algorithmes d'adaptation des communications (boucle de rétroaction qui permet d'adapter les communications en fonction de ce qui est observé). Ce méta-modèle pourra être utilisé a minima pour documenter les connecteurs présents dans l'architecture, ou de façon plus automatique pour générer leur code.

6. Références

[Aldrich et al., 2002] J. Aldrich, C. Chambers, D. Notkin, *ArchJava: Connecting Software Architecture to Implementation*, 24th International Conference on Software Engineering (ICSE'02), pages 187-197, May 2002.

[Bencomo et al., 2013] N. Bencomo, A. Bennaceur, P. Grace, G. Blair, V. Issarny, *The Role of models@run.time in Supporting On-The-Fly Interoperability*, Springer Journal of Computing, 95(3):167-190, March 2013.

[Blair et al., 2011] G. Blair, A. Bennaceur, N. Georgantas, P. Grace, V. Issarny, V. Nundloll, M. Paolucci, *The Role of Ontologies in Emergent Middleware: Supporting Interoperability in Complex Distributed Systems*, 12th ACM/IFIP/USENIX International Middleware Conference (Middleware'11), pages 410-430, December 2011.

[Interopérabilité, Wikipédia] http://fr.wikipedia.org/wiki/Interopérabilité

[Lau et al., 2005] K.-K. Lau, P. Elizondo, Z. Wang, *Exogenous Connectors for Software Components*, 8th ACM SIGSOFT International Symposium on Component-Based Software Engineering (CBSE'05), LNCS, pages 90-106, 2005.

[Mehta et al., 2000] N. Mehta, N. Medvidovic, S. Phadke, *Towards a Taxonomy of Software Connectors*, 22nd International Conference on Software Engineering (ICSE'00), pages 178-187, 2000.

[Ontology, Wikipedia] http://en.wikipedia.org/wiki/Ontology

[Seinturier et al., 2012] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, J.-B. Stefani, *A Component-Based Middleware Platform for Reconfigurable Service-Oriented Architectures*, Software Practice and Experience (SPE), 42(5):559-583, May 2012.

[Shaw & Garlan, 1996] M. Shaw, D. Garlan, *Software Architecture: Perspectives on a Emerging Discipline*, Prentice-Hall, 1996.