Le monitoring pour la construction de la représentation graphique de l'architecture des applications orientées services

Mariam CHAABANE
University of Sfax, ReDCAD, B.P. 209, 3021
Sfax, Tunisia
meriam.chaabane@redcad.org

Ismael BOUASSIDA RODRIGUEZ
CNRS, LAAS, 7 avenue du colonel Roche,
F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse,
France
University of Sfax, ReDCAD, B.P. 1173, 3038
Sfax, Tunisia
bouassida@laas.fr

Résumé

Ce travail se situe dans le cadre des applications logicielles conçues selon le modèle architectural orienté service. Ces applications sont construites par composition de plusieurs services web provenant de sources différentes et fonctionnant ensemble, indépendamment de leur emplacement afin de réaliser une fonctionnalité complexe. Ainsi, lors de l'interaction entre ces Services Web, plusieurs évènements surviennent, à savoir la dégradation de qualité de service, les pannes des services déployés, etc. Ceci engendre des perturbations de communication. Ce travail a permis le développement d'une application orientée services appelée Smart City, la mise en place des mécanismes de monitoring afin de scruter l'architecture de cette application, suivre ses interactions et construire la représentation graphique de son architecture en utilisant les graphes. Nous montrons enfin les résultats de nos expérimentations.

Mots Clés

Monitoring, Architecture orientée services, Représentation graphique d'architecture, Les Graphes

1. INTRODUCTION

Les applications logicielles conçues selon le modèle architectural orienté service sont construites par la composition de plusieurs services web provenant de sources différentes et fonctionnant ensemble, indépendamment de leur emplacement. Cependant, la valeur d'un Service Web semble insatisfaisante pour avoir des scénarios complexes. D'où la nécessité de composition de services afin d'apporter une valeur ajoutée. En effet, un Service Web composé assemble divers Services Web élémentaires et coordonne les interactions au cours de leurs exécutions en vue de réaliser des tâches plus complexes. La composition dans notre cas est décrite à partir d'un point de vue local. Dans ce cas là, nous parlons d'une orchestration. Dans l'orchestration, les Services Web invoqués sont sous le contrôle d'un seul processus central appelé orchestrateur de Services Web.

Lors de l'interaction entre ces Services Web, plusieurs évènements surviennent, à savoir la dégradation de qualité de service, les pannes des services déployés, etc. Ceci engendre des perturbations de communication. Pour remédier à ce problème, nous proposons une approche pour la mise en place des mécanismes de monitoring afin de scruter les architectures des applications orientées services et suivre les interactions entre leurs services. Notre défi consiste ainsi à suivre le flux de messages entre les Services Web de ces applications. Nous nous appuyons sur le principe d'interception de messages pour extraire les valeurs de métriques de qualité de service. Les résultats de monitoring permettant de percevoir les perturbations de communication seront enregistrées dans une base de données. Ils seront exploités ultérieurement afin de construire graphiquement les architectures des applications mentionnées précédemment et d'étudier l'évolution des métriques de qualité de service. Pour assurer la description de ces architectures, nous avons recours aux graphes qui représentent un moyen naturel, intuitif et avec un pouvoir expressif assez puissant pour spécifier les aspects statiques et dynamiques des architectures. Les nœuds du graphe représentent ainsi les services, et les arcs représentent les liens entre eux. Nous générons plusieurs types de graphes en utilisant le frame work Java Universel Network/Graph Framwork (JUNG). En effet, ceci facilite le monitoring de l'architecture en permettant d'observer les interactions selon plusieurs points de vue.

Pour mettre en œuvre notre approche, nous nous sommes appuyés sur un cas d'étude illustrant un réseau intelligent de distribution d'énergie, appelé Smart City. La mise en oeuvre de ce réseau en utilisant des dispositifs hétérogènes permet d'identifier de manière précise la consommation d'énergie des habitants d'un bâtiment, d'un quartier, d'une ville ou d'une région. Ce genre de réseau est censé encourager les consommateurs d'énergie à rationaliser leur consommation en électricité et permettre des économies globales dans une perspective de développement durable. Pour la réalisa-

tion du cas d'étude Smart City, nous avons implémenté une application orientées services tout en appliquant notre approche de monitoring et de construction de représentation graphique d'architecture. Ce travail définit la solution que nous envisageons afin de monitorer les applications orientées services et de construire la représentation graphique de leurs architectures en cours de l'exécution.

Le reste de l'article est organisé comme suit : La section 2 présente les travaux connexes. La section 3 définit notre approche qui repose sur deux phases : la création et le déploiement des moniteurs auprès de chaque entité de l'application orientée service puis la présentation de l'architecture en graphe en exploitant les mesures récupérées. Les sections 4 et 5 présentent notre cas d'étude appelé 'Smart City' qui vise la diminution de la consommation d'énergie dans un quartier ou une région ainsi que sa modélisation. Dans la section 6 nous avons appliquée notre approche sur le cas d'étude Smart City. La section 7 permet de détailler les étapes d'implémentation et d'effectuer des expérimentations à travers un ensemble de scénarios.

2. TRAVAUX CONNEXES

Notre travail se base sur le monitoring d'une part, la reconstruction d'une représentation d'architecture d'autre part. Ainsi, nous présentons dans cette section un ensemble de travaux connexes qui abordent ces deux thèmes.

2.1 Approches existantes de monitoring des architectures logicielles

Fournir une bonne qualité de service des applications distribuées offre un support puissant de diffusion des informations à large échelle. Toutefois, il est difficile de maintenir la qualité de services dans les environnements dynamiques. Plusieurs chercheurs ont traité ce problème, en proposant des techniques de monitoring afin de contrôler la qualité de service et détecter les éventuelles pannes affectant le système. Plusieurs approches de monitoring existent dans la littérature. Généralement et suite à une dégradation de qualité de services, le monitoring est suivi par des actions de réparation [18] [9]. Mais nous nous concentrons sur les travaux qui traitent surtout la partie monitoring. Il est à noter que le problème de monitoring est traité pour plusieurs types d'architectures, à savoir les architectures SOA [26], les middlewares publier/souscrire [3], et aussi le cloud [29].

Comme notre travail se base sur une application orientée services, nous nous intéressons plus aux travaux qui implémentent des solutions de monitoring sur les architectures SOA, et plus précisément qui se basent sur les services web simples [15] ou des services web composites [17]. Nous citons ainsi des approches de monitoring des applications orientées services orchestrées en utilisant le Business Process Execution Language for Web Services (BPEL). Nous citons aussi des approches de monitoring des applications implémentées avec le BPEL et qui utilisent la Programmation Orientée Aspect(AOP). Nous clôturons cette section par la présentation de l'approche de [13] qui est la plus proche de notre travail.

Psiuk et al. [26] présentent un framework de monitoring pour la couche d'intégration de systèmes SOA réalisé par un Enterprise Service Bus (ESB). Il introduit un métamodèle ESB générique (EMM) et définit des mécanismes qui collectent les données de monitoring relatives aux entités du modèle. Cependant, le mecanismes EMM n'est pas uniforme pour supporter le monitoring des Business Activity (BAM). En effet, les données de monitoring collectées par EMM ne peuvent être exploitées dans BAM qu'après filtrage et enrichissement par des données sémantiques relatives aux business process.

Le travail d'An et al. [3] présente une technologie de monitoring basée sur le paradigme publier/souscrire. Les auteurs proposent ainsi une solution légère et scalable appelée SQRT-C qui exploite le Service OMG de Distribution de Données (OMG DDS) afin d'être utilisable avec une multitude de plateformes Cloud.

