Contractualisation des ressources pour le déploiement des composants logiciels

Nicolas Le Sommer

Contractualisation des ressources pour le déploiement des composants logiciels

Nicolas Le Sommer

Laboratoire Valoria
Université de Bretagne Sud
Centre de recherches Yves Coppens
Campus de Tohannic
56000 Vannes, France
`Nicolas.Le-Sommer@univ-ubs.fr`

RÉSUMÉ. *Le déploiement de composants logiciels peut s'avérer être une tâche complexe et difficile à réaliser dès lors que les composants logiciels exhibent des propriétés non-fonctionnelles. En effet, de tels composants peuvent ne pas fonctionner de manière convenable si certaines de ces propriétés ne peuvent être satisfaites par leur plate-forme de déploiement. Dans cet article, nous nous proposons de prendre en compte via une approche contractuelle et réflexive une catégorie spécifique des exigences non-fonctionnelles exprimées par les composants logiciels, à savoir celles portant sur les ressources qui sont nécessaires à leur exécution.*

ABSTRACT. *Software deployment can turn into a baffling problem when the components being deployed exhibit non-functional requirements. If the platform on which such components are deployed cannot satisfy their non-functional requirements, then they may in turn fail to perform satisfactorily. In this paper, we present a contract-based approach to take a specific category of non-functional properties specified by components into account, that is those that pertain to the resources that are necessary for their execution.*

MOTS-CLÉS : *Composants logiciels, ressources, contrats, deploiement*

KEYWORDS: *Software components, resources, contracts, deployment*

1. Introduction

De nos jours, les composants logiciels sont devenus des éléments architecturaux incontournables dans le monde du logiciel. Ils doivent permettre d'assimiler à terme le développement du logiciel à un simple processus d'assemblage. Cet objectif ne pourra être atteint qu'à la condition de disposer de composants logiciels conçus comme de véritables unités de déploiement fiables, performantes et parfaitement décrites tant du point de vue de leurs propriétés fonctionnelles que de leurs propriétés non-fonctionnelles. Les modèles de composants actuels (e.g. CORBA Component Model, Entreprise Java Beans, Fractal, COM, OSGi) n'offrent pas aux programmeurs le moyen de spécifier toutes les propriétés non-fonctionnelles de leurs composants. Ainsi, les ressources nécessaires à l'exécution des composants logiciels –qui constituent l'une des principales catégories de propriétés non-fonctionnelles pouvant être exhibées par les composants– sont actuellement ignorées dans ces modèles de composants. Les composants logiciels n'ont pourtant pas tous les mêmes besoins vis-à-vis des ressources. Certains composants peuvent fonctionner avec peu de ressources et avec aucune garantie de disponibilité de ressources (e.g. des composants réalisant des calculs élémentaires), alors que d'autres composants vont exiger de disposer de ressources en quantités importantes et des garanties de disponibilité vis-à-vis de ces ressources afin de pouvoir fournir à leurs utilisateurs un certain niveau de qualité de service (e.g. des composants traitant et affichant des flux vidéos).

Dans cet article, nous nous proposons de prendre en compte les besoins en ressources des composants logiciels avec une approche contractuelle et réflexive. Cette approche vise à permettre aux composants logiciels d'indiquer à leur environnement de déploiement qu'ils n'utiliseront pas plus de ressources que celles dont ils font explicitement la demande, et à leur permettre de bénéficier en retour d'un certain service de la part de cet environnement d'accueil vis-à-vis de la disponibilité des ressources demandées. Cette approche a également pour objectif d'offrir aux composants logiciels le moyen de raisonner sur leurs conditions d'exécution à travers les contrats qu'ils souscrivent avec leur environnement de déploiement, de pouvoir modifier ces conditions d'exécution en renégociant dynamiquement les modalités de leurs contrats, et éventuellement de pouvoir adapter leur comportement en fonction des contrats négociés. Nous montrons également dans cet article comment cette approche a été mise en œuvre au sein d'une plate-forme de déploiement expérimentale baptisée JAMUS.

