π-Diapason : un langage pour la formalisation des architectures orientées services Web

Frédéric Pourraz* — Hervé Verjus* — Flavio Oquendo**

* *LISTIC*
Université de Savoie
B.P. 806, F - 74016 Annecy Cedex
{frederic.pourraz, herve.verjus}@univ-savoie.fr

** *VALORIA*
Université de Bretagne-Sud
B.P. 573, F - 56017 Vannes Cedex
flavio.oquendo@univ-ubs.fr

RÉSUMÉ. *Cet article présente l'utilisation d'une approche centrée architecture appliquée aux Architectures Orientées Services Web et en particulier à la composition de services Web. Nous avons pour objectif d'offrir un langage de haut niveau, spécifique au domaine de l'orchestration de Services Web : le langage π-Diapason. Ce dernier est fondé formellement sur les bases du π-calcul et reprend certaines caractéristiques des langages d'orchestration de services Web. Nous nous sommes intéressés à l'interprétation d'une description π-Diapason par une machine virtuelle afin d'en vérifier le comportement. Pour ce faire, nous utilisons un vérificateur de méta modèles appelé MMC que nous avons étendu pour la prise en compte et l'interprétation du langage π-Diapason.*

ABSTRACT. *This paper presents an architecture centric approach applied to Web services oriented architectures and Web services composition. We aim at providing a high level Web services choreography specific language: π-Diapason. This later is formally based on the π-calculus and reuses most characteristics of Web services choreography languages. We were interested in interpreting π-Diapason description by using a virtual machine in order to check its behaviour. To do this, we use a meta model checker called MMC and we extend it in order to cover and interpret the π-Diapason language.*

MOTS-CLÉS : *Approche centrée architecture, architectures orientées services Web, orchestration, composition de services Web, π-calcul, π-Diapason, MMC.*

KEYWORDS: *Architecture centric approach, Web services oriented architectures, orchestration, Web services composition, π-calculus, π-Diapason, MMC.*

1. Introduction

Dans le monde industriel et des services, les applications e-business, e-administration, e-gouvernement, l'utilisation des services web est de plus en plus courante. Pour assurer l'interopérabilité de gros composants logiciels, d'applications patrimoines ou à l'étagère (COTS), les services web font partie de la couche d'intermédiation (Pourraz et al., 2006). Reposant sur des standards promus par le W3C, les services Web sont parfois vus comme une réponse assez peu coûteuse aux difficultés que connaissent les organisations pour gérer l'interopérabilité entre applications hétérogènes et ce, dans un contexte de distribution à large échelle. Dans ce cadre, le processus (*business process*) sert à la fois à définir le logique métier, tout autant qu'il constitue le moyen d'interconnecter de telles applications (on parle alors d'intégration par le processus). Les solutions EAI, par exemple, constituent une illustration de nos propos. Des formalismes (WSFL, XLANG, BPEL4WS, etc.) ont été proposés pour orchestrer des services web en mettant le processus au cœur du dispositif, selon une approche de type BPM (d'autres approches existent pour la composition de services web – voir, par exemple (Rao et Su, 2004)). Pour autant, des problèmes accompagnent l'utilisation de tels langages : diversité des concepts (méta-modèles) des processus mis en œuvre, incohérence, manque de maîtrise dans la définition des processus, réutilisation de fragments de processus (d'orchestrations) qui est souvent compliquée voire impossible, et d'autres limitations inhérentes aux formalismes eux-mêmes (van der Aalst et al., 2003a). Un certain nombre de travaux et de positions ont montré que l'utilisation d'approches formelles pourrait constituer une avancée significative dans ce cadre (en particulier parce que le contexte du web pose un certain nombre de problème – sécurité, aléas dus à l'état du réseau, pannes, mise à disposition de services sur un réseau que personne ne maîtrise, etc.). Aussi, dans un contexte professionnel, la conception d'une application basée sur l'utilisation de services doit répondre aux attentes et offrir également un certain nombre de garanties. Dans ce contexte, les attributs qualités, propriétés au sens large de telles architectures orientées services web doivent être étudiées. Ce n'est en soi, pas vraiment un phénomène nouveau puisque de nombreux travaux s'intéressent à la caractérisation des propriétés ou à la séparation des propriétés dites non fonctionnelles de la logique fonctionnelle de l'application (programmation par aspects, séparation des préoccupations par exemple, etc.).