Bratanis et al. [15] ne visent pas par leur travail la présentation d'une approche de monitoring particulier mais plutôt d'une architecture qui pourrait être utilisée pour intégrer les différentes approches de monitoring. Ainsi, ils donnent dans cet article une mise en œuvre d'une architecture extensible pour le monitoring des services Web conversationnels au cours de l'exécution. L'architecture a été conçue et implémentée pour faciliter l'intégration avec les architectures orientées services existantes, et pour permettre l'utilisation de différentes approches de monitoring. Cependant, les résultats expérimentaux ont montré une augmentation importante dans le temps d'exécution en fonction du nombre de consommateurs simultanés.

Erradi et al. [17] proposent une approche de monitoring des Services Web composites. Pour cela, ils présentent un projet appelé Manageable and Adaptive Service Compositions (MASC) effectuant le monitoring au cours de l'exécution en se basant sur le langage WS Policy4MASC qui étend WS-Policy [2]. En effet, cette solution permet de monitorer les messages SOAP et les processus d'orchestration d'une manière synchrone et asynchrone. L'avantage principal de MASC est l'utilisation d'une politique permettant la séparation entre le déroulement des opérations de base et le monitoring. Mais, cette approche n'est pas intégrée dans les normes existantes.

Le travail de Baresi et al. [10] présente un langage de spécification de monitoring générale appelé Service Centric Monitoring Language (SECMOL). Ce langage est créé comme une extension de WS-Agreement [5]. Il a été développé pour suivre les exigences fonctionnelles et les contraintes de qualité de service des processus BPEL et monitorer les interactions entre le processus et un service externe, en regardant les messages qu'ils échangent. SECMOL prend également en charge différents niveaux de granularité, offrant une approche flexible. Cette généralité signifie que l'approche peut facilement être portée et exploitée dans le cadre d'autres technologies basées sur des processus.

Le travail de Baresi et al. [11] au lieu de travailler sur la définition d'une nouvelle approche pour le monitoring BPEL, il a proposé une approche coopérative basée sur l'intégration de solutions différentes. Ainsi il décrit un Framework de monitoring obtenu par l'intégration de deux approches bien connues, à savoir Dynamo [7], [8] et Astro [6]. L'intégration proposée est à deux niveaux : au niveau des langages

utilisés pour exprimer les propriétés à monitorer et au niveau architectural (architecture de monitoring). Dynamo et Astro se complètent mutuellement et produisent une nouvelle solution cohérente pour monitorer les processus BPEL au cours de l'exécution. Le défi ici est d'éviter la redondance conceptuelle, maximiser la réutilisation du code et obtenir en même temps une solution efficace.

L'approche de Sun et al. [28] étend le moteur d'exécution BPEL [4] en vue de récupérer les données d'interactions entre les Services Web. Elle propose un cadre de monitoring de bout en bout basé sur AOP [22]. Tout d'abord, le modèle WS-Policy est utilisé pour exprimer les exigences en matière de monitoring de l'utilisateur. Comme il n'y a pas de méthode standard pour exprimer les exigences de monitoring pour les services, cette approche cherche à innover sur la forme d'expression standard. Ainsi, elle utilise Extended Charts Message Sequence (ECMS) et Message Event Transferring Graph (METG) pour exprimer les propriétés du service. ECMS et METG peuvent décrire non seulement la propriété d'origine du service, mais aussi certains aspects des exigences de surveillance.

Le travail de Barbon et al. [6] présente une nouvelle solution au problème de monitoring des services Web implémenté dans BPEL. Le travail se base sur une architecture qui prend en charge des 'moniteurs d'instance' qui traitent l'exécution d'une seule instance de processus BPEL, ainsi que des 'moniteurs de classe' qui déclarent des informations relatives à toutes les instances d'un processus BPEL. Ce travail définit aussi un langage pour la spécification de l'instance et de la classe. Celui-ci permet de spécifier les propriétés booléennes, statistiques, et les propriétés liées au temps. Enfin, ce travail donne une technique pour la traduction automatique de tous ces types de moniteurs pour les programmes Java.

Les travaux de Moser et al. [24] présentent une approche de monitoring des applications à base BPEL se servant de l'AOP. Ils proposent dans ce cadre le système VieDAME qui permet d'intercepter les messages SOAP. De plus, il offre une stratégie de substitution des Services Web au cours de l'exécution. Cependant, il ne permet pas de choisir la meilleure stratégie de substitution en se basant sur l'historique des choix précédents.

L'approche de Yoo et Lee [19] s'appuie aussi sur l'AOP mais elle fournit une solution de monitoring capable de considérer le système comme une seule entité, contrairement aux approches qui se concentrent sur le partitionnement du système. Ainsi, elle peut détecter les points de défaillance de chaque fonction aussi bien que du système dans son ensemble.

Ben Halima [13] présente dans son travail, une architecture appelée Middleware Auto-Réparable orienté QdS (MARQ), pour l'auto-réparation, partant du monitoring jusqu'à l'étape de diagnostic/pronostic. En effet, l'approche de monitoring proposée, repose sur des moniteurs appelés handlers [1] dans les approches orientées services. Ces moniteurs sont capables d'intercepter les messages SOAP échangés entre le client et le fournisseur de service et les étendre par des métadonnées décrivant les paramètres de QdS. Cette approche propose le déploiement d'un Moniteur Côté Fournisseur (MCF) et un Moniteur Côté Client (MCC). Ainsi, elle peut être appliquée pour les applications orientées services orchestrées ou chorégraphées. Cependant, les applications contenant plusieurs niveaux d'orchestration restreignent la possibilité de déploiement de certains moniteurs. De plus, cette approche ne permet pas le déploiement des moniteurs côté orchestrateurs.

2.2 Approches existantes de représentation d'architectures logicielles

Afin de présenter une architecture logicielle, un concepteur peut se servir de deux types de présentations : graphique et textuelle. Nous mettons l'accent dans cette section sur les travaux qui traitent les représentations graphiques d'architectures. Ces représentations peuvent être statiques en utilisant l'UML par exemple ou dynamiques en se servant des outils tels que les graphes. En effet, les graphes sont utilisés initialement pour la réalisation des tests de comportements d'une architecture [20] [12] [25]. C'est pour cela, on n'a rencontré que peu de travaux qui utilisent cette technique pour la représentation d'architectures.

Pour cela Heijstek et al. [21] ont mené une expérience afin d'étudier si les artefacts visuels ou textuels sont plus efficaces dans la communication des décisions de conception d'architecture logicielle. Les résultats de cette expérience montrent que ni les schémas ni les descriptions textuelles ne sont considérablement plus efficaces en ce terme. D'où, nous avons besoin de plus de recherches pour bien comprendre comment le texte et les schémas pourraient se compléter mutuellement.

Krichen et al. [23] présentent une approche formelle appelée P/S-COM+ permettant de mieux présenter les Styles d'Architecture Publier/Souscrire (PSAS). Pour cela, ils ont créé un plugin Eclipse P/S-CoM'SD qui permet de concevoir ce style architectural graphiquement. En effet, l'approche P/S-COM+ étend sa précédente P/S-CoM par l'aspect comportementale en spécifiant des règles d'interaction codées en notation Z. Cependant, elle ne supporte pas l'aspect temporel.