La suite de cet article s'articule de la manière suivante. Le paragraphe 2 offre une vue d'ensemble de notre approche. Le paragraphe 3 propose une modélisation orientée objet des contrats portant sur les conditions d'accès aux ressources. Le paragraphe 4 présente les mécanismes qui permettent de gérer ces contrats de manière dynamique, ainsi que la mise en œuvre de ces mécanismes au sein de la plate-forme JAMUS. Le paragraphe 5 présente quelques travaux dont la démarche ou la finalité s'apparentent à la nôtre. En guise de conclusion, le paragraphe 6 résume les différents éléments clés de notre approche et énumère quelques unes des perspectives offertes par ce travail.

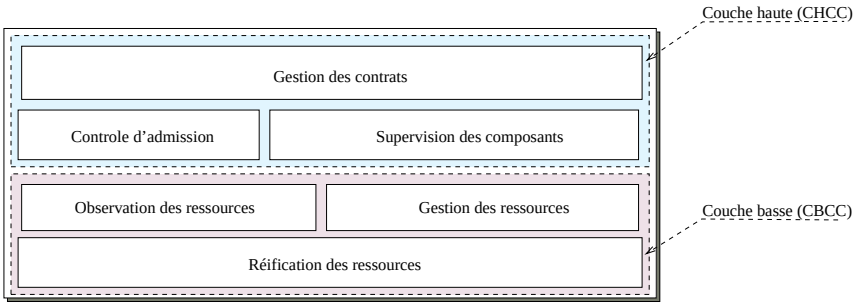

Figure 1 – Fonctionnalités offertes par le cadre de conception

2. Vue d'ensemble de l'approche proposée

Certains composants logiciels présentent des besoins en ressources invariables et parfaitement identifiés, alors que d'autres exhibent des besoins en ressources partiels qui évoluent en cours d'exécution (e.g. composants dont les besoins en ressources sont liés aux choix de l'utilisateur). Il nous paraît donc essentiel que les composants logiciels puissent exprimer leurs besoins en ressources dynamiquement. De la même manière, pour pouvoir tenir compte des besoins en ressources exprimés par les composants logiciels, l'environnement de déploiement des composants doit être doté de mécanismes lui permettant de déterminer dynamiquement s'il est en mesure de satisfaire ces besoins au vu des ressources dont il dispose, et doit être pourvu de mécanismes lui permettant de gérer ces ressources afin d'en assurer la disponibilité le cas échéant.

Dans cette optique, nous avons conçu un cadre de conception qui définit la structure d'un système de contractualisation dynamique des ressources. Ce cadre de conception, dont une vue d'ensemble est présentée dans la figure 1, est organisé en deux couches. Chaque couche peut être complétée et étendue à volonté par les développeurs, leur permettant ainsi de concevoir un système de contractualisation répondant précisément à leurs besoins.

La couche inférieure du cadre de conception définit la structure d'un ensemble de fonctionnalités conçues pour offrir une représentation d'un environnement par réification des ressources de ce dernier, et conçues pour observer et gérer ces ressources à travers cette représentation. Les fonctionnalités de la couche inférieure du cadre de conception ont été définies indépendamment de celles de la couche supérieure. Elles peuvent donc par exemple être utilisées pour concevoir des mécanismes de supervision et de contrôle de machines hôtes ou des composants logiciels adaptatifs.

La couche supérieure du cadre de conception définit quant à elle la structure d'une fonctionnalité de contrôle d'admission, qui permet à un environnement de déploiement de déterminer s'il est en mesure de satisfaire les besoins en ressources exprimés par

les composants logiciels ; la structure d'une fonctionnalité de supervision des composants, qui permet de vérifier que les composants utilisent les ressources conformément à leur contrat ; et la structure d'une fonctionnalité qui offre le moyen de réifier les contrats sous la forme objet, ainsi que de négocier et de vérifier dynamiquement ces contrats.