L'approche que nous proposons, issue de nos récents travaux dans le cadre du projet ArchWare (Oquendo et al., 2004) repose sur l'utilisation d'une approche centrée architecture appliquée aux Architectures Orientées Services Web (AOSW). Selon l'approche ArchWare (voir figure 1), une architecture abstraite spécifiée formellement à l'aide d'un langage couvrant les aspects structurels et comportementaux peut ensuite être raffinée de manière à obtenir une architecture concrète (généralement le code de l'application). Bien entendu, l'ensemble des étapes de raffinement doit se faire en préservant les propriétés de l'architecture initiale.

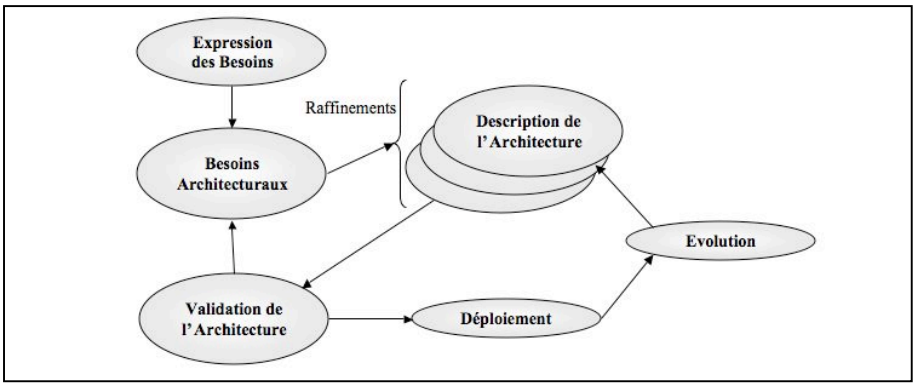

Figure 1. *Approche centrée architecture ArchWare*

Notre approche qui s'inspire de l'approche ArchWare a pour objectifs : (1) d'offrir à l'architecte un langage de haut niveau (π-Diapason), spécifique au domaine des Architectures Orientées Services Web (AOSW), langage reposant sur des fondements formels ; (2) de permettre l'interprétation d'une description π-Diapason par une machine virtuelle pour en vérifier le comportement ; (3) de considérer les attributs qualité pour les AOSW ; (4) de déployer des AOSW et de permettre leur exécution ainsi que leur évolution ; (5) de fournir un cadre conceptuel au développement des AOSW permettant la définition et la réutilisation de fragments d'architectures. D'autres travaux dans le domaine des services web sont davantage axés sur la caractérisation sémantique des services (Cabral et al., 2004) ou/et la découverte de services existants. Nos travaux offrent, nous le pensons, une vision complémentaire puisque nous partons d'une hypothèse selon laquelle les services web sont des boîtes noires dont seule l'interface (WSDL) nous est connue contrairement aux approches sémantiques qui, bien souvent, proposent d'étendre les descriptions WSDL. Des travaux similaires ont été engagés (Foster et al., 2003, Salaün et al., 2004) ; cependant, ils se concentrent sur un des aspects du développement d'applications basées sur l'utilisation de services Web sans couvrir l'ensemble du processus de développement (depuis la spécification jusqu'à considérer l'évolution d'une AOSW). Par exemple, des travaux portent sur la formalisation d'une AOSW avec des algèbres de processus (Salaün et al., 2004), sur la vérification d'orchestrations de services Web (Betin-Can et al., 2005), sur le couplage d'approches formelles et sémantiques (Rao et al., 2006), mais d'aucun considèrent le processus de développement dans sa globalité.

Nous nous focaliserons, dans le contexte de cet article, sur les principes du langage π-Diapason, permettant de formaliser des Architectures Orientées Services Web, ainsi que l'interprétation et la validation des AOSW exprimées à l'aide de π-Diapason (section 2). Nous proposerons en section 3 les travaux que nous menons actuellement en complément du langage π-Diapason ainsi que certaines perspectives à court terme.

