UNIVERSITÉ DE BOURGOGNE,
Dijon,

Modèle d'abstraction des données de contexte dans la configuration d'un reseau d'applications

MÉMOIRE

MASTER RECHERCHE IMAGE INFORMATIQUE ET INGÉNIERIE

par

Ghislain Loaec

Tuteurs:
Nader Mbarek
Emmanuel Garette

2014

Table des matières

Table des figures

Liste des tableaux

LISTE DES ALGORITHMES

Page

REMERCIEMENTS

Je souaiterais remercier...

RÉSUMÉ

Modèle d'abstraction des données de contexte dans la configuration d'un reseau d'applications

Par

Ghislain Loaec

Master Recherche Image Informatique et Ingénierie in Informatique

Unversité de Bourgogne, Dijon, 2014

Nader Mbarek

L'objectif fondamental de l'informatique ubiquitaire est de faciliter l'utilisation de l'ordinateur. Cela passe par extraire le maximum de bénéfices de l'environement numérique. Les défaillances logicielles deviennent monnaie courante à mesure que les systèmes informatiques et leur complexité continuent de croître. Le problème réside principalement dans l'absence de standards ou de modèles réutilisables pour la gestion des informations de contexte.

Chapitre 1

État de l'art

1.1 Introduction

La configuration des composantes logicielles impose un coût majeur dans l'administration d'un système. Des erreurs de configuration peuvent se traduire par des vulnérabilités en termes de sécurité, de sévères perturbations dans le fonctionnement de la brique logicielle, ou purement et simplement provoquer un déni de service. La prise en considération du contexte pourrait permettre une abstraction partielle ou complète de cette couche très technique et extrêmement pénible à configurer.

Un système sensible au contexte doit être capable de mimer la capacité humaine à reconnaître et exploiter l'information implicitement présente dans l'environnement. Cela implique une configuration dynamique de chacune des composantes de l'architecture, de manière à pouvoir ajuster leur comportement respectif en fonction de la situation. Identifier l'activité humaine est un défi certain, il est essentiel que les applications opèrent en transmettant l'information appropriée au bon endroit et au bon moment par inférence de l'intention des utilisateurs. L'informatique sensible au contexte est un paradigme dans lequel les application peuvent découvrir et tirer profit d' informations de circonstance telles que la position actuelle, l'heure de la journée, les personnes et périphériques dans l'environnement et leurs activités.

Dans ce mémoire, nous aborderons dans un premier temps les principes communs à chacune des solutions existantes. Ces concepts seront résumés à l'aide d'un framework conceptuel dérivé par couches. Nous présenterons un certaine variété d'infrastructures et d'intergiciels reconnus pour faciliter la configuration d'applications et de services basés sur le contexte. Dans un second temps, nous proposerons un modèle d'abstraction des informations de contexte basé sur les ontologies. Ce modèle met en corrélation des concepts tels que le consensus de raft et la théorie de la promesse pour aboutir à un système distribué permettant la coordination d'informations de contexte. Enfin, nous apporterons une preuve de concept par une implémentation du modèle en langage Python.

1.2 Contexte

De nombreux débats ont eu lieu et continuent de voir le jour sur le sens du contexte et de l'informatique sensible au contexte. Alors que la plupart des gens comprennent de manière tacite ce qu'est le contexte, ils le trouve par ailleurs particulièrement difficile à élucider.

> "... toute information pouvant être utilisée pour caractériser la situation d'une entité.
> Une entité peut être une personne, un lieu ou un objet considéré pertinent dans l'interaction
> entre un utilisateur et une application, notamment l'utilisateur et l'application eux même." [1]
> *Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike*
> *Pinkerton. Cyberguide : a mobile context-aware tour guide. Wireless Networks, 3(5) :421–433, Octobre 1997.*

FIGURE 1.1: Le contexte défini par Abowd et. al. (1997)

Cette définition rend la tâche plus facile à un développeur d'application pour énumérer le contexte dans un scénario d'application donné. Si un fragment d'information peut être utilisé pour caractériser la situation d'un participant dans une quelconque interaction, alors cette information appartient au contexte.

1.3 Vue d'ensemble sur le contexte

> "Le contexte est efficace, seulement lorsqu'il est partagé." [21]
> *Terry Winograd. Architectures for Context.*
> *Human-Computer Interaction, 16(2) :401–419, Décembre 2001.*

FIGURE 1.2: Winograd à propos du contexte (2001)

Pour s'assurer que le contexte soit partagé, il doit d'abord être recueilli et rigoureusement traité. Cela implique qu'un système sensible au contexte doit être en mesure de comprendre ce qu'est le contexte avant d'aller à la recherche de ces informations et de pouvoir les catégoriser.

1.3.1 Classes de contexte

Schilit et al. proposent la classification suivante des informations de contexte :

- **Contexte informatique** - Connectivité réseau, bande passante, et ressources à proximité telles que des imprimantes, des affichages ou des postes de travail.
- **Contexte utilisateur** - Le profil utilisateur, sa situation géographique, sa situation sociale actuelle et les individus qui l'entourent.
- **Contexte physique** - L'éclairage, le niveau de bruit, les condition de circulation ou la température.

Chacune de ces catégories contiennent une richesse d'informations pertinentes pour le système sensible au contexte. Elle ne peuvent cependant pas être traitées de manière isolée pour pouvoir en extraire le meilleur. L'intention du système sensible au contexte est de rassembler et de fusionner ces informations pour aboutir à une vue d'ensemble de la situation. Une fois le contexte mis en tampon ou en base, le système doit alors filtrer les informations pertinentes pour l'utilisateur, dans le moment présent.

Les informations de contexte peuvent alternativement être subdivisées en 2 catégories bien distinctes : contexte virtuel ou physique.

1.3.1.1 Contexte virtuel

Le contexte virtuel inclut la version du système d'exploitation, les possibilités d'interface, la technologie en charge de l'accomplissement des communications, les emails envoyés et reçus, et les documents édités. Autrement dit, il s'agit là de tous les faits non tangibles. Pierre Lévy donne une définition intéressante du contexte virtuel à travers une question assez philosophique :

> "Pourquoi la consommation d'une information n'est-elle pas destructive
> et sa détention n'est-elle pas exclusive ?
> Parce que l'information est virtuelle" [15]
> *Pierre Lévy. Qu'est ce que le virtuel ?*
> *La découverte, 56/159, Juillet 2010.*

FIGURE 1.3: Le contexte virtuel par Pierre Lévy (2010)

1.3.1.2 Contexte physique

Les contexte physique d'un autre coté peut se traduire par la présence d'une autre entité, qu'elle soit utilisateur ou périphérique, la proximité d'une imprimante en particulier, une indication que l'utilisateur est debout, en train de marcher ou assis ou même les conditions météorologiques actuelles. En d'autres termes, le contexte physique peut être défini comme toute donnée acquérable par le biais d'un capteur.

1.3.1.3 Contexte historique

Les contextes mémorisés au cours d'un certain laps de temps. Cette information est considérée très utile, mais n'est que très rarement utilisée, sauf peut-être par les applications mobiles. Le système doit être en mesure d'estimer les informations valant la peine d'être conservées. Cette évaluation est excessivement coûteuse et nécessite donc des algorithmes très performants.

1.3.2 Les caractéristiques des informations de contexte

Les chercheurs l'université de Queensland ont classifié quatre caractéristiques majeures : [5]

1. **Les informations de contexte présentent une gamme de caractéristiques temporelles**
   L'information de contexte est d'ores et déjà catégorisée selon l'environnement auquel il appartient : virtuel ou physique ; elle peut en outre est subdivisée selon un critère de temporalité :
   – Information statique : toute information apparentée à l'environnement de l'utilisateur qui ne varie pas.
   – Information dynamique : l'information accumulée continuellement, fréquemment et automatiquement.
   De plus, l'information de contexte appartenant au passé semble indispensable pour la compréhension de l'état global de l'environnement.
2. **L'information de contexte n'est pas parfaite**
   Cela considère la validité du contexte, majoritairement concernant les informations de contexte dynamique. La vitesse et la fréquence à laquelle l'information varie soulève de sérieuses raisons de douter de sa solidité. Ce "délai entre la production et l'utilisation de l'information de contexte" [5] est une préoccupation non négligeable. D'autres sources d'inquiétudes quant au bien-fondé de l'information de contexte incluent la fiabilité des informations fournies par les producteurs de contexte : défaillance d'un capteur, chemin rompu entre les producteurs ou n'importe quelle source qui fournirait une information erronée ou désuète.
3. **L'information incarne un grand nombre de représentations**
   Les données brutes recueillies depuis l'environnement physique et virtuel peuvent prendre de nombreuses formes et doivent être traitées pour se mêler à d'autres informations de contexte. La probabilité pour qu'un système sensible au contexte obtienne une satisfaction totale lors d'une "capture des relations existantes entre les représentations alternatives" [5] de l'information et celle apte à la situation courante est quasi nulle.
4. **Les informations de contexte sont très fortement corrélées**
   L'information contextuelle provenant d'une origine particulière peut avoir un lien très étroit avec sa source, si bien qu'elle en est dépendante. Le contexte peut ne pas être fiable "là où les caractéristiques de l'information dérivée sont intimement liées aux propriétés de l'information dont il est issu." [5]

1.3.3 Système sensible au contexte

Un système est dit sensible au contexte s'il utilise le contexte pour fournir des informations pertinentes et/ou des services à l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur. Les systèmes sensibles au contexte peuvent être implémentés sous plusieurs formes, mais pour accomplir cet objectif, le système doit d'un manière générale :

1. Recueillir l'information
2. Sérialiser cette information
3. Fusionner l'information pour générer un contexte de plus haut niveau

4. Prendre automatiquement des mesures basées sur l'information recueillie
5. Rendre l'information disponible, dans l'immédiat, dans le futur ou au moment approprié pour améliorer et aider à la complétion de la tâche de l'utilisateur.