3. Contrats et avenants

3.1. *Contrats*

Les contrats que nous avons définis permettent d'exprimer les besoins en ressources des composants logiciels de manière qualitative (e.g. droits d'accès, contraintes de disponibilité) et quantitative (e.g. quotas). Ces contrats sont définis comme un ensemble de profils d'utilisation des ressources. Un profil est composé de quatre attributs. Le premier attribut identifie une ressource de manière précise. Le deuxième attribut et le troisième attribut définissent respectivement les droits d'accès et les quotas requis par le composant vis-à-vis de cette ressource. Le quatrième attribut spécifie quant à lui la manière selon laquelle l'environnement de déploiement doit assurer la disponibilité de cette ressource. Grâce à ce dernier attribut, les composants logiciels peuvent préciser que certaines ressources doivent leur être réservées, alors que d'autres ne doivent pas nécessairement l'être. Ils peuvent ainsi indiquer à leur plate-forme de déploiement comment ils sont susceptibles de se comporter en cours d'exécution.

À titre d'exemple, considérons un composant logiciel baptisé *JMailer* qui offre le moyen de rédiger et d'envoyer des courriers électroniques. Ce composant offre à l'utilisateur deux modes de fonctionnements possibles. Dans le premier mode, l'utilisateur doit spécifier sans aucune assistance l'adresse des destinataires, le sujet et le contenu du courrier électronique. Dans le second mode, une assistance lui est fournie pour saisir les adresses électroniques à travers l'utilisation d'un carnet d'adresses. Pour pouvoir fonctionner dans le mode sans assistance, ce composant logiciel exige de pouvoir lire et écrire jusqu'à 500 Kilo-octets de données dans le répertoire *~/.jmailer*, et demande à disposer de 1 Méga-octet de mémoire. Pour s'exécuter dans le second mode, le composant *JMailer* demande à pouvoir lire et écrire jusqu'à 500 Kilo-octets de données dans le répertoire *~/.jmailer* et jusqu'à 1 Méga-octets de données dans le répertoire *~/.jaddrbook*, et exige de disposer de 2 Méga-octet de mémoire. Pour contractualiser ces conditions d'accès aux ressources, notre composant de démonstration *JMailer* pourra par exemple soumettre à son environnement de déploiement les contrats *contract1* et *contract2* de la figure 2.

3.2. *Avenants*

Ainsi que nous l'avons mentionné précédemment, les besoins en ressources des composants logiciels sont susceptibles d'évoluer au cours du temps. Il est donc nécessaire d'offrir aux composants la possibilité de pouvoir renégocier dynamiquement les

```
int Ko=1024 ; int Mo=1024*1024 ;
ResourceUtilisationProfile r1, r2, r3, r4 ;
// Accès en lecture et écriture au répertoire ~/.jmailer
// quotas = 500 Koctets en lecture et écriture
r1 = new ResourceUtilisationProfile(new FilePattern("~/.jmailer"), new FilePermission(FilePermission.ALL),
                                    new FileQuota(500* Ko, 500*Ko), new BestEffort())) ;
// Accès en lecture et écriture au répertoire ~/.jaddressbook
// quotas = 1 Moctets en lecture et écriture
r2 = new ResourceUtilisationProfile(new FilePattern("~/.jaddrbook"), new
FilePermission(FilePermission.ALL),
                                    new FileQuota(1* Mo, 1*Mo), new BestEffort())) ;
// réservation de 1 Moctets de mémoire
r3 = new ResourceUtilisationProfile(new MemoryPattern(), new MemoryPermission(),
                                    new MemoryQuota(1* Mo), new ResourceReservation())) ;
// réservation de 2 Moctets de mémoire
r4 = new ResourceUtilisationProfile(new MemoryPattern(), new MemoryPermission(),
                                    new MemoryQuota(2* Mo), new ResourceReservation())) ;

ResourceOrientedContract contract1= new ResourceOrientedContract ({r1, r2, r4}) ;
ResourceOrientedContract contract2= new ResourceOrientedContract ({r1, r3}) ;
```

Figure 2 – Exemple de contrats pouvant être soumis à la plate-forme JAMUS

modalités du contrat qu'ils ont passées avec leur environnement de déploiement. Le mécanisme d'avenants que nous avons introduit dans notre cadre de conception offre aux composants cette possibilité. En effet, ces avenants sont définis comme un ensemble de clauses qui décrivent les opérations d'ajout, de suppression, de modification de clause d'un contrat. À l'instar des contrats, les avenants sont réifiés sous la forme objet afin de pouvoir être définis et négociés dynamiquement par les composants et leur environnement.