2. Une approche formelle, centrée architecture, pour la formalisation, l'interprétation et la validation des AOSW

Dans le domaine de la composition des services Web, quelques travaux sont consacrés à l'orchestration de services Web (van der Aalst et al., 2003a) et proposent des langages issus des approches de type BPM (où le processus est le moyen d'intégration). Les récents travaux dans le domaine des architectures logicielles se sont intéressés à leurs descriptions en considérant : (1) la structure de l'architecture, (2) son comportement et (3) les propriétés de l'architecture. En particulier, l'approche centrée architecture ArchWare a permis de formaliser des scénarios basés sur des processus industriels et EAI (Pourraz et al., 2006). D'un côté nous disposons de langages adaptés à la problématique de l'orchestration de services Web (tels les langages WSFL, XLANG, BPEL4WS) et de l'autre, des langages de haut niveau permettant l'expression de modèles architecturaux (couvrant également les aspects processus) et de raisonner sur ces modèles.

Nos travaux actuels ont pour objectif de fournir un langage de haut niveau, adapté à l'orchestration des services Web, combinant les avantages issus des travaux des deux domaines précédemment cités. Le pouvoir d'expression de π-Diapason n'a pas la prétention d'être beaucoup plus complet que des langages d'orchestration tels que BEPL4WS par exemple ; mais ses fondements ont pour objectif de le placer comme le langage pivot d'un environnement centré architecture, pour le développement des AOSW. Nous avons mené un certain nombre d'études pour définir les caractéristiques du formalisme que nous proposons :

- une étude couvrant les formalismes de type BPM qui nous a permis de caractériser les langages d'orchestration de services Web. En particulier, (van der Aalst et al., 2003b) ont mis en évidence 20 patrons d'orchestration. Après avoir identifié ces patrons dans quelques langages d'orchestration (WSFL, BPEL4WS), et des cas d'utilisation que nous avons rencontrés, nous avons retenus 17 patrons (les trois patrons restant étant assez peu pertinents pour l'orchestration de services Web) ;

- le π-calcul (Milner, 1999) comme fondement théorique de nos travaux et qui sert de base à l'expression structurelle et comportementale des AOSW.

Nous proposons un langage spécifique au domaine (π-Diapason), qui d'une part reprend les patrons identifiés et d'autre part permet l'expression de l'architecture en π-calcul. Ce langage présente les caractéristiques suivantes : (1) les services Web sont les éléments architecturaux, dont les principales caractéristiques sont obtenues par leur WSDL (en particulier la description des opérations - leurs noms ainsi que leurs différents paramètres d'entrée/sortie) ; (2) couvre 17 patrons d'orchestration pour la gestion des flux de contrôle ; (3) propose un patron particulier d'orchestration pour la prise en compte des flux de données ; (4) est un langage ne manipulant que les concepts propres aux AOSW (Langage Spécifique au Domaine). Nous adoptons une approche de type boîte noire pour laquelle le comportement interne des services Web que nous orchestrons nous est inconnu (dans le cas

général). On distingue deux types de services Web : les services Web que l'on appelle *atomiques* (dont le comportement interne est inconnu) et les services Web que l'on appelle *complexes* qui invoquent d'autres services et qui sont (ou peuvent être) eux-mêmes l'expression d'une orchestration en π-Diapason. Chaque expression d'une architecture AOSW écrite en π-Diapason, est, par construction du langage, une expression π-calcul qui peut ainsi être vérifiée. Ainsi, chaque patron d'orchestration est formalisé comme un processus π-calcul.

```
service calculator {
    url = "http://...";
    operation multiply {
        request { integer int1; integer int2; }
        response { integer result; } }
    operation divide {
        request { integer int1; integer int2; }
        response { integer result; } } }

complexService scientific_calculator {
    url = "http://...";
    operation percentage {
        request { integer int1; integer int2; }
        response { integer result; } }
        orchestration {
            invoke calculator.multiply( int1, int2 );
            invoke calculator.divide( calculator.multiply.response.result, 100 );
            return calculator.divide.response.result; } } }
```