L'approche de Weinreich et Buchgeher [30] intègre les besoins, les scénarios et les décisions de conception dans une seule représentation architecturale cohérente, formellement définie, appelée le modèle LISA. Cette approche permet le traçage des relations entre les besoins, les décisions de conception, d'autres éléments de l'architecture et des artefacts implémentés d'une part. D'autre part, entre l'analyse d'architecture et l'analyse d'impact. Néanmoins, le modèle LISA ne permet pas de présenter les relations plus complexes.

L'approche présentée par Rui et al. [27] introduit la modélisation graphique d'architecture orientée aspect (MAO). Cette approche donne les notations visuelles de chaque concept en étendant UML en utilisant les mécanismes d'extension pour UML. Elle montre aussi comment utiliser les notations de modélisation de l'architecture orientée aspect à travers un exemple. Cependant, cette modélisation est statique, et ne permet pas de présenter l'architecture au cours de l'exécution.

Le travail de Bouassida et al. [14] propose une approche de

reconfiguration des architectures multi-niveaux. Cette approche présente ces architectures sous forme de graphes. En outre, elle utilise la grammaire des graphes en vue d'obtenir un modèle de gestion à base de règles. Ces règles gèrent à la fois les transformations d'architecture tant à l'intérieur d'un même niveau que entre plusieurs niveaux architecturaux.

Le travail de Conradi et al. [16] a comme objectif d'intégrer le travail existant de représentation de la conception dans un modèle de base commun. Ce travail est considéré comme premier effort vers un modèle de représentation de conception basé sur des graphes composés.

3. APPROCHE DE MONITORING ET DE CONSTRUCTION DE LA REPRÉSENTATION GRAPHIQUE DE L'ARCHITECTURE DES APPLICATIONS ORIENTÉES SERVICES

Notre approche est divisée en deux phases, à savoir : Le monitoring : consiste à surveiller les entités qui constituent une application orientée services ainsi que leurs interactions afin d'apercevoir les perturbations de communication qui peuvent arriver. La construction de la représentation graphique de l'architecture : consiste à reconstruire l'architecture de l'application, décrite précédemment, au cours de l'exécution et d'étudier l'évolution des données de qualité de service.

3.1 Architecture de l'approche

Dans cette section, nous présentons l'architecture de notre approche. Un Service Web appartenant à une application orientée service peut être élémentaire ou composé. Dans notre cas, nous avons eu recours à l'utilisation de la composition de Services Web par orchestration. Ce type de composition est centralisé. Un orchestrateur de Services Web est considéré comme le seul processus central qui détient la coordination et le contrôle de tous les Services Web participant à sa composition. En effet, le genre d'application orientée service que nous traitons est composé de trois types d'entités, à savoir, les Services Web élémentaires (Service Web), les orchestrateurs de Services Web et les clients de Services Web (Client de Service Web).

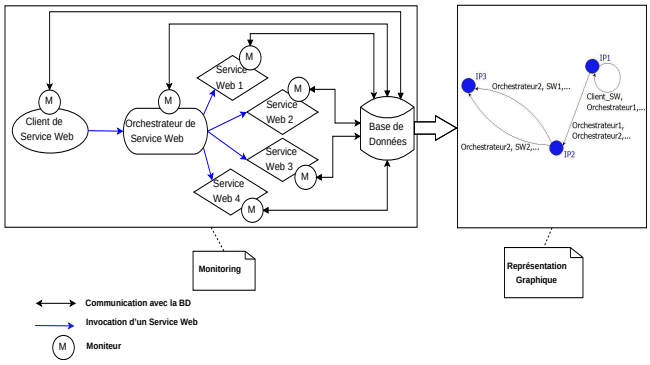

Figure 1: Architecture de l'approche

Dans le cadre de notre approche, plus précisément dans la phase de monitoring, nous avons conçu une architecture qui déploie un moniteur auprès de chaque entité. Ces moniteurs interceptent alors les messages échangés entres les entités. Puis, ils capturent les données permettant d'apercevoir les perturbations de communication, afin d'étudier l'évolution des données de qualité de service. Pour accomplir la phase de monitoring, les moniteurs enregistrent les données récupérées dans une base de données. La phase construction de la représentation graphique de l'architecture en cours de l'exécution requière l'exploitation des données de monitoring enregistrées dans la base de données (Figure 1). Ces données seront exprimées alors sous forme de différents types de graphes. En fait, dans un graphe, un nœud (Figure 1, partie droite) correspond à une entité d'une architecture qui interagit avec une autre. Les arcs illustrent les liens entre elles. Autrement dis, un arc définit le flux de messages entre deux entités. La Figure 1 illustre l'architecture de notre approche de monitoring et de construction de la représentation graphique de l'architecture.

Dans notre approche, un client de Service Web (Figure 1) appelle un orchestrateur de Services Web en vue d'obtenir un service complexe. Celui-ci -étant le seul coordinateur qui possède l'information sur la démarche de l'exécution du processus d'orchestration- invoque un nombre de Services Web (Figure 1) selon un ordre précis. En fait, chacun de ces derniers offre un service précis pour aboutir au résultat souhaité. D'autre part, les moniteurs attachés à chaque entité agissent en interceptant les messages SOAP. Cette interception a pour objectif d'enrichir le message et extraire les informations pertinentes pour étudier l'évolution des données de qualité de service, tels que les adresses sources et cibles du message, la durée d'exécution du Service Web, le temps de transfert du message, etc. Pour terminer, les moniteurs enregistrent ces informations dans une base de données. Dans la phase de construction de la représentation graphique de l'architecture en cours de l'exécution, les données de monitoring sont récupérées de la base de données. Ensuite, des calculs en vue d'obtenir les données nécessaires pour la génération des graphes sont effectués. En effet, cette étape aboutit à la génération de trois types de graphes. Nous présentons les différents types de graphes dans ce qui suit.

3.2 Phase de Monitoring

Pendant la phase de monitoring, un moniteur est déployé auprès de chaque entité d'une application orientée services. Le monitoring se base sur l'interception des messages SOAP échangés entre ces entités afin d'extraire les données permettant d'étudier l'évolution de qualité de service et détecter les perturbations de communication. Nous présentons dans un premier temps l'architecture du monitoring, puis dans un deuxième temps, nous décrivons la procédure de monitoring.

3.2.1 Architecture de la phase de monitoring

Un moniteur est une entité logicielle utilisée pour intercepter les messages SOAP. Il permet d'enrichir les messages SOAP avec des données personnalisées selon le besoin du développeur tel que le temps d'envoi ou de réception du message. Dans notre architecture de monitoring, nous mettons en place un Moniteur côté client de Service Web (Figure 2), un Moniteur côté orchestrateur de Services Web (Figure 2) et un Moniteur côté Service Web (Figure 2).

Comme mentionné dans la Figure 2 :

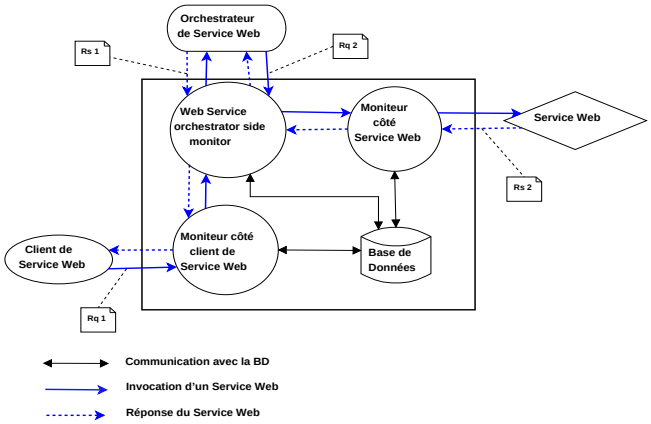

Figure 2: Architecture de monitoring

- Le Moniteur côté client de Service Web intercepte les messages échangés entre le client de Service Web et l'orchestrateur de Services Web.
- Le Moniteur côté Service Web intercepte les messages échangés entre l'orchestrateur de Services Web et le Service.
- Le Moniteur côté Orchestrateur de Services Web intercepte tous les messages arrivant et partant de l'orchestrateur de Services Web.