La figure 3 présente un exemple d'avenant qui peut être soumis par le composant *JMailer* à son environnement de déploiement pour renégocier les modalités du contrat qui les lie. Cet avenant a pour effet d'ajouter un profil d'utilisation des ressources portant sur les conditions d'accès au répertoire */tmp* dans le contrat *contract2* présenté dans la figure *2*.

Le paragraphe suivant présente les mécanismes qui permettent de gérer ces contrats et ces avenants de manière dynamique au sein de la plate-forme JAMUS.

```
int Mo=1024*1024 ;
ResourceUtilisationProfile r5 ;
// accès en lecture et en écriture au répertoire /tmp
r5 = new ResourceUtilisationProfile(new FilePattern("/tmp",), new FilePermission(FilePermission.ALL),
                                    new FileQuota(2* Mo, 2*Mo), new BestEffort()) ;
AmendmentClause ac1 = new AmendmentClause(AmendmentClause.ADD, r5) ;
Amendment a1= new Amendment (contract2, {ac1}) ;
```

Figure 3 – Exemple d'un avenant pouvant être soumis à la plate-forme JAMUS

4. La plate-forme JAMUS

JAMUS (*Java Accommodation of Mobile Untrusted Software*) est une plate-forme expérimentale dédiée à l'hébergement de composants mobiles Java capables d'exprimer leurs besoins vis-à-vis des ressources. JAMUS a été conçue par spécialisation de la couche supérieure de notre cadre de conception. Elle s'appuie sur les fonctionnalités d'observation et de contrôle d'accès des ressources offertes par l'environnement d'exécution RAJE (*Resource-Aware Java Environment*), environnement que nous avons conçu par spécialisation de la couche inférieure du cadre de conception. Cet environnement RAJE n'est pas présenté plus amplement dans cet article par manque de place. Le lecteur pourra toutefois se référer à [LES 03] pour avoir une présentation détaillée de RAJE.

La plate-forme JAMUS offre aux composants logiciels le moyen de contractualiser dynamiquement leurs modalités d'accès aux ressources. En outre, étant dédiée à l'hébergement de composants logiciels potentiellement non dignes de confiance, la plate-forme JAMUS soumet chaque composant hébergé à une supervision constante au cours de son exécution et interdit toute utilisation des ressources non conforme aux contrats qu'ils ont souscrits avec elle.

4.1. *Gestion des contrats*

Dans JAMUS, les propositions de contrats soumises par les composants logiciels sont évaluées par un courtier de ressources. Ce courtier implémente une fonctionnalité de contrôle d'admission et un mécanisme de réservation des ressources afin d'offrir aux composants hébergés une certaine qualité de service vis-à-vis de la disponibilité des ressources. La figure 4 montre comment ce courtier est interrogé par le gestionnaire de contrats de la plate-forme JAMUS pour évaluer les contrats. Le processus de contractualisation des ressources mis en œuvre au sein de la plate-forme JAMUS comporte deux phases : une phase de soumission des contrats et une phase de souscription des contrats. Ces deux phases ont été distinguées afin de permettre aux composants

logiciels de pouvoir soumettre à leur environnement de déploiement plusieurs contrats pour évaluation avant de souscrire un contrat particulier avec celui-ci.