Les chercheurs du Context Toolkit à l'université de Berkley proposent 3 fonctionnalités qu'une application sensible au contexte doit impérativement supporter : [8]

1. Présentation de l'information et des services à l'utilisateur
2. Exécution automatique d'un service pour l'utilisateur
3. Journalisation de l'information de contexte pour une extraction ultérieure

Les développeurs du système Kimura ont conçu des composants distribués destinés à un système sensible au contexte global. Ces composants se divisent en trois classes [20], qui sont les suivantes :

– **Acquisition du contexte** - Le système récupère l'information de contexte et l'ajoute dans un dépôt dédié.
– **Interprétation du contexte** - Les système convertit l'information recueillie en un contexte de travail.
– **Interaction de l'utilisateur** - Le système affiche le contexte de travail à l'utilisateur.

1.3.4 Recueillir l'information

Certaines informations de contexte sont explicitement données au système, comme le nom de l'utilisateur, son âge, son adresse email, les variables d'environnement ou le registre. D'autres informations physiques primitives comme la lumière, la température ou des lectures de pression peuvent être acquises par l'intermédiaire d'un capteur.

La situation géographique et l'identité sont les deux fragments de contexte les plus fréquemment détectés.

> "Les capteurs ne sont pas toujours précis ou fiables à 100%,
> tout particulièrement s'ils sont à usage unique. Le processus de
> récupération de l'information doit être tolérant à la défaillance
> potentielle d'un capteur. Toutes les informations recueillies doivent
> être assujetties à des contrôles de validité pour vérifier leur exactitude.
> La fusion des capteur est un moyen de remédier à cette difficulté." [17]
> *Albrecht Schmidt, Michael Beigl, and Hans-W Gellersen.*
> *There is more to context than location. Computers & Graphics, 23(6) :893–901, 1999.*

FIGURE 1.4: La fiabilité des capteurs par Schmidt (1999)

Prenez l'exemple d'une centrale électrique, possédant de nombreuses sondes de température implantées à diverses endroits d'un réacteur nucléaire. Le système doit prendre la décision de refroidir plus ou moins généreusement

en fonction de ces informations sondées. La sous-estimation du contexte de température aurait des conséquences catastrophiques. Il semblerait judicieux de simplement calculer un valeur moyenne et/ou d'écarter les valeurs reportées qui diffèreraient manifestement des autres mesures. Cette méthode éviterait des prises de mesures drastiques du système sensible au contexte dans le but de corriger ce qu'il considère comme une variation de température, mais qui en réalité n'est que le résultat d'une panne de capteur.

1.3.5 Récupérer l'information

- **Modèle "Push"** - La source de contexte récupère l'information avant même qu'elle soit requise. Cela améliore indéniablement les performances, mais nécessite une consommation des ressources fréquente et considérable pour une information qui pourrait bien ne jamais être exploitée.
- **Modèle "Pull"** - Collecte l'information de contexte au moment opportun. Cela autorise le recueil d'information à la demande, mais expose le système aux latences réseaux et aux indisponibilités potentielles de certains services.

La méthode de d'acquisition des données de contexte est très importante lorsque l'on conçoit un système de cette nature, puisque c'est ce qui prédéfinit le style architectural du système. Chen (2003) [6] présente trois approches différentes pour acquérir l'information de contexte.

1.3.5.1 Capteur en accès direct

Cette approche est souvent utilisée dans les périphériques avec capteurs intégrés. Le logiciel client récupère l'information désirée directement depuis les sondes, i.e., il n'y a pas de couche supplémentaire pour obtenir et traiter les données du capteur. Les pilotes pour les capteurs sont raccordés à l'application, donc cette méthode de couplage étroit n'est utilisable que dans des cas très rares. Par conséquent, elle n'est pas adaptée pour les systèmes distribués.

1.3.5.2 Infrastructure intergicielle

L'approche intergicielle introduit un architecture en couches avec l'intention d'abstraire les détails de bas niveau du processus de détection. La méthode est équivalente au modèle client/serveur, plus flexible que le widget puisqu'elle favorise l'indépendance de chacune des composantes du système. Chaque composante doit être en mesure d'effectuer les opération suivantes : établir des connexions, envoyer et recevoir des messages et gérer les erreurs. Cet modèle est plus complexe de manière significative, mais l'approche est plutôt aisée puisqu'elle supporte un grand nombre de périphériques et d'applications par l'utilisation de normes de codage et des protocoles réseaux standards.

1.3.5.3 Serveur de contexte

Cette approche distribuée élargie l'infrastructure basée sur les intergiciels en introduisant un gestionnaire d'accès distant. Les données recueillies par les capteur sont déplacées vers ce dit "serveur de contexte" dans le but de faciliter les accès concurrents.

1.3.6 Les architectures de gestion de contexte

Winograde (2001) [21] décrit trois modèles différents afin de coordonner les processus et composantes multiples :

- **Widgets** L'objectif clé du widget est distinguer l'application du processus d'acquisition de contexte, de manière à abstraire la complexité que représente le recueil et la gestion de l'information de contexte. Le widget est considéré comme un médiateur qui transmet exclusivement des informations pertinentes à l'application. Un widget de contexte fonctionne totalement indépendamment de l'application, cet qui permet à plusieurs applications d'en faire usage simultanément. Le widget est notamment responsable de l'entretien d'un historique complet du contexte sondé au fil du temps. C'est le modèle le plus répandu.
- **Services réseaux** Cette approche plus flexible, comme l'argumente Hong and Landay (2001) [12], ressemble à l'architecture de serveur de contexte. Au lieu d'un gestionnaire de widget centralisé, des techniques sont utilisées pour découvrir les services réseaux dans l'infrastructure. Cette approche basée sur les services n'est aussi performante que l'architecture basée sur les widgets à cause de la complexité des composants orientés réseaux, mais fournit une certaine robustesse.
- **Tableau noir (Blackboard)** En contraste avec la vue orientée processus du widget et le modèle orienté services, le tableau noir représente une approche orientée sur les données. Cette approche donne une attention toute particulière aux données, en faisant correspondre des motifs afin d'extraire l'information recherchée. Dans ce modèle asymétrique, les processus envoient de messages dans un média partagé, le dit "tableau noir", et souscrivent à recevoir des notifications lorsque certains évènements se produisent. Les avantages de ce modèle résident dans la simplicité de configuration et d'ajout de nouvelles sources de contexte.

1.3.6.1 Critères d'arbitrage

Afin de définir au mieux quel modèle choisir dans la mise en place d'un système sensible au contexte, il est avisé de considérer les caractéristiques suivantes :

- **Efficacité** : accélérer le débit de l'information compte tenu de la bande passante et de la latence causée par l'explosion du nombre d'applications et de périphériques dans le réseau.
- **Effort de configuration** : compte tenu de la quantité variable de composants, effectuer des changements dans l'état de la configuration, sans engendrer de perturbations ou mettre le système en échec, n'est pas une tâche fastidieuse. Le modèle doit s'assurer que l'édition est sans danger.
- **Robustesse** : Le degré auquel le système peut faire face à la défaillance.
- **Simplicité** :

un système qui nécessite une compréhension avancée des mécanismes qu'il implémente pour faire l'usage, ne sera utilisé que par ceux qui auront le dévouement et la motivation de le maîtriser [21].

– **Extensibilité** :

Les services supportant la notion générale de contexte doivent être facilement extensible pour accueillir toute nouvelle source d'information de contexte non anticipée [10].

Ces critères peuvent être utilisés pour comparer et contraster les différents modèles de gestion de contexte présentés précédemment.

> Le modèle widget a lien très étroit avec les composantes système, ce qui en fait le modèle de plus efficace dans certaines circonstances. Il souffre malheureusement d'une configuration très complexe et est impuissant face à l'échec. Le modèle d'infrastructure, constitué de composants indépendants, est très délicat à appréhender, mais cela ne semble pas être un facteur de résignation pour de nombreux développeurs. Toutefois, la configuration est simple et le système est robuste, ce qui compense le critère négatif de difficulté de compréhension. Pour finir, le modèle tableau noir avec des composants très faiblement liés présente des problèmes d'efficacité, mais il reste simple, robuste et très facilement configurable. [21]
>
> *Terry Winograd. Architectures for Context.*
>
> *Human-Computer Interaction, 16(2) :401–419, Décembre 2001.*

FIGURE 1.5: Comparaison des différents modèles de gestion de contexte par Winograd (2001)

1.3.7 Représentation du contexte

Il existe plusieurs manières de modéliser l'information de contexte :

– **Clé/valeur** : La structure de données la plus simple pour modéliser le contexte. Elle est très fréquemment utilisées dans les framework de services, où les paires clé/valeur servent à décrire les aptitudes d'un service.
– **Langage de balisage** : Structure de données constituée de balises avec des attributs et des contenus associés.
– **Graphique** : Langage de Modélisation Unifié (UML), extension de la Modélisation Role Objet (ORM) par le contexte.
– **Orienté objet** : Exploite tout la puissance de modélisation objet : l'encapsulation, la réutilisabilité, l'héritage. Les approches existantes utilisent des objets variés pour représenter différents types de contextes, et encapsule les détails du traitement et de la représentation du contexte.
– **Logique** : Haut degré de formalité. Typiquement, des faits, des expressions et des règles sont utilisés pour définir le modèle de contexte.
– **Basé sur les ontologies** : Représente une description des concepts et des relations. Cet outil est très prometteur pour modéliser l'information contextuelle, grâce à son expressivité élevée et très formelle et les

possibilités d'appliquer des techniques de raisonnement propres aux ontologies.