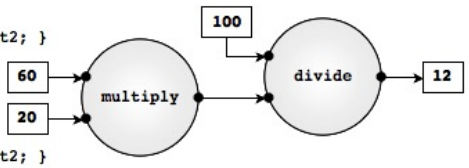

Figure 2. *Expression d'une AOSW (service complexe) décrite en π-Diapason*

π-Diapason propose une syntaxe concrète qui couvre les différents concepts et informations sur les services Web et leur composition. Pour tout service, il faut décrire son nom, son url d'invocation, ainsi que l'ensemble des opérations qui seront utilisées dans le cadre d'une AOSW (dont les invocations se feront en fonction de l'orchestration décrite). Chacune des opérations est interprétée comme un processus π-calcul qui reçoit des paramètres, exécute son comportement interne (la notion de boîte noire est interprétée comme un « tau » en π-calcul) et renvoie une valeur de retour. L'ensemble de ces paramètres ainsi que leurs types associés est exprimé en π-Diapason. Le type des paramètres reçus et envoyés par les opérations est défini en s'inspirant des recommandations du W3C sur la définition de Schéma XML (Bray et al., 2004) : les types simples (string, integer, etc.) et les types complexes (collections de types simples ou complexes). π-Diapason permet d'exprimer l'ensemble des opérations et des types utilisés au sein d'une orchestration. L'orchestration des invocations des opérations décrites est décrite en utilisant les patrons que nous avons retenus et formalisés. Dans la terminologie π-calcul, un patron d'orchestration est vu comme un processus possédant une ou plusieurs connexions (en fonction du comportement interne du patron). L'unification[1] de ces différentes connexions va

[1] Afin de pouvoir communiquer, deux connexions doivent être de même type et doivent être reliées entre elles ; dans notre cas, elles doivent avoir le même nom.

permettre d'exprimer le comportement du processus global d'orchestration de l'AOSW formalisée. Cette dernière va être définie comme un élément architectural à part entière qui pourra être réutilisé dans d'autres orchestrations. L'expression π-Diapason d'une AOSW est donc l'expression d'un service complexe (selon la terminologie que nous avons adoptée). La Figure 2 présente un exemple simple de description d'une AOSW permettant d'invoquer des opérations de multiplication et de division selon une orchestration donnée.

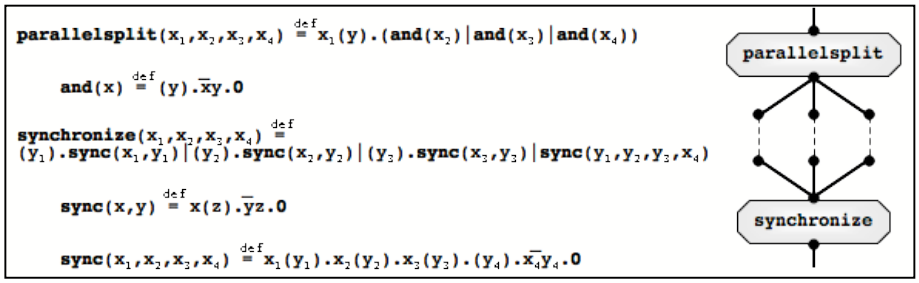

Figure 3. *Description π-calcul de patrons d'orchestration*

Fondé sur le π-calcul, une AOSW décrite en π-Diapason est directement formalisée. La Figure 3 présente la définition π-calcul de deux patrons π-Diapason : le parallélisme et la synchronisation. Une fois formalisée en π-calcul, notre approche consiste à interpréter, valider et exécuter notre architecture en utilisant une machine virtuelle. Pour cela, nous avons implémenté une machine virtuelle π-calcul dans le langage XSB Prolog sous la forme de faits et de prédicats. Cette implémentation est basée sur un vérificateur de méta modèles appelés MMC (Yang et al., 2004), lui aussi fondé sur le π-calcul. L'ensemble de cette approche est décrit en Figure 4.