4.1.1. *Soumission des contrats*

Lors du démarrage de la plate-forme, le courtier reçoit en paramètre un ensemble de profils d'utilisation des ressources décrivant les ressources disponibles au sein de la plate-forme JAMUS. Au vu de ces informations et de celles qu'il a maintenues à jour en fonction des contrats souscrits avec les composants hébergés par la plate-forme JAMUS, le courtier de ressources décide de la validité des contrats soumis par les composants candidats à l'hébergement. Un contrat est déclaré acceptable par le courtier si toutes les clauses de celui-ci peuvent être satisfaites. Lorsqu'un contrat est déclaré inacceptable par le courtier de ressources, le gestionnaire de contrats de la plate-forme JAMUS invoque une nouvelle fois ce courtier afin de connaître les profils qui ne peuvent pas être satisfaits (invocation de la méthode *getConflictingClause()*). Sur la base des informations retournées par le courtier de ressources, le gestionnaire de contrats construit un rapport de soumission qu'il retourne au composant logiciel. Ces informations visent à aider ce dernier dans la formulation de nouvelles propositions de contrats.

La figure 4 illustre ces propos en présentant un scénario de négociation de contrats entre notre composant de démonstration *JMailer* et la plate-forme JAMUS.

4.1.2. *Souscription des contrats*

Lors de la phase de souscription, les contrats sont évalués une nouvelle fois avant que les ressources ne soient réservées pour le composant par le courtier. Cette réévaluation est nécessaire dans JAMUS car toute ou partie des ressources requises par le composant candidat à l'hébergement peuvent avoir été allouées aux autres composants présents dans la plate-forme JAMUS au moment de la souscription du contrat. Lorsque cette réévaluation est passée avec succès, le courtier réserve les ressources pour le composant logiciel. Cette réservation s'effectue en prélevant sur les quotas des profils d'utilisation des ressources concernés les valeurs de quotas définis dans les profils du contrat.

À titre d'exemple, le contrat *contract2* ayant été déclaré acceptable par le courtier de JAMUS, le composant *JMailer* peut initier une phase de souscription du contrat *contract2* (voir figure 4). Ce contrat est évalué une nouvelle fois par le courtier de ressources afin de déterminer si les besoins requis par le composant peuvent toujours être satisfaits. Dans le cas présent, nous supposerons qu'ils le sont. Le gestionnaire de contrats de JAMUS demande alors au courtier de ressources de réserver les ressources pour le composant *JMailer*.

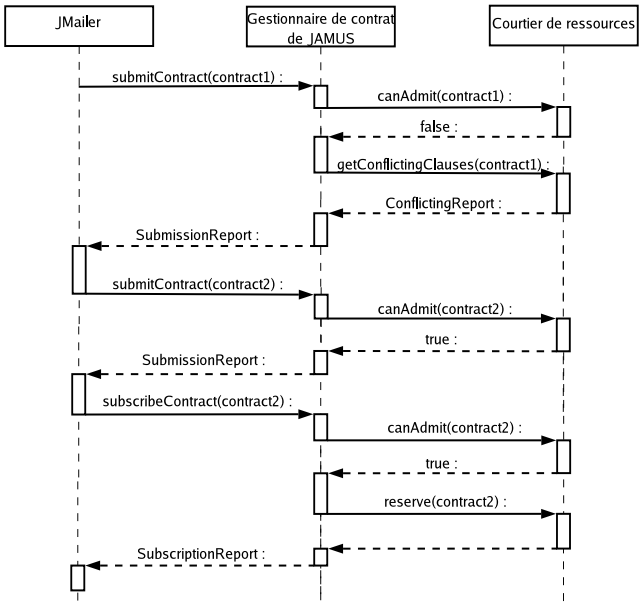

Figure 4 – Exemple de négociation des modalités d'accès aux ressources entre le composant *JMailer* et la plate-forme JAMUS

4.2. *Supervision des composants logiciels*

Chaque composant hébergé par la plate-forme JAMUS s'exécute au sein d'un conteneur. Le conteneur permet d'isoler un composant en lui offrant son propre espace de nommage. Ainsi, les objets créés par les différents composants s'exécutant au sein de la plate-forme JAMUS ne peuvent être partagés et un composant ne peut accéder aux objets manipulés par d'autres composants.