La conclusion de l'évaluation présentée par Strang et Linnhoff-Popien [19], basé sur six critères d'exigences, montre que les ontologies incarnent de loin le modèle de plus expressif et comblent la plupart de ces exigences.

Korpipää et al. [14] présente un ensemble de conditions requises et d'objectifs dans la conception d'une ontologie de contexte :

- **Simplicité** : Les expressions et relations définies doivent être les plus simples possible pour ne pas décourager les développeurs d'applications.
- **Flexibilité et extensibilité** : L'ontologie doit supporter l'ajout de nouveaux éléments de contextes et de relations.
- **Généricité** : L'ontologie ne doit pas être limitée à un seul type d'atome de contexte, mais au contraire supporter un maximum de formats pour l'information de contexte.
- **Expressivité** : L'ontologie doit permettre de décrire le plus d'états de contexte possibles et de manière la plus détaillée qu'il soit.

Un atome de contexte peut être décrits à l'aide de quelques attributs seulement. Les deux attributs les plus évidents sont :

- **Type du contexte** : Catégorie dans laquelle s'inscrit le contexte, qu'il s'agisse d'une donnée de température, temporelle, de vitesse ou d'accélération, etc. L'information de genre peut être utilisée en tant que paramètre dans une requête ou une souscription pour certains types de contexte.
- **Valeur du contexte** : Les données brutes recueillies par une sonde. L'unité dépend très étroitement du type de contexte et du capteur dont l'information provient, e.g., degré Celcius, kilomètres par heure, mégabit, etc.

Le type et la valeur du contexte ne sont néanmoins pas des informations suffisantes obtenir un système sensible au contexte opérationnel. En effet, d'autres attributs supplémentaires doivent être mis en œuvre dans un atome de contexte pour améliorer sa concision :

- **Horodatage** : Valeur de type date/heure représentative de l'instant à laquelle l'information de contexte a été capturée. Elle est requise dans l'élaboration d'un historique de contexte ou même dans le traitement des conflits.
- **Source** : Comment l'information a-t-elle été recueillie ? Dans le cas d'un capteur matériel, l'identifiant de la sonde doit être conservé pour permettre à une application de préférer les informations provenant d'un capteur ou d'un autre.
- **Confiance** : L'incertitude confiée à l'information sondée. Toutes les sources de données ne fournissent pas une information juste, les données de situation géographique souffrent d'une imprécision certaine intrinsèquement liée à la puce GPS utilisée.

1.3.8 Interprétation du contexte

> L'interprétation fait référence au processus d'élévation
> du niveau d'abstraction d'un fragment de contexte [9]
> *Anind Dey, Gregory Abowd, and Daniel Salber.*
> *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications.*
> *Human-Computer Interaction, 16(2) :97–166, Décembre 2001.*

FIGURE 1.6: Interprétation du contexte par Dey (2001)

Après avoir détecté, récupéré et sauvegardé le contexte provenant de sources variées, le système sensible au contexte devra implémenter des mécanismes d'interprétation du contexte, de sorte à ce que les informations recueillies soit utilisées de manière convenable. Cette étape d'interprétation implique l'intégration des plusieurs contextes en un seul et unique contexte de plus haut niveau. Ainsi, l'interprète modifie les informations de contexte en augmentant son niveau d'abstraction. Une approche est d'utiliser la fusion de contexte pour convertir des informations de bas niveau en un contexte global directement à la portée des applications.

1.3.9 Framework conceptuel en couches

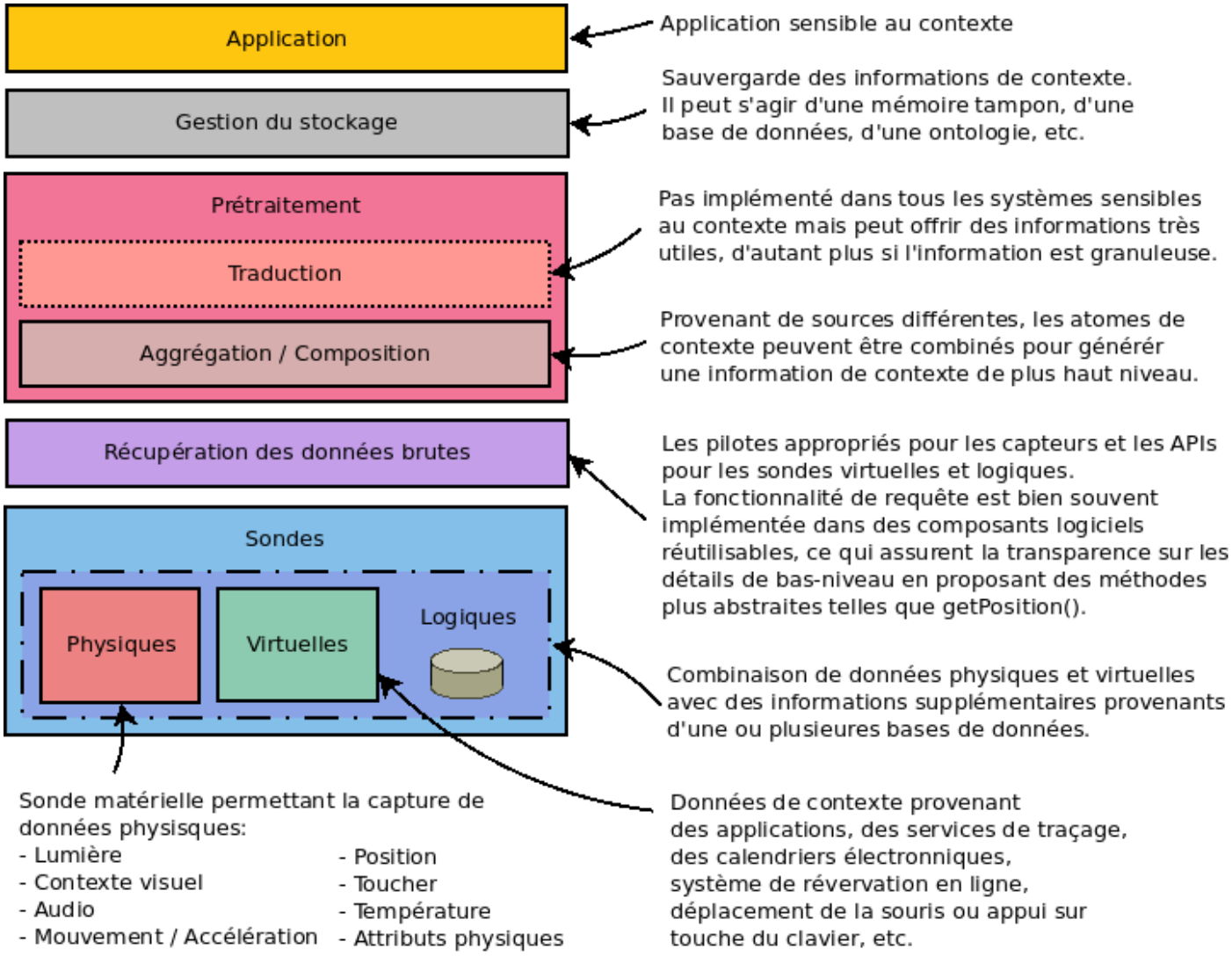

FIGURE 1.7: Framework conceptuel d'un système sensible au contexte

1.3.10 Sécurité et confidentialité

> La confidentialité est intrinsèquement liée au contrôle. [3]
>
> *Mark Ackerman, Trevor Darrell, and Daniel Weitzner. Privacy in Context. Human-Computer Interaction, 16(2) :167–176, Décembre 2001.*

FIGURE 1.8: La confidentialité par Ackerman (2001)

Comme le contexte est susceptible de contenir des informations sensible sur des personnes, leur situation géographique ou leur activité par exemple, il est nécessaire de protéger le caractère personnel de ces données. Les systèmes sensibles au contexte soulèvent des défis en termes de confidentialité inhabituels et inconsidérés

par les systèmes traditionnels. La question quant au contrôle des différents capteurs et services n'est pas abordée dans Dey et al. [9], et bien souvent, des applications telles que le Context Toolkit sont supposées être sous le contrôle d'un "utilisateur". Les utilisateurs sont assignés comme propriétaires des données sondées par leurs soins. Ils sont par ailleurs autorisés à octroyer l'accès à ces données à d'autres utilisateurs.

Néanmoins, en ce qui concernent les sondes capables de déterminer la présence d'individus dans une pièce ou à tout autre endroit, il n'existe aucunes préconisations à ce sujet. C'est un point toutefois critique.