Tous les moniteurs communiquent avec la base de données afin d'enregistrer les données issues par l'interception des messages. Nous décrivons dans la section suivante la procédure de monitoring et les données récupérées par les moniteurs à chaque interception.

3.2.2 Description de la phase de monitoring

Comme le montre la Figure 2, un client de Service Web envoie une requête SOAP (Rq 1) à l'orchestrateur de Services Web et reste bloqué puisque notre travail se base sur la communication synchrone. Comme cette requête demande un service complexe, l'orchestrateur invoque les Services Web élémentaires nécessaires et envoie par exemple une requête SOAP (Rq 2) au Service Web. Le Service Web génère un nouveau message SOAP réponse (Rs 2) contenant le résultat de l'invocation. Apres avoir reçu toutes les réponses des Services Web invoqués, l'orchestrateur génère un nouveau message SOAP réponse (Rs 1) au client contenant le résultat de l'invocation.

Nous mettons en place (Figure 2) un moniteur côté Service Web, un moniteur côté client de Service Web et un autre côté orchestrateur pour intercepter les différents messages SOAP. Tous les moniteurs communiquent avec la base de données afin d'enregistrer les données interceptées.

En effet, pour une requête envoyé par le client de Service Web à l'orchestrateur de Service Web, le moniteur côté client de Service Web permet d'extraire l'identifiant du message (Idreq), l'adresse IP du client de Service Web (FromIp), son nom (FromWS) et le moment d'invocation de l'orchestrateur par le client (TreqOut). Le moniteur côté orchestrateur de Services Web intercepte le même message et extrait l'adresse IP de l'orchestrateur de Services Web(ToIp), son nom (ToSW), le nom de l'opération appelée (ToOp) et le moment du début de traitement de la requête par l'orchestrateur de Services Web (TreqIn).

Pour une réponse, le moniteur côté orchestrateur de Services Web extrait le moment d'envoi de la réponse (Rs 1) par l'orchestrateur (TrespOut). Enfin, le moniteur côté client de Service Web extrait le moment de réception de la réponse (Rs 1) par le client.

Les valeurs des données extraites par ces moniteurs sont résumés dans la Table 1. En effet, les cinq premiers lignes de la Table 1 présentent des données qui sont liées plus aux entités. Le reste de la Table 1 présente des données liées plus aux liens.

3.3 Construction de la représentation graphique de l'architecture

Nous avons choisi d'utiliser les graphes pour la construction de la représentation graphique d'une architecture orientée services. Pour cela, nous avons exploité les données de monitoring enregistrées dans la base de données puis nous avons calculé des grandeurs supplémentaires à savoir, la durée moyenne d'exécution d'un Service Web, le temps de transfert moyen d'une requête et d'une réponse, le nombre moyen d'appels d'un Service Web et d'une opération Dans notre travail, nous générons trois types de graphes qui sont : le graphe de Srvices Web (Figure 3), le graphe de machines (Figure 4) et le bi-graphe d'interactions qui combine les deux types précédents (Figure 5). Les trois types de graphes seront exploités afin de visualiser le flux des messages entre les différentes entités ou machines selon plusieurs points de vue : Un premier point de vue plus focalisé sur les Services Web, un deuxième point de vue plus focalisé sur les machines dans lesquelles sont déployées ces services et troisième qui fait la synthèse des deux points de vu précédents. Ces trois vues permettent d'étudier l'évolution des données de qualité de service. Cela facilite la création d'outils d'analyse de l'ensemble de données collectées pendant le monitoring.

3.3.1 Graphe de Services Web

Un nœud du graphe de Services Web correspond à un Service Web élémentaire ou composé. Quant aux arcs, ils définissent les liens entre ces Services Web.

Figure 3: Extrait du graphe de Services Web

Les nœuds du graphe de Services Web (Figure 3) représentent les Services Web. Chaque nœud est caractérisé par :

Table 1: Données de monitoring

Nom	Signification	Commentaires
FromIp	L'IP de l'entité source	L'entité source : client de Service Web, orchestrateur de Services Web
FromWS	Le nom de l'entité source	
ToIp	L'IP de l'entité cible	L'entité cible : orchestrateur de Services Web, Service Web
ToSW	Le nom de l'entité cible	
ToOp	Le nom de l'opération cible	L'opération qui exécute la tâche demandée de l'entité
Idreq	L'identifiant du message (requête ou réponse)	
TreqOut	Le temps d'envoi de la requête par l'entité source	
TreqIn	Le temps de réception de la requête par l'entité cible	
TrespOut	Le temps d'envoi de la réponse par l'entité cible	
TrespIn	Le temps de réception de la réponse par l'entité source	

Nom du Service Web, IP, Nombre d'invocations du Service Web, Nombre moyen d'invocations du Service Web et Durée moyenne d'exécution du Service Web. Les arcs représentent le flux des messages transmis entre deux nœuds. C'est-à-dire, ils illustrent les messages entre les Services Web enrichis par des informations qui concernent les opérations appelées et les durées des messages, (Nom de l'opération destination, Nombre d'invocations de l'opération destination, Nombre moyen d'invocations de l'opération destination, Temps de transfert moyen d'une requête, Temps de transfert moyen d'une réponse). Un extrait du graphe de Service Web est donné dans la Figure 3.

3.3.2 *Graphe de machines*

Les nœuds du graphe de machines (Figure 4) présentent les adresses IP des machines. Les arcs décrivent les messages transmis entre les machines en détaillants les données relatives aux Services Web appelés et appelants (Nom du Service Web source, Nom du Service Web destination, Nombre d'invocations du Service Web destination, Nombre moyen d'invocation du Service Web destination). Ces informations présentent les adresses IP des machines utilisés, les Services Web déployés sur chacune d'elles ainsi que leur importance.

Figure 4: Extrait du graphe de machines

Pour expliquer, les données comme le nom du Service Web source et le nom du Service Web destination, montre les Services Web déployés sur chaque machine. D'autre part, le nombre d'invocations du Service Web destination et le nombre moyen d'invocations du Service Web destination, montre si une machine reçoit un nombre important de demandes d'invocation de Service Web ou non. Ceci reflète l'importance de la machine ainsi que ses Services Web en se basant sur le flux de messages qu'elle reçoit. Un extrait du graphe de machines est présenté dans la Figure 4.

3.3.3 *Bi-Graphe d'interactions*

En ce qui concerne le bi-graphe d'interactions (Figure 5), il comporte deux types de nœuds : les nœuds Services Web et les nœuds machines. Cependant, il a un seul type de lien représentant le flux de messages entre les nœuds Service Web.