Chaque conteneur intègre un registre de ressources qui est chargé de référencer les ressources créées par le composant logiciel considéré, ainsi qu'un moniteur d'application capable d'extraire des contrats les profils d'utilisation des ressources, et capable d'instancier en conséquence des moniteurs de ressources qui seront chargés de s'assurer que l'utilisation qui est faite des ressources demeure conforme aux modalités d'utilisation définies par le contrat souscrit par le composant. La plate-forme JAMUS intègre un type de moniteurs de ressources spécifique pour chaque type de ressources considéré dans RAJE. Chaque conteneur intègre également un auditeur de registre de ressources (*ResourceTracker*) qui est chargé d'indiquer aux moniteurs de ressources quelles sont les ressources qu'ils doivent superviser.

Afin de présenter le principe de fonctionnement d'un conteneur, considérons le scénario présenté dans la figure 5. Lors de la configuration du conteneur, l'auditeur de

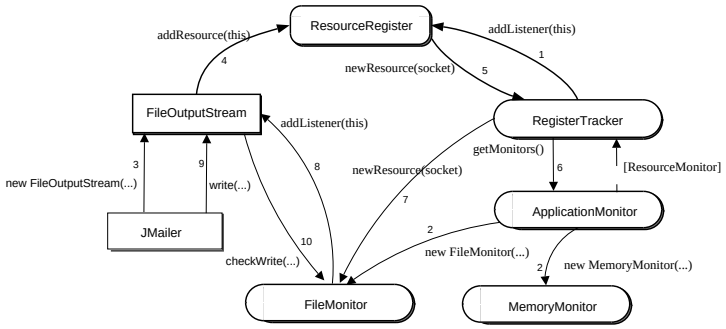

Figure 5 – Principe de fonctionnement du conteneur

registre de ressources s'enregistre en tant qu'auditeur auprès du registre de ressources afin d'être informé de toute nouvelle création ou destruction de ressources dans le conteneur (action 1). Le moniteur d'application instancie quant à lui des moniteurs de ressources en fonction des profils d'utilisation des ressources contenus dans le contrat souscrit par le composant considéré. Dans notre exemple, le moniteur d'application va instancier un moniteur de type *FileMonitor* afin de superviser les accès du composant *JMailer* au répertoire de l'utilisateur, ainsi qu'un moniteur de type *MemoryMonitor* pour vérifier l'utilisation qui est faite de la mémoire (action 2). À l'issue de cette phase de configuration, le conteneur charge le composant *JMailer* et initie son exécution.

Les objets ressources définis dans RAJE sont conçus de manière à s'enregistrer spontanément auprès du registre de ressources lors de leur création. Ainsi, l'objet ressource de type *FileOutputStream* est capable d'informer automatiquement le registre de ressources (action 3) lors de sa création (action 4). Le registre de ressources informe à son tour l'auditeur de registre de ressources de la création de cette nouvelle ressource (action 5). Ce dernier invoque alors le moniteur d'application pour obtenir la liste des moniteurs de ressources instanciés (action 6). Sur la base de cette information, l'auditeur du registre de ressources sélectionne le moniteur devant superviser cette nouvelle ressource (le moniteur *FileMonitor*), et l'informe de l'apparition de celle-ci dans le conteneur (action 7). Le moniteur *FileMonitor* s'enregistre alors en tant qu'auditeur auprès de la ressource *FileOutputStream* afin que cette dernière l'informe de toute action du composant *JMailer* (action 8). Ainsi, lorsque le composant écrit des données sur le flux de données (action 9), celui-ci informe spontanément le moniteur *FileMonitor* de cette écriture (action 10). Grâce à l'ensemble de ces mécanismes, la plate-forme JAMUS est capable de superviser dynamiquement les ressources utilisées par les composants logiciels qu'elle héberge.