D'une manière générale, dans une environnement mettant à disposition un certain éventail d'applications sensibles au contexte, un individu opère à travers de multiples environnements sociaux, qui eux même font l'usage des données d'un grand nombre d'autres individus.

Aspects souvent négligés, la sécurité et la confidentialité font partie des composantes les plus importantes d'un système sensible au contexte, car la protection des données sensibles doit être garantie.

1.4 Systèmes et frameworks existants

Le tableau 1.1 met en concurrence les principaux aspects des approches existantes discutées précédemment. Le style architectural d'un système sensible au contexte dépend en grande majorité de la méthode d'acquisition des informations de contexte. Le principal critère permettant de qualifier une approche architecturale comme raisonnable est la démarquation claire entre l'acquisition de contexte et les composantes de l'utilisateur proposée par Dey (2000) [8]. Tous les frameworks présentés dans ce mémoire supportent cette scission des préoccupations.

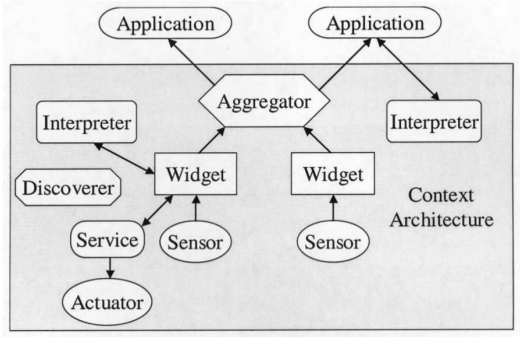

FIGURE 1.9: Exemple de configuration des composants du Context Toolkit

1.4.1 Technologies de détection

Dans chaque framework, la technologie de détection est implémentée différemment. Afin de garantir la séparation détection/composants susmentionnée, il est préférable d'encapsuler le mécanisme concret de détection dans des composantes séparées. De plus, cela permet un accès aux données de contexte par le biais d'interfaces définies. Il n'existe aujourd'hui aucun langage de description ou d'ontologie permettant la réutilisation de l'information de contexte à travers les différents intergiciels et frameworks. Par conséquent, des solutions propriétaires ont vu le jour et sont utilisées par de nombreux frameworks. Pour la détection d'information de contexte, SOCAM implémente l'approche la plus sophistiquée : des services web fournissent de APIs (interfaces SOAP) pour les sondes virtuelles externes et les sondes internes sont interrogées par l'utilisation d'évènements contextuels. Ces évènements de contexte sont modélisés en OWL et basés sur une ontologie prédéfinie.

Architecture	Detection	Modèle de contexte	Traitement du contexte	Découverte de ressources	Données de contexte historiques	Sécurité et confidentialité	
CASS	Intergiciel centralisé	Noeuds de capteurs	Modèle de données relationnel	Moteur d'inférence et base de connaissance	n.a.	Disponible	n.a.
Cobra	Basée sur les agents	Modèle d'acquisition de contexte	Ontologies (OWL)	Moteur d'inférence et base de connaissance	n.a.	Disponible	Politiques en langage Rei
Context Management Framework	Basée sur le "tableau noir" (blackboard)	Serveurs de ressources	Ontologies (RDF)	Service de reconnaissance de contexte	Serveurs de ressources et mécanisme de souscription	n.a	n.a.
Context toolkit	Basée sur les widgets	Widgets de contexte	Tuples attribut/valeur	Interprétation et agrégation du contexte	Composant de découverte	Disponible	Propriété de contexte
CORTEX	Modèle d'objet sensible	Framework de composants de contexte	Modèle de données relationnel	Framework de découverte de services	Framework de gestion des composantes ressources	Disponible	n.a.
Gaia	MVC (étendu)	Fournisseurs de contexte	Prédicats quaternaires (DAML + OIL)	Module de contexte de service (logique de premier ordre)	Service de découverte	Disponible	Supporté (e.g., traçage sécurisé, confidentialité de localisation, contrôle d'accès)
Hydrogen	Architecture trois couches	Adaptateurs variés pour les types de contexte	Orienté objet	Interprétations et agrégation des données brutes seulement	n.a.	n.a.	n.a.
SOCAM	Distribué avec serveur central	Fournisseur de contexte	Ontologies (OWL)	Raisonneur contextuel	Service de localisation de service	Disponible	n.a.

TABLE 1.1: Comparaison des framework existants pour la gestion de contexte

1.4.2 Représentations du contexte

Afin d'obtenir des applications et services sensibles au contexte intelligents et adaptables, le modèle et la logique de traitement du contexte pris en charge individuellement par chaque framework est un critère majeur. Les ontologies offrent un formalisme très riche pour représenter l'information de contexte. Basé sur un modèle ontologique très sophistiqué, les raisonneurs peuvent dériver de nouveaux concepts et ajuster le comportement d'un service en conséquence. La représentation clé/valeur utilisée dans le Context Toolkit devient donc un inconvénient majeur. Le fait que de tels attributs ne peuvent contenir aucune sémantique restreint de manière considérable de le développement de procédés de traitement et d'agrégation de contexte. De plus, les modèles non basés sur les ontologies nécessitent un effort de programmation conséquent et couplent très étroitement le modèle de contexte avec le reste du système. En outre, le raisonnement et le partage de connaissances à travers les systèmes sont impossibles en raison du manque de sémantique déclarative des programmes. SOCAM, par exemple, fait l'usage d'une ontologie générale de haut niveau pour spécifier des propriétés contextuelles de base communes et pour affiner cette même ontologie. Dans le but de fournir des possibilités très granuleuses pour spécifier et formaliser le contexte, certaines ontologies spécifiques à un domaine peuvent être définies.

1.4.3 Découverte de ressources

Les mécanismes de découverte de nouvelles ressources sont très rarement implémentés malgré l'importance d'un tel dynamisme, surtout dans un environnement ubiquitaire, où les sources de contexte et les sondes disponibles changent constamment. SOCAM est le seul système basé sur une architecture orientée services. Il offre un service de localisation de services, qui se raccorde dynamiquement aux fournisseurs de contexte disponibles. Cet fonctionnalité manquante à la plupart des frameworks est en effet un tort puisque cela implique que les sources

de contexte utilisées doivent être stables et disponibles en permanence, ce qui n'est pas toujours le cas dans les applications du monde réel. Si une ou plusieurs sources de contexte se comportent mal, cela pourrait conduire à une disponibilité réduite des services et des applications sensibles au contexte.

1.4.4 Gestion du contexte historique

De la même manière, la majorité des frameworks conservent un historique des données de contexte, mais aucun d'entre eux ne les exploitent vraiment. Pourtant, cela pourrait permettre de fournir des services sensibles au contexte fortement adaptables, si le contexte historique était utilisé pour implémenter des algorithmes d'auto apprentissage. De plus, l'information de contexte pourrait même être anticipée pour offrir proactivement un ensemble de services à l'utilisateur.

1.4.5 Sécurité et confidentialité

Un autre aspect important à ne pas négliger est la sécurité et la confidentialité. L'information de contexte le plus souvent considère le profil utilisateur et d'autres informations sensibles. Pour cette raison, des concepts sont requis pour exprimer des règles politiques et définir la propriété de l'information de contexte.

CoBrA inclus son propre langage de définition de politiques appelé Rei [13], pour contrôler de manière flexible les accès aux données de contexte. Ce langage de définition de politiques est calqué sur les concepts déontiques de droits, d'interdictions, d'obligations et de dispenses, et contrôle l'accès aux données grâce à des règles de politique modifiables dynamiquement et dépendantes d'un domaine particulier. Gaia fait l'usage de plusieurs mécanismes pour définir les restrictions en termes de confidentialité et pour sécuriser les communications lors de localisations d'individus. Le Contexte Toolkit implémente le concept de propriété des information de contexte, qui autorise seulement le propriétaire des informations à consulter les données sondées.

L'aspect sécurité n'est pas implémenté dans les autres frameworks. De nombreux systèmes sont totalement dépourvus de modèles de sécurité, d'autres offrent des mécanismes de sécurité très génériques et seulement quelques systèmes sont dotés d'options de sécurités avancées et suffisantes.

1.4.6 Conclusion

Chacun des systèmes et frameworks présentés dans cet état de l'art implémentent leur propre format pour la description des informations de contexte et des mécanismes de communication dissemblables. La définition d'un modèle standard d'abstraction des données de contexte semble être la priorité première dans l'évolution des systèmes sensibles au contexte existants. Si le degré de sophistication des systèmes distribués font leur force de calcul, leur extrême complexité sont souvent la cause de leur boycott. Le modèle issu de cette étude devra être simple, accessible, extensible et doté d'un haut degré de formalisation.

Chapitre 2

Problématiques émergentes et orientations de recherche

Dans ce mémoire, nous avons décrit divers modèles et principes de conception d'un système sensible au contexte et présenté une multitude d'intergiciels et d'approches distribuées pour simplifier la configuration d'applications sensibles au contexte. De cet état de l'art se dégagent plusieurs problématiques dans le développement d'un système de configuration basé sur le contexte auxquelles nous tenterons d'apporter une solution.