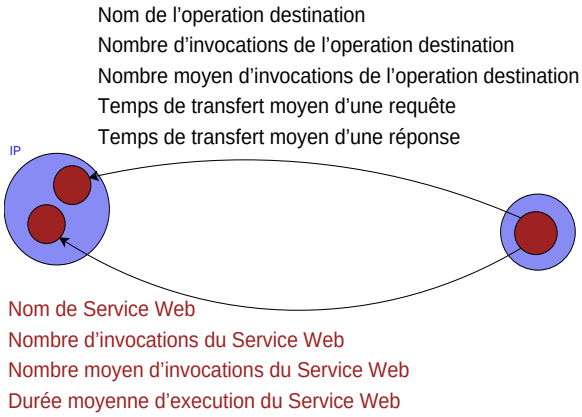

Figure 5: Extrait du Bi-Graphe d'interactions

Il résume les données introduites par les deux graphes précédents du côté nœuds (IP, Nom du Service Web, Nombre d'invocations du Service Web, Nombre moyen d'invocations du Service Web, Durée moyenne d'exécution du Service Web) aussi bien que du côté lien (Nom de l'opération destination, Nombre d'invocations de l'opération destination, Nombre moyen d'invocations de l'opération destination, Temps de transfert moyen d'une requête, Temps de transfert moyen d'une réponse). Le graphe assure ainsi une meilleure vue

globale de l'architecture de notre application. Un extrait de ce bi-graphe est donné dans la Figure 5.

4. CAS D'ETUDE SMART CITY

Face à l'augmentation de la consommation d'énergie durant ces dernières années, une attention considérable se porte sur la mise en œuvre de réseaux intelligents de distribution d'énergie. Ces réseaux utilisent des dispositifs hétérogènes (capteurs, actionneurs, etc.) en vue d'identifier et rationaliser de manière précise la consommation d'énergie des habitants d'un bâtiment, d'un quartier, d'une ville ou d'une région. La Figure 6 illustre notre cas d'étude appelé Smart City qui présente un quartier. Celui-ci contient un bâtiment appelé 'Centre de commande' et un certain nombre de bâtiments dont chacun comprend un ensemble de pièces. Une Unité de Contrôle du Quartier (UCquartier) installée dans le centre de commande est utilisée pour gérer cette structure (quartier)

Chaque bâtiment est équipé par un appareil appelé Unité de Contrôle Bâtiment (UCbâtiment). Chaque pièce du bâtiment est muni de capteurs (thermomètre, capteur de présence, capteur de luminosité), d'actionneurs (lampe, climatiseur, etc.) et une Unité de Contrôle de Pièce (UCpièce). Nous désignons les capteurs et les actionneurs par le terme 'dispositif'. Il est à noter que, les capteurs (thermomètre, capteur de présence, capteur de luminosité) installés dans chaque pièce surveillent les paramètres d'environnement. Les actionneurs (lampe, climatiseur, etc.) sont ajustés en cas de besoin par l'UCpièce. En effet, l'UCquartier gère les UCbâtiments. Ces derniers gèrent les UCpièces qui à leur tour gèrent les différents dispositifs en vue d'optimiser la consommation d'énergie. Cette optimisation permet de contribuer à la satisfaction de la demande croissante d'électricité, notamment en période de pointe. D'ailleurs, les UCpièces permettent de diminuer la consommation ou même couper l'alimentation de certains dispositifs suite à une requête distante pour éviter une défaillance du réseau électrique. Dans ce contexte, il est important d'indiquer que l'administrateur du réseau électrique peut, en se basant sur les données rassemblées par les UCpièces et les UCbâtiments, établir des prix de consommation d'électricité plus élevés aux moments de pics prévisibles selon les heures de la journée et la saison. C'est dans les UCs que sont stockés les paramètres qui permettent de maitriser la consommation comme les seuils par exemple. Cela est censé encourager les consommateurs d'énergie à rationaliser leur consommation en électricité et permettre des économies globales dans une perspective de développement durable.

Pendant l'exécution, plusieurs évènements surviennent et perturbent la communication. Ainsi, il s'avère primordiale pour le gestionnaire de Smart City de surveiller cette structure pendant l'exécution afin de savoir à titre d'exemple :
- Les dispositifs affectés par une dégradation de qualité de service
- Les messages échangés entre les dispositifs et le temps de transfert en moyenne
- Le temps de réponse de chaque dispositif

Ceci justifie l'attachement d'un moniteur à chaque dispositif/UC pour contrôler le flux de messages sortants et entrants.

5. MODÉLISATION DE CAS D'ÉTUDE

Dans notre cas d'étude, il s'agit de gérer chaque UC ou dispositif par une entité logicielle de notre approche (Service Web, orchestrateur de Services Web, client de Service Web).

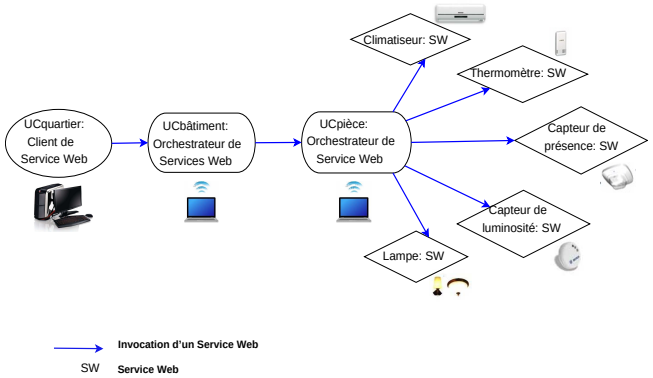

Figure 7: Modélisation de cas d'étude

La modélisation du cas d'étude Smart City est décrite par la Figure 7.

- Les dispositifs sont gérés par des Services Web élémentaires (Service Web)
- Les UCpièces sont gérées par des Services Web composés (Orchestrateur de Services Web)
- Les UCbâtiments sont gérées par des orchestrateurs de Services Web
- Les UCquartiers sont gérées par des clients de Service Web

En fait, dans notre travail, nous voulons ajouter une couche logicielle à une infrastructure qui pourrait être existante dans le quartier, le bâtiment ou la pièce. Cette infrastructure est composée par les appareils électriques que nous avons appelés les dispositifs, les UCpièces, les UCbâtiments et les UCquartiers. Nous avons implémenté des entités logicielles telles que les Services Web, les orchestrateurs de Services Web et les clients de Services Web qui gérent l'infrastructure. Donc, notre couche logicielle peut s'ajouter à une infrastructure existante. Nous avons décidés d'utiliser des machines autres que les dispositifs pour l'implémentation de la partie logicielles. Ceci permet de déployer plus qu'une entité sur chaque machine. Ceci n'exclut pas la possibilité de déployer nos Services Web sur les dispositifs intelligents qui le permettent. Ainsi, cette couche logicielle permet au gestionnaire de Smart City de gérer, activer et désactiver les UCs ou les dispositifs selon le besoin.

6. APPLICATION DE L'APPROCHE SUR LE CAS D'ETUDE

Notre approche est divisée en deux phases : monitoring et construction de la représentation graphique de l'architecture.