Au vu de la complexité des mécanismes de supervision, on peut légitimement penser que le surcoût de la supervision dynamique est important. En fait, les évaluations des performances de la plate-forme JAMUS que nous avons réalisées montrent que ce n'est pas le cas, puisque dans le contexte d'observation le plus défavorable, le

```
Sanction s1, s2 ;
// Sanction différée s'appliquant au répertoire ~/
// l'accès à la ressource sera rejeté au bout de 2 violations successives
s1 = new DifferedSanction(new FilePattern("~/"), Sanction.REJECT, 2) ;
// Sanction immédiate s'appliquant aux connexion TCP sur le port 80
s2 = new ImmediateSanction(new SocketPattern("*", 80), Sanction.LOCK) ;
```

Figure 6 – Exemple de sanctions pouvant être utilisées pour configurer la plate-forme JAMUS

temps d'accès à une ressource se trouve réduit d'environ 2% par moniteur. L'ensemble de ces évaluations peuvent être consultées dans [LES 03].

4.3. *Gestion de la violation des contrats*

La plate-forme JAMUS offre via les moniteurs d'applications et les moniteurs de ressources le moyen de superviser les composants logiciels au cours de leur exécution, ainsi que le moyen de sanctionner les composants ayant violés les modalités de leur contrat. Ces moniteurs s'appuient sur les fonctionnalités de contrôle d'accès offertes par l'environnement RAJE pour sanctionner les composants.

Nous avons identifié deux catégories de sanctions : les sanctions à effet immédiat, et les sanctions à effet différé. Ces dernières sont assujetties à un avertissement préalable qui vise à notifier le composant responsable de la violation de contrat, et à lui permettre ainsi de prendre les mesures nécessaires pour éviter l'application de la sanction. Ces mesures peuvent impliquer par exemple la renégociation des modalités du contrat ou la terminaison du contrat. La figure 6 présente ces deux types de sanctions. La première sanction est une sanction différée s'appliquant au répertoire de l'utilisateur. Elle consiste à rejeter les accès en lecture ou en écriture du composant à ce répertoire au bout de deux violations successives des modalités d'accès relatives à ce répertoire. La seconde sanction est une sanction de type immédiate qui porte sur les connexions TCP établies sur le port 80. Cette sanction consiste à verrouiller l'accès du composant à la socket concernée par la violation.

5. Travaux apparentés

5.1. *Travaux apparentés relatifs à la qualité de service*

La prise en compte des besoins en ressources des composants logiciels et plus généralement des programmes d'application est une problématique connue dans le domaine de la qualité de service. Certains travaux se focalisent uniquement sur l'expression de la qualité de service (e.g. QML [FRO 00]), alors que d'autres proposent

à la fois des solutions pour exprimer la qualité de service et transposer celle-ci en une expression des besoins en ressources (e.g. QuO [SCH 03], QosTalk [WIC 02], 2k [WIC 01]). Cependant, ces travaux proposent uniquement une expression quantitative et globale des besoins en ressources des programmes d'application. Bien qu'étant parfaitement suffisante pour assurer la disponibilité des ressources pour les applications, cette expression des besoins en ressources n'est pas assez précise pour pouvoir être également utilisée au sein des mécanismes de sécurité. Pourtant à l'instar de la plate-forme JAMUS, il est possible de confronter les besoins exprimés par les composants à la politique de sécurité mise en œuvre au sein de la plate-forme de déploiement afin de déterminer si le composant peut être déployé, et utiliser les besoins ainsi exprimés pour superviser les composants en cours d'exécution.

5.2. *Travaux apparentés relatifs à la sécurité*

L'environnement d'exécution de Java (JRE *: Java Runtime Environment*) met en œuvre un modèle de sécurité connu sous le nom de *SandBox*. Depuis la plate-forme Java 2, ce modèle de sécurité repose sur la notion de domaine de protection [GON 97]. Un domaine de protection constitue un environnement d'exécution dont la politique de sécurité peut être spécifiée sous la forme de permissions. Le modèle de sécurité mis en œuvre dans le JRE repose sur des mécanismes « sans état ». Il n'est donc pas possible d'imposer des contraintes quantitatives sur les ressources manipulées au sein d'un domaine de protection. En conséquence, les dysfonctionnements résultant de l'utilisation abusive des ressources ne peuvent pas être prévenus.