2.1 Intergiciel et détails d'implémentation

Le besoin d'un nouvel intergiciel se fait ressentir du coté de l'implémentation. Les technologies intergicielles actuelles ne sont pas adaptées pour le prise en considération des restrictions imposées pas la mobilité et les systèmes environnementaux intelligents : connexions volatiles, traitements et restrictions mémoire sur les périphériques mobiles, canaux de communication étroits, écrans réduits, mécanismes d'entrées restreintes, et la liste continue. Il existe néanmoins une gamme d'implémentations de systèmes sensible au contexte dans la littérature avec quelques prototypes fonctionnels :

- **Hydrogen** [11] : une architecture en trois couches.
- **Gaia** [7] : une autre infrastructure intergicielle, étends les caractéristiques génériques des systèmes d'exploitation pour y incorporer la sensibilité au contexte.
- **CybreMinder** [2] : un système sensible au contexte permettant de générer des messages de rappel.
- **Context Toolkit** [9] : une architecture basée sur les widgets.

2.2 Représentation des informations de contexte

L'une des principales innovations à apporter dans les infrastructures d'applications sensibles au contexte réside dans l'introduction d'une interface présentant un niveau d'abstraction élevé. Cela permettrait de représenter la connectivité des composants applicatifs avec les politiques de haut niveau qui la régissent.Cette couche doit rester simple d'utilisation pour les développeurs d'applications, et le modèle améliorer simultanément l'automatisation et la sécurité.

2.2.1 Ontologie de contexte

La représentation des informations de contexte est grande préoccupation. Nous avons présenté dans ce mémoire une approche basée sur les ontologie très générique. Elle est basée sur quatre concepts fondamentaux : utilisateur, environnement, plateforme et ressources. Actuellement, les ontologies sont surtout utilisées pour permettre la communication entre les différents périphériques dans le même réseau. Comme proposé par le ContextUML [18], le Langage de Modélisation Unifié (UML) peut également être utilisé pour modéliser le contexte. Ces modèles pourraient être utilisés pour séparer la définition et l'information liée au contexte de l'implémentation spécifique. Il existe d'autres caractéristiques qui font que l'information de contexte est difficile à modéliser : comme abordé précédemment, il est parfois nécessaire de différencier une information statique d'une information dynamique.

2.2.1.1 Ontologie de services

- ServiceProfile (partie "Quoi")
- ServiceModel (partie "Comment abstrait")
- ServiceGrounding (partie "Comment concret")

Tout comme pour la standardisation de la gestion intelligente de la configuration, la classe ServiceModel est très importante. Les services sont modélisés comme des processus et la classe Process est utilisée pour indiquer les opérations à des niveaux granularité différents. La classe Process comprend principalement deux types de processus : atomiques et composites. Puisque l'information de gestion de configuration est définie comme un ontologie OWL, l'information peut être utilisée comme les paramètres des opérations de gestion de la configuration, qui sont définis sous la forme de processus composites. Chaque service de gestion de configuration correspond à un processus composite, lui-même composé de plusieurs processus atomiques. Quand le gestionnaire invoque un service de gestion de configuration prédéfini en OWL-S, le service sera alors exécuté pour le dispositif réseau en place.

2.3 Règles sémantiques et politiques de comportement

Un autre domaine de recherche concernerait l'utilisation de règles pour exprimer les comportements souhaités en termes d'éléments de haut niveau. Des langages de définition de règles sont utilisés dans certains cas pour obtenir la sensibilité au contexte. CRIME [16] par exemple, est une implémentation prototypique du modèle Fact Space, qui est un langage de coordination fournissant aux applications une vue de leur environnement. Les règles CRIME décrivent le comportement des applications conformément à l'information de contexte. CRIME traite également les déconnexions en invalidant les faits et conclusions qui sont tirés des périphériques qui ne sont plus disponibles dans l'environnement.

2.3.1 Gestion basée sur des politiques (Policy-Based)

La politique est identifiée comme une spécification de la configuration moyenne du système sur des périodes persistantes. Un aspect important de cette définition est qu'elle permet une certaine tolérance à l'erreur, nécessitée par les évènements aléatoires se produisant susceptibles de corrompre la politique en place dans la boucle de gestion du système. Il existe un élément probabiliste ou stochastique pour le comportement du système et, par conséquent, la politique ne peut être qu'une propriété moyenne en général.

2.3.2 Théorie de la promesse

Le modèle permettant la définition des politiques d'administration serait un modèle orienté sur les ontologies et basé sur la théorie de la promesse. Celle-ci s'appuie sur un contrôle évolutif des objets intelligents, contrairement aux modèles impératifs plus traditionnels pensés comme des systèmes de gestion descendants. Dans ces derniers, le gestionnaire central doit être informé des commandes de configuration des objets sous-jacents et de l'état actuel de ces objets.

Au sein du contexte, le modèle fournit une série d'objets qui définissent l'application. Les objets englobent les terminaux, les groupes de terminaux et les politiques qui définissent leurs relations.

L'infrastructure conçoit un modèle d'objet pour le déploiement d'applications, ces dernières constituant le point central. Historiquement, les applications étaient limitées par les capacités du réseau et par des configurations visant à prévenir leur utilisation abusive. Des concepts tels que l'adressage, le VLAN et la sécurité sont depuis toujours intrinsèquement liés, ce qui limite l'évolutivité et la mobilité des applications. Alors que les applications sont redessinées pour la mobilité et l'évolutivité web, cette approche traditionnelle empêche leur déploiement rapide et homogène.

La théorie de la promesse est une théorie nouvelle sur ce qu'il peut arriver dans un réseau de composants entièrement autonomes.

Plutôt que d'adopter la croyance conventionnelle que "seulement ce qui est programmé peut arriver", elle prends le point de vue opposé : "seulement ce qui est promis peut être prédit". Elle aborde donc la gestion du point de vue de l'incertitude avec réalisme plutôt que celui de la foi en la conformité. Dans la théorie de la promesse, on suppose un point de vue orienté service des interactions entre les différents dispositifs informatiques. Chaque nœud propose de jouer un rôle dans un réseau de collaboration en promettant de restreindre son comportement de différentes manières. Une utilisation typique de la théorie de la promesse est d'examiner le comportement en régime permanent d'un certain nombre d'agents (composants) autonomes. La théorie de la promesse ne détermine pas nécessairement pleinement le comportement de ses composants, elle représente seulement les contraintes au sein d'un ensemble de comportements définis. Elle n'est pas limitée aux promesses linéaires.

La théorie de la promesse adhère à un point de vue orienté service des politiques des gestion et utilise un langage graphique pour composer et analyser les propriétés systèmes.

Une caractéristique essentielle des promesses est qu'elles séparent clairement ce qui contraint et à qui la contrainte s'applique. C'est un autre domaine qui soulève bien souvent des confusions dans la théorie des politiques. Néanmoins, le modèle ontologique permet un fois de plus de bien distinguer ces deux aspects grâce à son haut degré de formalisme.

2.3.2.1 Définition de la promesse

Une promesse est différente d'un engagement : un engagement est le moment où un agent rompt avec une ligne de conduite pour une autre discontinue avec des vues sur un objectif, la plupart du temps à travers une action spécifique ou un investissement sur des résultats futurs. Dans certains cas, l'acte d'engagement peut résulter en une promesse persistante, mais promettre n'implique pas une action provoquant un changement discontinu.

> "Une promesse est la spécification d'un état ou comportement
> ultérieur d'un agent autonome à un autre. Elle est ainsi,
> une unité de politique." [4]
> *Mark Burgess, Alva Couch, Modeling Next Generation Configuration Management Tools,*
> *Pp. 131-147 of the Proceedings of LISA '06, Décembre 2006*

FIGURE 2.1: La promesse par Burgess et al. (2006)

Les promesses sont faites à un agent par un agent et sont modélisées par une relation unidirectionnelle labellisée par un *corps* de promesse qui définit la substance de la promesse. Une promesse avec le *corps* **+b** est une déclaration pour "donner" un comportement d'un agent à un autre, tandis qu'une promesse avec le *corps* **-b** est la spécification de quel comportement sera reçu, accepté ou utilisé d'un agent à l'autre.

2.3.2.2 CfEngine3 : une implémentation de référence de la théorie de la promesse

CfEngine3 implémente deux approches :

- **Centralisée** : le système va pousser avec force les règles et règlements établis de manière centralisée avec ou sans la volonté de l'utilisateur final hôte.
- **Basé sur des politiques** : cette approche fonctionne de la même manière que le protocole SNMP (Simple Network Management Protocol), à l'exception qu'elle donne une autonomie totale aux agent, de manière à ce qu'ils aient plein droit de tirer et implémenter un ensemble centralisé de politiques de configuration.

En théorie, chacun des agents autonomes va faire formuler une promesse sur le comportement qu'on attend de lui, basé sur un choix qui lui est propre. C'est ce que rend la théorie de la promesse optimale pour un framework de gestion basé sur des politiques.

La théorie de la promesse décrit des services régies par des politiques, dans un framework d'agents complètement autonomes, qui s'aident les uns les autres seulement par une coopération volontaire. Dans CfEngine3, chaque élément de configuration va faire une promesse concernant ses caractéristiques propres et sa relation avec d'autres éléments de configuration.

Nous souhaitons être en mesure de promettre que le système est cohérent, de le vérifier et d'effectuer des changements seulement si nos promesses ne sont pas tenues. Cela nécessite, en plus de l'implémentation initiale de ces promesses, la vérification continue de l'état de chacun des éléments de configuration vis-à-vis de leur promesses respectives pour assurer un système cohérent et en conformité avec les politiques en vigueur.