6.1 Phase de monitoring pour le cas d'étude

La Figure 8 décrit l'application de l'approche de monitoring pour le cas d'étude. Dans cette application, nous déployons un moniteur à coté de chaque service. Chacun de ces moniteurs enregistre les données capturées, dans une base de

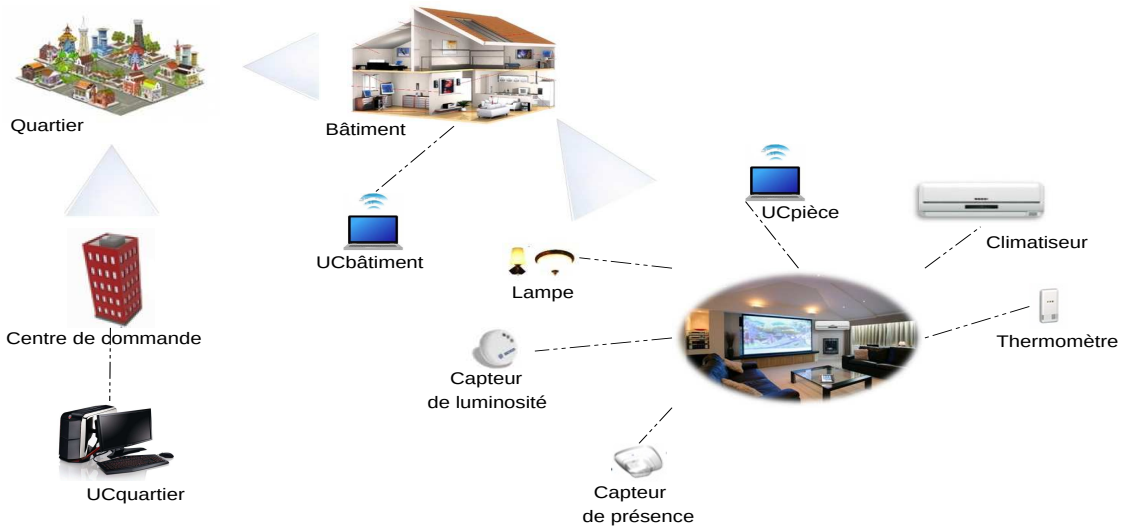

Figure 6: Cas d'étude Smart City

données. Le scenario présenté correspond à une reconfiguration de climatiseur. En fait, l'administrateur souhaite limiter l'utilisation de climatiseur : il définit une valeur de température seuil minimal. Ce qui interdit aux climatiseurs de fournir une température plus basse. D'abord, l'UCquartier envoie la température seuil à l'UCbâtiment (Rq 1) et reste bloqué. Cette requête (Rq 1) est interceptée une première fois par le Moniteur côté UCquartier et une deuxième fois par le Moniteur côté UCbâtiment.

L'UCbâtiment étant un orchestrateur de Services Web, appelle l'UCpièce en lui adressant la requête (Rq 2). Celle-ci est interceptée pareillement deux fois.

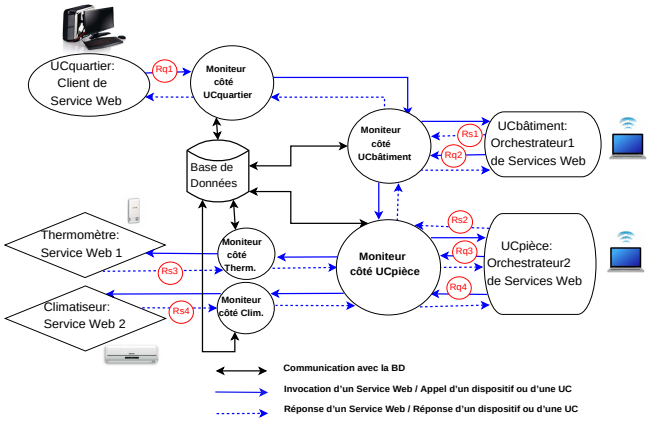

Figure 8: Phase de monitoring pour le cas d'étude Smart City

Ensuite, l'UCpièce invoque le thermomètre (Service Web 1) par l'expédition de la requête (Rq 3). Le thermomètre retourne la valeur de la température ambiante (Rs 3). En se basant sur la valeur reçue par le thermomètre, l'UCpiece compare celle-ci par la valeur seuil reçu par l'UCquartier et décide la reconfiguration du Climatiseur : atténuer, éteindre, élever, etc. Cette décision sera transmise au climatiseur (Service Web 2) par la requête (Rq 4). Une fois reconfiguré, le climatiseur retourne son nouvel état à l'UCpiece (Rs 4). L'UCpièce transfère à son tour, les données relatives à ses dispositifs à l'UCbâtiment à travers sa réponse (Rs2). Enfin, l'UCbâtiment retourne à l'UCquartier via la réponse (Rs 1), l'état actuel des dispositifs et de l'UCpièce appartenant au bâtiment, ce qui la débloque.

6.2 Phase de construction de la représentation graphique de l'architecture pour le cas d'étude

Dans cette section, nous visons appliquer la phase de construction de la représentation graphique de l'architecture pour le cas d'étude Smart City en cours de l'exécution. Afin d'illustrer le comportement normal de l'application, nous avons exécuté le scénario de test décrit par la Figure 8, 110 fois. Ceci nous donne trois graphes illustrés par les diagrammes des figures 9, 10 et 11.

6.2.1 Graphe de Service Web

Le graphe de Service Web illustré par la Figure 9 comporte cinq nœuds représentant les cinq entités qui participent dans le scenario, à savoir : l'UCquartier (UCquartier), l'UCbâtiment (UCbâtiment), l'UCpièce (UCpiece), le thermomètre (Thermometre) et le climatiseur (Climatiseur). En plus du nom du Service Web, chaque nœud affiche l'adresse IP dans laquelle le Service Web est déployé, le nombre de fois qu'il est invoqué pendant l'exécution de 440 messages aussi bien que le nombre de fois qu'il est invoqué pendant l'exécution de 440 messages par rapport au nombre total d'invocations et enfin le temps moyen qu'exige le Service Web pour l'exécution d'une requête.

En ce qui concerne les liens, chacun porte le nom de l'opération appelée tel que 'ConfigurerClimatiseur', le nombre de fois que celle-ci est invoquée pendant l'exécution de N messages, de plus le nombre de fois que l'opération est invoquée pendant l'exécution de N messages par rapport au nombre total d'invocation.

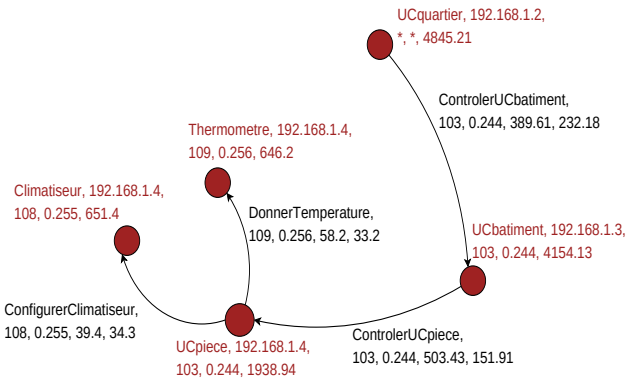

Figure 9: Graphe de Services Web

6.2.2 *Graphe de machines*

Le graphe de machines (Figure 10), contient trois nœuds. Chaque nœud représente une machine et n'affiche que son adresse IP. Chaque lien porte le nom de Service Web source, le nom de Service Web destination, le nombre de fois que le Service Web destination est invoqué pendant l'exécution de 440 messages et le nombre de fois qu'il est invoqué pendant l'exécution de 440 messages par rapport au nombre total d'invocation.