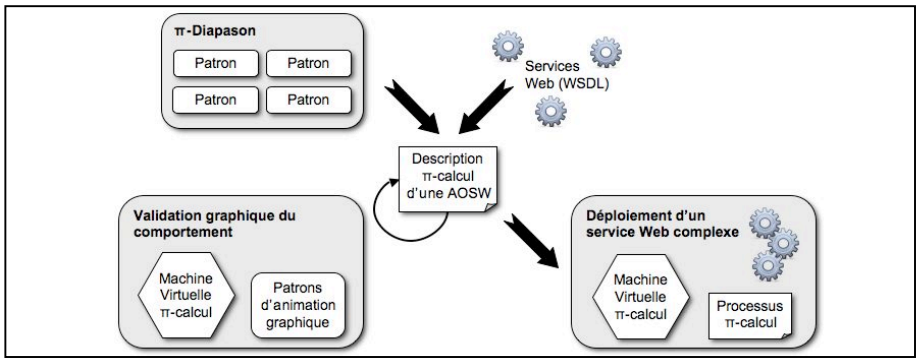

Figure 4. *Formalisation, interprétation, validation et déploiement d'une AOSW*

Une validation intuitive du comportement d'une AOSW (donc son orchestration et l'ensemble des éléments architecturaux – les services) peut être réalisée en utilisant un animateur graphique d'architectures logicielles que nous avons développé au cours du projet ArchWare (Azaiez et al., 2004). Cet animateur (voir Figure 5), permet de représenter graphiquement la structure et le comportement d'une architecture décrite en π-calcul. Cet outil intègre notre machine virtuelle du π-calcul à laquelle nous avons connecté des patrons d'animation graphique. Chaque état atteint par la machine virtuelle déclenche une animation graphique (patron d'animation) correspondant à l'action exécutée : un patron d'animation est une représentation graphique de l'interprétation d'une structure particulière du langage π-calcul. Au cours de la validation, aucun service Web n'est réellement invoqué (on peut voir la validation comme une simulation du comportement de l'orchestration d'une AOSW).

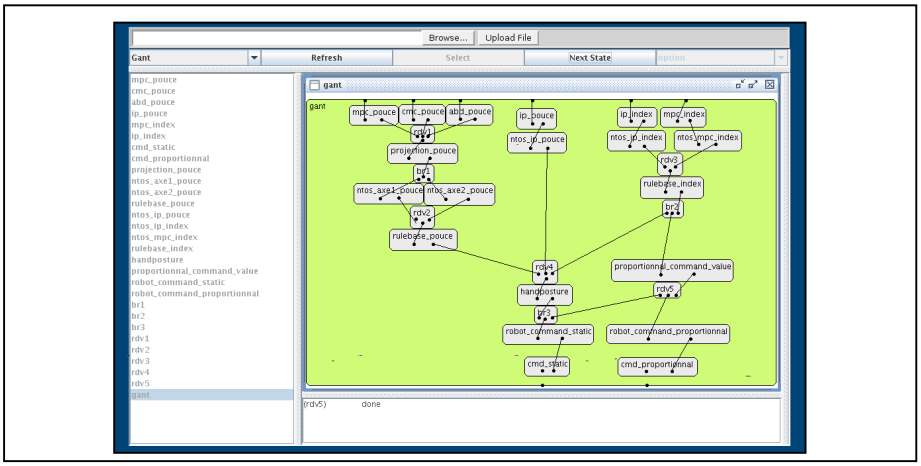

Figure 5. *Capture d'écran de l'animateur d'architectures logicielles*

Après avoir décrit et formalisée notre AOSW, en utilisant le langage π-Diapason, et validée intuitivement son comportement par animation visuelle, notre approche permet son déploiement et son exécution. Le déploiement se fait sur une plateforme Tomcat/Axis ; l'orchestration est déployée sous forme d'un service Web à part entière, un service complexe selon notre terminologie. Une simple classe Java (déployée en tant que service Web sous Axis) contient notre machine virtuelle et la représentation π-calcul de l'AOSW. L'invocation de ce service sur le Web a pour effet d'invoquer la machine virtuelle qui interprète le comportement interne du service décrit en π-calcul. À l'inverse de l'animation, les services atomiques (existants et déployés par ailleurs) qui composent notre architecture sont réellement invoqués selon l'orchestration qui a été formalisée. Le fait de disposer d'une machine virtuelle comporte plusieurs avantages : le premier est de pouvoir exécuter

directement l'architecture qui a été validée (les spécifications sont directement interprétées), sans générer de code exécutable (Java, C++, …) ; ceci permet d'éviter les erreurs, les pertes sémantiques ou des comportements non souhaités (et non validés) qui accompagnent généralement toute génération de code à partir de description abstraite de haut niveau. Le second avantage est de pouvoir plus facilement modifier notre architecture, sans avoir à générer à nouveau du code exécutable, en ne modifiant que l'expression π-calcul de l'architecture (et donc pouvoir valider à nouveau l'architecture ainsi modifiée). De plus, l'orchestration d'une AOSW étant déployée sous forme d'un service Web, elle peut, elle aussi, être réutilisée comme une boîte noire et devenir à son tour un élément architectural d'une autre AOSW.