Des environnements tels que JRes [CZA 98] et KaffeOS [BAC 00] apportent un début de réponse au problème du contrôle quantitatif des ressources posé par les environnements Java traditionnels. Ils offrent en effet des mécanismes permettant de comptabiliser – et, éventuellement, de limiter – l'utilisation de chaque type de ressources par une entité active (un *thread* dans le cas de JRes et un processus dans le cas de KaffeOS). Cependant dans ces environnements, les ressources sur lesquelles portent la comptabilisation et le contrôle sont des ressources globales. On peut ainsi comptabiliser les accès au réseau réalisés par un *thread* (ou processus), mais on ne peut pas distinguer les transmissions réalisées vers une machine donnée, ou vers un numéro de port précis.

Les projets Naccio [EVA 00] et Ariel [PAN 99] proposent chacun un langage et des mécanismes permettant de définir de manière très précise la politique de sécurité qui doit être appliquée à un programme d'application lors de son exécution. L'application d'une politique de sécurité est réalisée statiquement, par réécriture du *bytecode* du programme d'application, mais aussi des classes de l'API de la plate-forme Java. Cette approche se prête donc bien à la génération anticipée d'un ensemble d'API pré-définies garantissant chacune le respect d'une politique de sécurité générique. En revanche, elle ne permettrait pas, comme le fait la plate-forme JAMUS, d'assurer la supervision d'un programme d'application en fonction d'une politique de sécurité définie à partir des besoins exprimés par ce même programme lors de son démarrage.

6. Conclusion

Dans cet article nous avons présenté un cadre de conception pour la construction d'un système de contractualisation dynamique des ressources, ainsi qu'une plate-forme d'accueil dédiée à l'hébergement de composants logiciels qui a été mise en œuvre à l'aide de ce cadre de conception. Cette plate-forme nous a permis de mettre en évidence la pertinence et la validité de notre approche. Nous étudions actuellement la possibilité d'intégrer notre approche au sein de modèles de composants logiciels existants (e.g. Fractal, OSGi) afin d'en améliorer les mécanismes de déploiement.

7. Bibliographie

[BAC 00] BACK G., HSIEH W. C., LEPREAU J., « Processes in KaffeOS : Isolation, Resource Management, and Sharing in Java », *4th Symposium on Operating Systems Design and Implementation*, octobre 2000.

[CZA 98] CZAJKOWSKI G., VON EICKEN T., « JRes : a Resource Accounting Interface for Java », *ACM OOPSLA Conference*, 1998.

[EVA 00] EVANS D., « Policy-Directed Code Safety », PhD thesis, Massachussets Institute of Technology, février 2000.

[FRO 00] FROLUND S., KOISTINEN J., « Quality of Service Specification in Distributed Object System Design », *4th Usenix Conference on Object-Oriented Technologies and Systems (COOTS)*, 2000.

[GON 97] GONG L., « Java Security : Present and Near Future », *IEEE Micro*, vol. -, 1997, p. 14-19.

[LES 03] LE SOMMER N., « Contractualisation des ressources pour les composants logiciels : une approche réflexive », Mémoire de thèse, Université de Bretagne Sud, décembre 2003.

[PAN 99] PANDEY R., HASHII B., « Providing Fine-Grained Access Control for Java Programs », *The 13th Conference on Object-Oriented Programming, ECOOP'99*, Springer-Verlag, juin 1999.

[SCH 03] SCHANTZ R. E., J.P. L., C. R., D.C. S., KRISHNAMURTHY Y., « Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware », *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brézil, juin 2003.

[WIC 01] WICHADAKUL D., NAHRSTEDT K., GU X., D. X., « 2KQ+ : An Integrated Approach of QoS Compilation and Component-Based, Run-Time Middleware for the Unified QoS Management Framework », *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, novembre 2001.

[WIC 02] WICHADAKUL D., NAHRSTEDT K., « A Translation System for Enabling Flexible and Efficient Deplyoment of QoS-aware Applications in Ubiquitous Environments », *First International IFIP/ACM Working Conference on Component Deployment (CD 2002)*, Berlin, Germany, juin 2002.