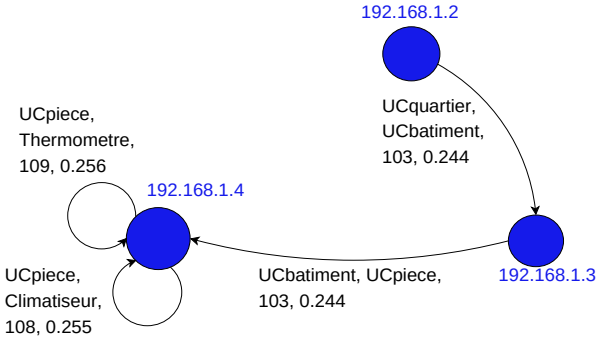

Figure 10: Graphe de machines

6.2.3 *Bi-Graphe d'interactions*

Le bi-graphe d'interactions (Figure 11) résume les informations introduites par les deux graphes précédents. Nous observons clairement les machines que nous avons exploitées dans notre application et les Services Web installés dans chacune d'elles.

Après l'exécution du scénario illustré par la Figure 8, 110 fois, puis la génération du graphe des Services Web, du graphe des machines et du Bi-graphe d'interactions, nous constatons que ces graphes représentent fidèlement l'application.

7. IMPLÉMENTATION ET EXPÉRIMENTATIONS

Dans notre travail, nous utilisons l'environnement de développement eclipse qui se base sur Java EE. Nous l'avons choisit afin de tirer profit des avantages de l'utilisation conjointe du Serveur Tomcat et de l'implémentation de Services Web

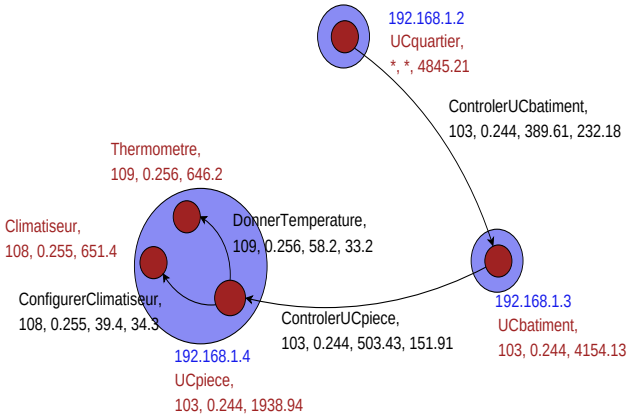

Figure 11: Bi-Graphe d'interactions

Axis2. Cela permet de développer facilement des applications en java puis de les convertir sous forme de Services Web à l'aide d'Axis2 et enfin de les déployer sous Tomcat. Ainsi, les orchestrateurs de Services Web exécutent le processus décrit par le langage BPEL. Cela permet d'invoquer des Services Web élémentaires en vue d'obtenir un Service Web composé. Ce dernier est présenté pareillement sous forme WSDL mais déployé sous le moteur Apache-Ode qui est le moteur de déploiement des processus d'orchestration.

Dans un premier lieu nous implémentons notre cas d'étude qui comporte l'implémentation des Services Web, des orchestrateurs de Services Web et des clients de Services Web. Nous créons et déployons les moniteurs sur les Services Web du cas d'étude en deuxième lieu.

Finalement, la représentation graphique d'architecture sous forme de graphes qui comporte le calcul des différentes grandeurs et la construction des graphes en graphML en cours de l'exécution, est effectuée automatiquement après lancement de l'application.

7.1 Comportement de l'application après provocation d'une perturbation de communication

L'objectif derrière cette expérimentation est de prouver que grâce aux données de monitoring et à la représentation en graphes, nous avons réussi à identifier les perturbations de communication provoquées. En effet, nous avons déroulé le même scénario de test décrit par la Figure 8, 110 fois, tout en provoquant un retard de 1000 ms pendant l'exécution du Service Web 'Climatiseur'. Ainsi, nous avons obtenu le graphe de Services Web décrit par la Figure 12 que nous comparons avec la Figure 9 (Exécution dans le cas normal).

En observant la Figure 12, qui présente le graphe de Services Web. L'augmentation de délai est visible sur le noeud du Service Web 'Climatiseur' ainsi que sur les orchestrateurs 'UCpiece' et 'UCbatiment', et le client de Service Web 'UCquartier'. Ainsi, nous constatons que ce graphe représente fidèlement l'application.

7.2 Coût de monitoring

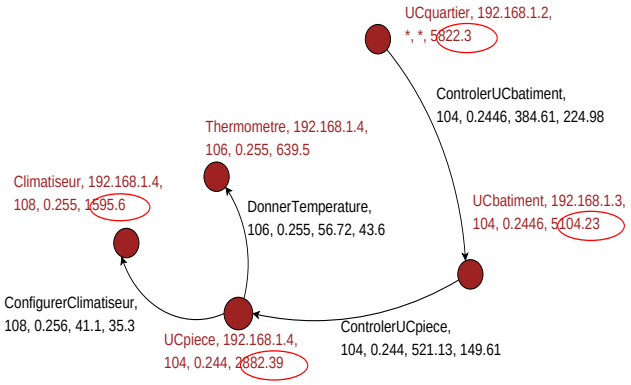

Figure 12: Graphe de Services Web après la provocation d'une perturbation de communication

Dans ce paragraphe nous exécutons le scénario : Comportement normal de l'application un certain nombre de fois pour observer le temps additionnel qu'ajoute les moniteurs au temps d'exécution de l'application. Nous utilisons deux machines de 4Go de RAM et Intel Pentium P6200 2.13GHz comme processeur et une machine de 6Go de RAM et Intel Core i3 2330M 2.2GHz comme processeur avec un réseaux LAN 100 MB. Nous représentons l'évolution du temps d'exécution du système a chaque envoi de 50 messages par le client de l'application.

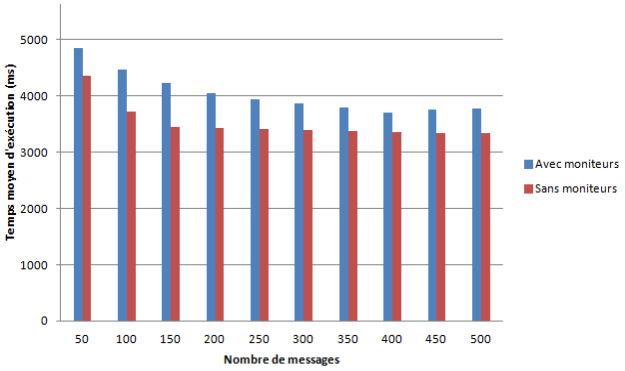

Figure 13: Le coût introduit par le module de monitoring en fonction de nombre de messages

La Figure 13, montre que le temps d'exécution avec l'introduction des moniteurs est très légèrement supérieur à celui mesuré sans l'introduction de ceux-ci. En effet, le temps d'exécution moyen de 50 messages est de 4350.32 ms sans les moniteurs et il est environ de 4845.21 ms après avoir introduit les moniteurs. Pour 500 messages, nous remarquons que le délai varie entre 3330.4 ms sans monitoring et 3779 ms après avoir introduit les moniteurs. L'opération de monitoring prend 10% du temps total d'exécution.

8. CONCLUSIONS

Nous avons présenté dans ce travail une approche de monitoring pour la construction de la représentation graphique de l'architecture des applications orientées services. Notre défi consiste ainsi à suivre le flux de messages entre les différentes entités de l'application dans le but de construire la représentation graphique de l'architecture du système en cours d'exécution.

Pour réaliser ces objectifs, nous avons mis l'accent en premier lieu sur la phase de monitoring qui vise à suivre l'évolution des métriques de qualité de service. Cette surveillance est à base d'interception de messages échangés entre les différentes entités de l'application à savoir les Services Web, les orchestrateurs de Services Web ou processus BPEL et les clients de Services Web. En effet, cette interception est assurée par des moniteurs que nous avons développé et déployé au sein de chaque entité. Les données tirées par ces trois types de moniteurs sont enregistrées dans une base de données.