3. Conclusion et perspectives

Seule la partie couvrant la formalisation et l'interprétation de notre approche portant sur le développement, le déploiement et la maintenance des AOSD a été illustrée dans le cadre de cet article. Nos travaux actuels ont pour objectif (1) de proposer un langage basé sur le μ-calcul (Kozen, 1983), qui sert de base à l'expression des propriétés des AOSW ; (2) de définir des attributs qualité pour les AOSW – ces attributs qualité seront exprimés sous forme de règles de transformation, permettant le raffinement (horizontal) d'une AOSW ; (3) de permettre l'obtention de l'architecture à déployer par raffinement vertical ; (4) de proposer un environnement complet supportant les étapes majeures d'un processus centré architecture spécifique aux AOSW. A notre connaissance, aucun environnement de ce type n'est actuellement proposé.

4. Bibliographie

Azaiez S., Pourraz F., Verjus H., Oquendo F., « Validation By Animation : Animating Software Architectures Based On pi-calculus », *Workshop on Systems Testing and Validation (SV04)*, Paris, France, décembre 2004, pp. 93-103.

Betin-Can, A., Bultan, T., and Fu, X. 2005. « Design for verification for asynchronously communicating Web services ». In *Proceedings of the 14th international Conference on World Wide Web*, Chiba, Japon, mai 2005.

Cabral L., Domingue J., Motta E., Payne T., Hakimpour F., « Approaches to Semantic Web Services: An Overview and Comparisons », *1st European Semantic Web Symposium (ESWE 2004)*, Grèce, mai 2004.

Foster H.,Uchitel S.,Magee J.,Kramer J. « Model-based verification of web service compositions ». *In Proceedings of the 18th International Conference on Automated Software Engineering*, 2003.

Kozen D., « Results on the Propositional μ-calculus », *Theoretical Computer Science*, 27:333-354, 1983.

Milner R., « Communicating and Mobile Systems: the pi-calculus », *Cambridge University Press*, 1999.

Bray T., Paoli J., Sperberg-McQueen C. M., Maler E., Yergeau F., « Extensible Markup Language (XML) 1.0 (Third Edition) », *W3C Recommendation 04*, February 2004.

Oquendo F., Warboys B., Morrison R., Dindeleux R., Gallo F., Garavel H., Occhipinti C., « ArchWare: Architecting Evolvable Software ». *In proceedings of the first European Workshop on Software Architecture (EWSA 2004)*, pages 257-271, St Andrews, UK, mai 2004.

Pourraz F., Verjus H., Oquendo F., « An Architecture-Centric Approach For Managing The Evolution Of EAI Service-Oriented Architecture », *In proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS'06)*, mai 2006 (à paraître).

Rao J., Su X., « A Survey of Automated Web Service Composition Methods ». *In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, San Diego, California, USA, juillet 2004.

Rao J., Kungas P., Matskin M., « Composition of Semantic Web Services using Linear Logic Theorem Proving ». *Information Systems Journal - Special Issue on the Semantic Web and Web Services*", Volume 31, Issues 4-5, page 229-296, juin-juillet 2006.

Salaün G., Ferrara A., Chirichiello A., « Negotiation Among Web Services Using LOTOS/CADP », *Lecture Notes in Computer Science*, Volume 3250, Jan 2004, Pages 198 - 212.

van der Aalst W.M.P., Dumas M., ter Hofstede A.H.M., « Web Service Composition Languages: Old Wine in New Bottles? », *euromicro*, *29th Euromicro Conference (EUROMICRO'03)*, p. 298, 2003 (a).

van der Aalst W.M.P., ter Hofstede A.H.M., Kiepuszewski B., Barros A.P., « Workflow Patterns », *Distributed and Parallel Databases*, pages 5-51, 2003 (b).

Yang P., Ramakrishnan C.R., Smolka S. A., « A Logical Encoding of the Pi-Calculus: Model Checking Mobile Processes Using Tabled Resolution », *International Journal on Software Tools for Technology Transfer*, 6(1), pages 38-66, juin 2004, Springer-Verlag.