2.3.2.3 Processus de raisonnement lié à la promesse

Le procédé de raisonnement qui accompagne l'émanation d'un promesse, se divisent en plusieurs sous-étapes, accrochez-vous :

- **Préparation de la promesse** : Processus de raisonnement effectué par A conduisant à la conception, la synchronisation et la délivrance de la promesse P par A.
- **Analyse de crédibilité** : Processus de raisonnement où les agents C dans le champ d'application de la promesse P déterminent la crédibilité qu'ils assignent à A promettant P compte tenu des faits connus de A (mais à l'exception des informations historiques précises sur le comportement individuel d'un membre de sa classe d'agent)
- **Détermination préliminaire de la confiance** : Processus de raisonnement effectué par C (C dans le domaine d'application de la promesse P) servant à :
  1. Déterminer la confiance que C accorde à A avant même de connaitre la promesse P (confiance préalable)
  2. Spécifier quelles sont les attentes générées par la prise en considération de la promesse P.
- **Délibération de contre-promesse** : Processus de raisonnement effectué par B concernant les contre-promesses pouvant potentiellement être émises par B en retour de considération de P (B dans le champ d'application de P)
- **Prédiction de l'impact de la promesse** : (cela peut être réalisé à condition que B ait émis une ou plusieurs contre-promesses plausibles)
  1. Processus de raisonnement effectué par B (dans le champ d'application de P) et C (n'importe quel agent

dans le champ d'application de P) servant à déterminer les (le changement des) attentes que P créé dans B (et que A a l'intention de générer).

2. Processus de raisonnement visant à la modification des ententes (détenues par B ou C) étant donné la changement des attentes de chacun d'entre eux amené par la prise en considération de la promesse P.

– **Évaluation de la promesse** : Processus de raisonnement effectué par C concernant :

1. La façon dont C va évaluer si la promesse de A a été tenue ou non.
2. L'évaluation de cette dernière au moyen de la méthode la plus adéquate

– **Surveillance de rétraction de promesse** : A peut être amené à un stade ultérieur à émettre une autre promesse Q, pour laquelle la tenue n'est pas compatible avec la tenue de P. Dans ce cas, Q qualifie un retrait de P. Un agent C applique un processus de raisonnement qui surveille et évalue les promesses postérieures émises par A pour déterminer si celles-ci seront amenées à rompre la promesse P et induire sont retrait.
– **Mise à jour de la confiance** : Processus de raisonnement en place pour chacun des agents C dans le champ d'application de P visant à mettre à jour la confiance préalablement accordée à A, en adéquation avec la résultat de l'évaluation que C fait concernant le degré avec lequel la promesse P a été tenue par A.
– **Critère de réputation** : Processus de raisonnement effectué par chacun des agents C dans le champ d'application de P visant à échanger entre les différent agents les effets des mises à jour de confiance. Le flux de réputation permet a un agent C n'ayant aucuns aprioris sur un agent A d'acquérir une confiance initiale en prenant en considération les preuves recueillies par les autres agents (parce que même les agents ont des casiers judiciaires).

2.3.2.4 Représentation de la promesse : π-calculus

Type de promesse	**Notation**	**Interprétation**
Basique	$n_1 \xrightarrow{\pi} n_2$	Fournit un service / un flux
Coopérative	$n_1 \xrightarrow{C(\pi)} n_2$	Imite / Suit
Utilisatrice	$n_1 \xrightarrow{U(\pi)} n_2$	Utilise / Accepte de la part de
Conditionnelle	$n_1 \xrightarrow{\pi_1/\pi_2} n_2$	"File" de promesses : π_1 if π_2

Les promesses de services basiques forment un certain nombre de types, comme par exemple 'fournir un web service en moins de 5 millisecondes' ou 'donner n'importe quel information sur l'agent X'. D'autres exemples :

– $X \xrightarrow{q \leq q_0} Y$: l'agent X promet de ne jamais éxeder la limite $q \leq q_0$.
– $X \xrightarrow{q=q_0} Y$: l'agent X promet de satisfaire $q = q_0$.
– $X \xrightarrow{\ell \subseteq L} Y$: X promet de garder ℓ comme sous-langage du langage L
– $X \xrightarrow{S} Y$: X offre le service S à Y.
– $X \xrightarrow{R} Y$: X promet de relayer R à Y.
– $X \xrightarrow{\neg R} Y$: X promet de ne jamais relayer R à Y.
– $X \xrightarrow{S,t} Y$: X promet de répondre avec le service S en l'espace de t secondes.

Une promesse est dite “rompue“ si un agent fait deux promesses différentes du même type en même temps (différent d'une promesse qui aurait expiré ou changé).

2.4 Consistance de l'information et tolérance à l'échec

2.4.1 Algorithme de consensus

Le consensus est un problème fondamental dans les systèmes distribués tolérants aux pannes. Un consensus implique de multiples serveurs acceptant des valeurs. Une fois qu'ils atteignent une décision sur une valeur, cette décision est définitive. Les algorithmes de consensus typiques sont amenés à faire des progrès lorsque la majorité de leurs serveurs sont disponibles, par exemple, un cluster de 5 serveurs peut continuer à fonctionner même si deux serveurs ne sont plus disponibles. Si plusieurs serveurs échouent, ils cessent de faire des progrès (mais ils ne retournerons jamais de valeurs erronées).

2.4.1.1 Pourquoi Raft ?

Raft est un algorithme de consensus qui est conçu pour être facile à comprendre. Il est équivalent à Paxos dans la tolérance aux pannes et en termes de performance. La différence est qu'il est décomposé en sous-problèmes relativement disjoints, et traite de manière rigoureuse toutes les pièces majeures nécessaires pour obtenir un système cohérent.

Il existe des écarts significatifs entre la description de l'algorithme de Paxos et les besoins d'un système dans le monde réel. Le système au final serait basé sur un protocole dont il n'existe pour l'heure aucune preuve de son bon fonctionnement.

Inconvénients de Paxos

- Extrêmement difficile à comprendre. L'opacité de Paxos dérive de son choix d'avoir un sous-ensemble de décrets uniques comme son fondement.
- Ne fourni pas une bonne base pour la construction d'applications pratiques. Par exemple, il y peu d'avantages à choisir de manière indépendante une collection d'entrées dans le registre, pour les fusionner par la suite dans un journal séquentiel ; cela ajoute juste de la complexité. Il est plus simple et plus efficace de concevoir un système autour d'un journal, où les nouvelles entrées sont ajoutées séquentiellement dans un ordre contraint.
- Utilise une approche symétrique pair-à-pair à sa base. Cela a un sens dans un monde simplifié où seule une décision sera prise, mais peu de systèmes concrets utilisent cette approche. Si une série de décisions doivent être prises, il est plus simple et plus rapide d'élire d'abord un leader, puis de le laisser coordonner les décisions.

Avantages de Raft

- Raft utilise une forme plus solide de leadership que les autres algorithmes de consensus. Par exemple, les entrées du registre de seront communiquées que par le leader vers les autres serveurs. Cela simplifie largement la gestion de la réplication des journaux et rend Raft beaucoup plus facile à comprendre.
- Raft utilise des minuteries aléatoires pour élire les leaders. Cela ajoute un peu de mécanismes au système de pulsation déjà nécessaire dans n'importe quel algorithme de consensus, tandis que la résolution des confits se fait simplement et rapidement.
- Le mécanisme de Raft pour changer l'ensemble des serveurs de la grappe utilise une nouvelle approche de consensus conjointe où les majorités des deux différentes configurations se superposent pendant les transitions. Cela permet au groupe de continuer à fonctionner normalement pendant les changements de configuration.

2.4.1.2 Architecture de machines à états répliquées

L'algorithme de consensus gère un registre répliqué contenant les commandes de la machine à états de chacun des clients dans le réseau. Les machines à états traitent des séquences identiques de commandes à partir de journaux partagés, de sorte qu'ils produisent les mêmes résultats.

Les systèmes à grande échelle n'ayant qu'un seul cluster leader utilisent une machine à état distincte pour gérer les élections et pour stocker les informations de configuration qui doivent survivre aux crashs du leader (exemple : Chubby, ZooKeeper).

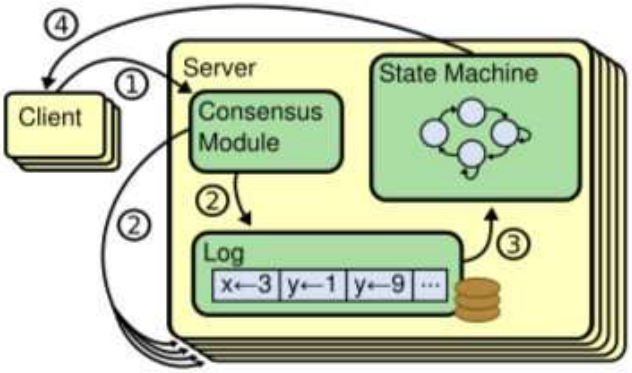

FIGURE 2.2: Procédé de réplication des machines à états

Garder le registre répliqué consistant est le travail de l'algorithme de consensus.