Une fois enregistrées, la phase de construction de représentation graphique d'architecture exploite les données tirées de la phase de monitoring qui sont liées aux entités. Nous avons procédé à une phase de calcul afin obtenir d'autres données qui sont liées à l'exécution, ou à la communication Ces données seront représentées sous forme de graphes en cours de l'exécution de l'application. En fait, nous avons généré trois types de graphes selon différents points de vue : le graphe de Services Web qui donne un point de vue plus focalisé sur les Services Web, le graphe de machines qui donne un point de vue plus focalisé sur les machines dans lesquelles sont déployées ces services et le bi-graphe d'interactions qui combine les deux types précédents et assure ainsi une vue globale de l'architecture de l'application. Ceci assure une meilleure visualisation pour l'étude de l'évolution des métriques de qualité de service.

Nous avons ensuite présenté notre cas d'étude appelé Smart City. Puis, nous avons appliqué notre approche sur ce cas d'étude. Dans la dernière partie de notre travail, nous avons exécuté quelques scénarios de test afin d'observer les résultats issus des moniteurs déployés sur les entités de notre application à travers les graphes. Ces résultats illustrent les interactions et les messages échangés entre les différentes entités et permettent d'observer s'il y a une perturbation de communication. Ceci permet ultérieurement de reconfigurer l'application en cas de pannes ou dégradations de qualité de service.

En perspective, nous envisageons à moyen terme, à étendre notre travail pour supporter les processus BPEL asynchrone et la chorégraphie comme moyen de composition. Nous visons également la prise en compte d'autres métriques de performance tel que le taux de pertes des messages. Nous souhaitons à long terme, l'intégration de notre approche de monitoring et de construction de représentation graphique d'architecture dans un processus d'auto-adaptabilité complet afin de la valider.

9. REMERCIEMENTS

Ce travail est supporté par le projet ITEA2 A2NETS (Autonomic Services in M2M Networks) [1].

10. REFERENCES

[1] Axis architecture guide, 1.4.1 version, april 2006, 2006.

1. urlhttps ://a2nets.erve.vtt.fi/

[2] W3c web services policy working group 2007, wspolicy 1.5., 2007.

[3] K. An, S. Pradhan, F. Caglar, and A. Gokhale. A publish/subscribe middleware for dependable and real-time resource monitoring in the cloud. In *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, SDMCMM '12, pages 3–6, New York, NY, USA, 2012. ACM.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, 2003.

[5] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, and J. Rofrano. Web services agreement specification (ws-agreement). *Global Grid Forum*, 2004.

[6] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *Web Services, 2006. ICWS '06. International Conference on*, pages 63–71, 2006.

[7] L. Baresi and S. Guinea. Towards dynamic monitoring of ws-bpel processes. In *Proceedings of the Third International Conference on Service-Oriented Computing*, ICSOC'05, pages 269–282, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] L. Baresi and S. Guinea. A dynamic and reactive approach to the supervision of bpel processes. In *Proceedings of the 1st India Software Engineering Conference*, ISEC '08, pages 39–48, New York, NY, USA, 2008. ACM.

[9] L. Baresi and S. Guinea. Self-supervising bpel processes. *Software Engineering, IEEE Transactions on*, pages 247–263, 2011.

[10] L. Baresi, S. Guinea, O. Nano, and G. Spanoudakis. Comprehensive monitoring of bpel processes. *Internet Computing, IEEE*, pages 50–57, 2010.

[11] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti. Dynamo + astro : An integrated approach for bpel monitoring. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 230–237, 2009.

[12] F. Belli and M. Linschulte. Event-driven modeling and testing of web services. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, pages 1168–1173, 2008.

[13] R. Ben halima. *Conception, implantation et expérimentation d'une architecture en bus pour l'auto-réparation des applications distribuées à base de services Web*. PhD thesis, Université Paul Sabatier, Toulouse-France and Université de Sfax-Tunisie, 2009.

[14] I. Bouassida, C. Chassot, and M. Jmaiel. Graph grammar-based transformation for context-aware architectures supporting group communication. *Revue des Nouvelles Technologies de l'Information*, pages 29–42, 2010.

[15] K. Bratanis, D. Dranidis, and A. Simons. An extensible architecture for run-time monitoring of conversational web services. In *Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond*, MONA '10, pages 9–16, New York, NY, USA, 2010. ACM.

[16] P. Conradi. Information model of a compound graph representation for system and architecture level design. In *Design Automation Conference, 1995, with EURO-VHDL, Proceedings EURO-DAC '95., European*, pages 22–27, 1995.

[17] A. Erradi, P. Maheshwari, and V. Tosic. Ws-policy based monitoring of composite web services. In *Proceedings of the Fifth European Conference on Web Services*, ECOWS '07, pages 99–108, Washington, DC, USA, 2007. IEEE Computer Society.

[18] D. Garlan, S.-W. C., A.-C. H., B. Schmerl, and P. Steenkiste. Rainbow : architecture-based self-adaptation with reusable infrastructure. *Computer*, pages 46–54, 2004.

[19] Y. Giljong and L. Eunseok. Monitoring methodology using aspect oriented programming in functional based system. In *Proceedings of the 12th international conference on Advanced communication technology*, ICACT'10, pages 783–786, Piscataway, NJ, USA, 2010. IEEE Press.

[20] F. Golshan and A. Barforoush. A new approach for tracing quality attributes in service oriented architecture using graph transformation systems. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 10–16, 2009.

[21] W. Heijstek, T. Kuhne, and M. R. V. Chaudron. Experimental analysis of textual and graphical representations for software architecture design. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ESEM '11, pages 167–176. IEEE Computer Society, 2011.

[22] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP*. SpringerVerlag, 1997.

[23] I. Krichen, I. Loulou, H. Dhouib, and A. Hadj Kacem. P/s-com+ : A formal approach to design correct publish/subscribe architectural styles. In *Proceedings of the 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, ICECCS '12, pages 179–188, Washington, DC, USA, 2012. IEEE Computer Society.

[24] O. M., F. R., and S. D. Non-intrusive monitoring and service adaptation for ws-bpel. In J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China*, pages 815–824. ACM, 2008.

[25] F. Mahdian, V. Rafe, R. Rafeh, and M. Miralvand. Considering faults in service-oriented architecture : A graph transformation-based approach. In *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, volume 1, pages 179–183, 2009.

[26] M. Psiuk, T. Bujok, and K. Zielinski. Enterprise service bus monitoring framework for soa systems. *Services Computing, IEEE Transactions on*, pages 450–466, 2012.

[27] W. R., M. X.G., D. Z.Y., and W. Y. Extending uml for aspect-oriented architecture modeling. In *Computer Science and Engineering, 2009. WCSE '09. Second International Workshop on*, volume 1, pages

362–366, 2009.

[28] M. Sun, B. Li, and P. Zhang. Monitoring bpel-based web service composition using aop. In *Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*, ICIS '09, pages 1172–1177, Washington, DC, USA, 2009. IEEE Computer Society.

[29] D. Tovarnak and T. Pitner. Towards multi-tenant and interoperable monitoring of virtual machines in cloud. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 436–442, 2012.

[30] R. Weinreich and G. Buchgeher. Integrating requirements and design decisions in architecture representation. In *Proceedings of the 4th European conference on Software architecture*, ECSA'10, pages 86–101, Berlin, Heidelberg, 2010. Springer-Verlag.