Les algorithmes de consensus pour les systèmes concrets ont généralement les caractéristiques suivantes :

– Ils assurent la sécurité (ne retournent jamais de résultat erroné) sous toutes conditions non byzantines, y compris les retards dans le réseau, les pertes de partitions et de paquets, la duplication et la réorganisation.
– Ils sont entièrement fonctionnels (disponibles) dans la mesure où au moins la majorité des serveurs sont opérationnels et peuvent communiquer avec les uns les autres et avec les clients. Ainsi, un groupe type de cinq serveurs peut tolérer la défaillance de deux ses membres. Les serveurs sont supposés tomber en panne en s'arrêtant ; ils pourront plus tard retrouver un état stable et rejoindre le cluster.
– Il ne dépendent pas de la contrainte temporelle pour assurer la cohérence des journaux : des horloges défectueuses et des retards abusifs dans la délivrance des messages peuvent causer des problèmes de disponibilité.
– Dans le cas le plus courant, une commande peut s'achever aussitôt que la majorité de la grappe a répondu à un seul tour d'appels à procédures distantes ; une minorité de serveurs lents de doivent pas impacter les performances globales du système.

2.4.1.3 États des serveurs

Un cluster Raft comprend une multitude des serveurs (cinq est un nombre courant permettant au système de tolérer deux échecs). À n'importe quel moment donné, chacun des serveur est dans l'un de ces états :

– **Leader** : En cycle normal, il existe un et un seul *leader* et tous les autres serveurs sont *followers*. Le *leader* traite les requêtes de tous les clients (si un client contact un *follower*, le *follower* le redirige vers le *leader*).
– **Follower** : Les *followers* sont passifs : ils n'émettent jamais de messages par leur propre initiative, ils se contentent simplement de répondre aux requêtes émises par le *leader* et les *candidats*.
– **Candidat** : Statut intermédiaire utilisé lors de l'élection d'un nouveau *leader*. La figure 2.3 montre les différents états et transitions possibles dans le consensus de Raft.

Les *followers* ne font que répondre aux requêtes des autres serveurs. Si un *follower* ne reçoit plus de communications, il devient alors *candidat* et initie une élection. Le *candidat* qui reçoit les votes d'un majorité du cluster complet devient le nouveau *leader*. D'une manière générale, les *leaders* opèrent jusqu'à ce qu'ils échouent.

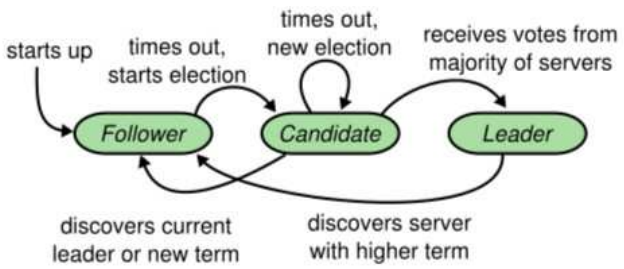

FIGURE 2.3: États et transitions dans un cluster Raft

Chapitre 3

Validation et détails d'implémentation

3.1 Vue d'ensemble de la solution proposée

Comme on le voit 3.1

3.2 Framework de gestion de configuration

La gestion de la configuration est le processus de contraindre le comportement d'un réseau de machines de manière à ce que le comportement de chaque machine soit conforme aux politiques et lignes directrices prédéfinies, et accomplisse les objectifs d'entreprise prédéterminés. Cela implique :

- Des personnes
- Outil gestion de configuration "zéro" (exemple : CfEngine)
- Un ensemble ce machines interconnectées
- Un ensemble de processus de configuration destiné à

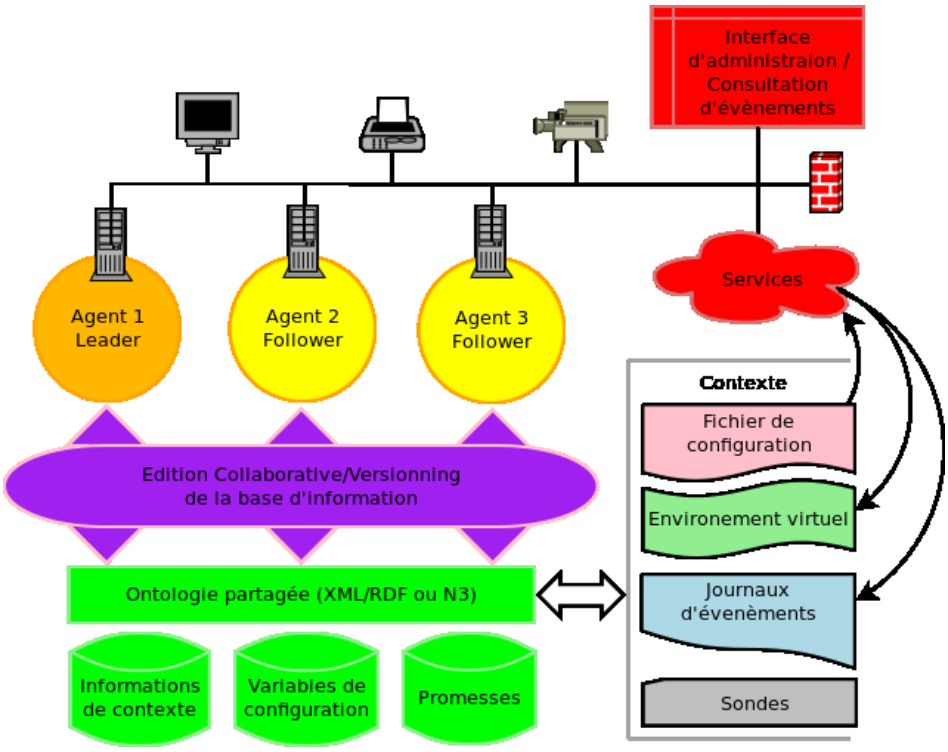

FIGURE 3.1: Schéma d'implémentation du système multi-agents

aboutir à un système conforme à la politique en vigueur.

Les paramètres de configuration peuvent être la permission d'un fichier, l'adresse d'un carte réseau, le type de système de fichier pouvant être monté sur un volume disque.

La possibilité d'automatiser la configuration des réseaux IP (Internet Protocol) en utilisant une technologie sémantique est également très prometteuse. L'initiative connexe de créer un environnement de réseau de capteurs basé sur le ontologies semble également être une solution viable.

La plupart des organisations aujourd'hui n'ont aucun procédé systématique pour gérer leurs informations de configuration. Cela signifie que des informations comme l'adresse IP d'une machine et le statut de son service réseau sont stockées de manière dispersée et désorganisée. Certains outils de supervision tels que Nagios présente une manière de réorganiser ces informations provenant de sources dispersées.

L'idée d'utiliser une CMDB (Configuration Management Database) est également un alternative permettant de solutionner le problème ci-dessus. Toutefois, le besoin pour un gestionnaire d'informations de plus haut niveau est mentionné dans de nombreuses littératures.

Cette nouvelle façon de renforcer l'efficacité de la gestion des connaissances dans le domaine de la gestion de configuration, nécessite la possibilité d'extraction automatique de données à partir d'emplacements de stockage précédents et la possibilité de mise à jour automatique de stockages permanents tels que la CMDB afin d'augmenter la qualité de l'information sous-jacente.

La possibilité de ces deux fonctions à savoir, l'extraction automatique des données et également la mise à jour automatique des informations stockées ont été fournies par ITIL.

Le travail de ce mémoire est de démontrer les possibilités d'obtenir un gestionnaire de connaissances intégré en conjuguant les ontologies, la théorie de la promesse et le consensus de Raft. La structure de l'information de configuration stockée dans une CMDB et dans d'autres fichiers sera représentée en concordance avec les standard de OWL. Cette structure de domaine de connaissance sera implémentée comme une promesse de structure d'information.

La Configuration Management Database (abrégé CMDB), ou base de données de gestion de configuration, est une base de données unifiant les composants d'un système informatique. Elle permet de comprendre l'organisation entre ceux-ci et de modifier leur configuration. La CMDB est un composant fondamental d'une architecture ITIL.

```
#include <iostream>
int main(int argc, char** argv) {
  std::cout << "Hello World." << std::endl;
  return 0;
}
```

FIGURE 3.2: Example source code.

3.3 Ontologie pour un gestionnaire de configuration basé sur la théorie de la promesse

3.4 Conclusion

Les technologies basées sur le contexte vont jouer un rôle fondamental dans la prochaine génération de systèmes informatiques à mesure que la complexité des logiciels, la diversité et l'omniprésence des périphériques continuent d'augmenter. Cependant, elle doivent offrir des mécanismes permettant la gestion automatique des dépendances inter-composants et composant/ressource. Dans le cas contraire, le développement de systèmes basés sur les composants demeura difficile à appréhender et conduira bien souvent à des systèmes peu fiables et pas assez robustes.

Les environnements qui composeront l'informatique ubiquitaire de demain sera composé de milliers de périphériques, avec des millions de composantes logicielles. Les systèmes actuels s'appuient fortement sur la configuration manuelle, ce qui à un telle échelle n'est plus possible. Il n'existe que deux issues possible à cette situation : une configuration statique, ou une configuration dynamique et automatique. Puisque les environnements ont tendance à être de plus en plus dynamiques, la configuration autonome semble incarner la seule solution viable.

Bibliographie

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Baltzer journals september 23, 1996. 1997.

[2] G. D. Abowd, M. Ebling, G. Hung, H. Lei, and H.-W. Gellersen. Context-aware computing [guest editors' intro.]. *Pervasive Computing, IEEE*, 1(3) :22–23, 2002.

[3] M. Ackerman, T. Darrell, and D. J. Weitzner. Privacy in context. *Human–Computer Interaction*, 16(2-4) :167–176, 2001.

[4] M. Burgess and A. Couch. Modeling next generation configuration management tools. *Proceedings of LISA '06 : 20th Large Installation System Administration Conference*, pages 131–147, Dec. 2006.

[5] C. Catharina and Kari. Context aware management architecture, 2002.

[6] H. Chen, T. Finin, and A. Joshi. An intelligent broker for context-aware systems. In *Adjunct proceedings of Ubicomp*, volume 3, page 183–184, 2003.

[7] S. Chetan, J. Al-Muhtadi, R. Campbell, and M. D. Mickunas. Mobile gaia : a middleware for ad-hoc pervasive computing. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, page 223–228. IEEE, 2005.

[8] A. K. Dey. *Providing architectural support for building context-aware applications.* PhD thesis, Georgia Institute of Technology, 2000.

[9] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2) :97–166, 2001.

[10] M. Ebling, G. D. Hunt, and H. Lei. Issues for context services for pervasive computing. In *Middleware 2001 Workshop on Middleware for Mobile Computing, Heidelberg*, 2001.

[11] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices-the hydrogen approach. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, page 10–pp. IEEE, 2003.

[12] J. I. Hong and J. A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2) :287–303, 2001.

[13] Kagal and Finin. A policy language for a pervasive computing environment. 2005.

[14] Korpipää and Mäntyjärvi. An ontology for mobile device sensor-based context awareness. 2003.

[15] P. LÉVY. *Qu'est ce que le virtuel ?* LA DECOUVERTE, July 2010.

[16] A. L. Murphy and International Conference on Coordination Models and Languages. *Coordination models and languages : 9th international conference : proceedings.* Springer, Berlin, 2007.

[17] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to context than location. *Computers & Graphics*, 23(6) :893–901, 1999.

[18] Q. Z. Sheng and B. Benatallah. ContextUML : a UML-based modeling language for model-driven development of context-aware web services. In *Mobile Business, 2005. ICMB 2005. International Conference on*, page 206–212. IEEE, 2005.

[19] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

[20] S. Voida, E. D. Mynatt, B. MacIntyre, and G. M. Corso. Integrating virtual and physical context to support knowledge workers. *IEEE Pervasive Computing*, 1(3) :73–79, 2002.

[21] T. Winograd. Architectures for context. *Human-Computer Interaction*, 16(2) :401–419, 2001.

Annexe A

Ontologie de contexte

A.1 Entités de l'ontologie

- **Machine** est n'importe quel périphérique en mesure d'accepter et de traiter des informations pour fournir le résultat désiré basé sur un programme ou une séquence d'instructions sur commment les données doivent être traitées.
- **Computers** sont le type principal de machines dans cette étude. C'est pourquoi les deux terminologies sont interchangeables dans diverses sections de ces travaux. Les ordinateurs personnels (PCs), les ordinateurs portables et les serveurs héritent tous de ce type.
- **Operating System** est un programme faisant le pont en les composants logiciels et les composants matériels sous-jacents d'une machine. Le système d'exploitation est une entité comportant plusieurs sous-classes disjointes telles que Windows XP, OS X ou Linux.
- **Packages** sont les programmes applicatifs conçus pour servir un but spécifique comme distribuer un service local, réseau ou web sur une ou plusieurs machines. Le paquet Apache est en bon example d'entité conçue pour activer un service web.
- **Service** est une fonctionnalité spécifique d'un système informatique comme un service web ou réseau offert par une machine. Un processus qui est défini dans ce document comme un ensemble ou d'une partie d'un ensemble de mesures permettant la fourniture de services en exécutant des activités réelles en tâche de fond.
- **Command** est un utilitaire pouvant être utilisé par les utilisateurs pour démarrer un processus spécifique, à la condition qu'il n'existe pas de plannification pour l'execution de cette même tâche. Un très bon exemple permettant d'illustrer cette relation entre ces deux entités serait un web service, qui nécéssite qu'un processus tel que "httpd" soit démarré en arrière plan à l'aide de la commande "

  ```
  httpd start
  ```

  ".
- **Storage** est une partition logique d'un média de stockage physique qui sert principalement d'hôte pour les différents types de fichiers. Un fichier est défini comme une collection d'informations complète comme un programme, un ensemble de données utilisées par un programme ou un document créé par un utilisateur.
- **Interface** est défini comme un périphérique permettant l'accès à un réseau par une machine.

A.2 Propriétés objets de l'ontologie (Relations)

- **Caused By** : Un type d'association où une entité joue le rôle d'affecter l'autre en changeant son état de l'état X à l'état Y.
- **Configured By** : Quand une entité X peux apporter les changemments nécéssaires à l'entité Y pour lui permettre d'accomplir ses objectifs et ses intentions. Le lien entre une machine et un progiciel de gestion de configuration comme CfEngine 3 est un exemple typique de ce type de relations.
- **Edited By** : Décrit qu'un fichier peut être modifié par un éditeur tel que l'utilisateur du fichier à certaines fins.
- **Managed By** : Si X est responsable du bon fonctionnement de Y dans l'accomplissement de son objectif, X est dit manager de Y. Le fait qu'une entité hérite de ce rôle implique une supervision constante et des prises de mesures lorsqu'intervient une déviation dans le comportement souhaité.
- **Monitoring By** : La supervision est principalement la responsibilité de garder un oeil sur quelque chose. Si une entité X supervise Y, elle surveille une quelquonque altération de comportement pour en informer le ou les managers en charge de cette entité.
- **Component Of** : La relation entre une entité X, composant d'une entité plus globale Y qui joue le rôle d'encapsulateur pour ce composant X.
- **Owned By** : La propriété est un type de relation pouvant exister entre entité devenue possession et son propritaire. Le lien entre un fichier et son propriétaire est un exemple de cet type d'association.
- **Promised By** : La relation en une promesse et l'entité qui la formule.
- **Required By** : Si X dépend de Y dans l'accomplissement de ses intentions, la relation entre ces deux entités est de type "Required By".
- **Runs On** : Si l'entité X exerce ses activités ou son exécution sur l'entité Y, la relation est appelé "Runs On".
- **Written By** : La relation entre une écriture X et son auteur Y.
- **Used By** : Si une entité X fait l'usage d'une entité Y à n'importe quelle moment de son exercice, la relation est de type "Used By".
- **Started By** : Quand une entité X est responsable du démarrage d'une entité Y ou qu'elle commence à se comporter d'un certaine manière, l'association est dite de type "Started By".
- **Provided By** : Quand une entité X a le potentiel et la bonne volonté de fournir quelque chose à une autre entité Y, la relation entre ces deux entités est appellée "Provided By".
- **Installed By** : Une entité X peut mettre en place un programme Y dans une machine de manière à accomplir son but recherché. La relation existante entre X et Y est alors de type "Installed By".
- **Contained In** : Si une entité X est contenue dans Y, physiquement ou conceptuellement, la relation est de type "Contained In".

A.3 Propriétés de données de l'ontologie (Attributs)

- **AllowConnectFrom** : Liste d'adresses IP ou de nom d'hôtes suceptibles d'avoir plus d'une connexion au port d'un serveur.
- **CheckForeign** : Indique s'il est nécessaire de vérifier les autorisations sur le répertoire racine lors de la recherche de la profondeur.

- **CheckRoot** : Liste de nom d'hôtes ou d'adresses IP auxquels accorder un droit de lecture sur le serveur.
- **ForceIpv4** : Indique s'il y a un usage forcé de l'adresse IP pour les connexions.
- **IsMachineVirtualized** : Indique si une machine est virtuelle ou non.
- **PackageFileRepository** : Liste de répertoires locaux à la machine pour la recherche de paquets.
- **TrustKeyFrom** : liste des hôtes pour lesquelles une machine acceptera les clés publiques sur base de la confiance. Définition des types d'occurences qui sont liés avec le type d'entité stockage sont répertoriés ci-dessous. Un stockage est une partition logique d'un support de stockage physique.
- **FreeSpace** : Pourcentage minimum ou absolu d'espace disque devant être disponible sur le support de stockage avec de lever un avertissement.
- **MountType** : Type de protocole d'un système de fichier distant.
- **FileSystemFlag** : Liste des options de menu à définir pour les drapeaux de système de fichiers BSD.
- **Atime** : Plage temporelle d'accès pour les fichiers adimissibles.
- **SecureInput** : Indique si les fichiers d'entrée sont accessibles en écriture pour les utilistateurs non-autorisés.
- **MountType** : L'option de menu pour le type de liens à utiliser lors de la copie comme le lien symbolique et le lien physique.
- **AuditingEnabled** : Indique si la fonctionnalité de journal d'audit d'un paquet est activé ou non.
- **LogLevel** : Le niveau de rapport envoyé à syslog.
- **Architecture** : L'achitecture relative à la section de paquets comme "x86-64" ou "i386".
- **BuildsOn** : Liste de groupemment de promesses sur lesquelles s'appuie ou dépend une promesse d'une certaine manière (pour la gestion des connaissances)
- **HomeDirectory** : Répertoire contenant les fichiers personnels d'un utilisateur.
- **ShellAccount** : Compte personnel donnant accès à une invite de commande Unix.
- **Module** : Indique s'il faut s'attendre à un protocole de module CfEngine.