ContextAA : plateforme sensible au Contexte pour aborder le problème de l'espace intelligent ouvert

par

Patrice Roy

Thèse présentée au Département d'informatique
en vue de l'obtention de docteur ès sciences (Ph.D.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, juillet 2019

Le 30 juillet 2019

le jury a accepté le mémoire de Monsieur Patrice Roy dans sa version finale.

Membres du jury

Professeur Bessam Abdulrazak,
Directeur de recherche
Département d'informatique

Yacine Belala
Codirecteur de recherche
Ingénieur logiciel principal
Teksystems Global Services

Professeur Gabriel Girard
Membre interne
Département d'informatique

Abdenour Bouzouane
Professeur titulaire en informatique
Membre externe
Département d'informatique et de mathématique
Université du Québec à Chicoutimi

Professeur Richard Egli
Président-rapporteur
Département d'informatique

Sommaire

Les environnements intelligents traditionnels sont des espaces délimités à l'intérieur desquels il est possible d'offrir un soutien spécialisé à des individus. Bien que ces espaces aient fait la démonstration de leur utilité, l'avènement des technologies mobiles et le gain de mobilité qui en découle pour les individus profitant d'environnements intelligents complique le maintien d'une continuité de service pour ceux-ci. Pourtant, les besoins des individus ne s'estompent pas lorsqu'ils franchissent le pas de la porte de leur logis.

C'est pour cette raison que nous avons conçu ContextAA, une plateforme visant à résoudre le problème de la continuité de service dans un contexte d'usagers mobiles à travers un environnement hétérogène où la qualité de la connectivité peut varier. ContextAA se veut une solution générale au problème de l'espace intelligent ouvert, concept qui inclut à la fois les espaces intelligents classiques et celui, plus large et sans frontières prédéterminées, dans lequel tout individu et tout agent est présumé capable de mobilité. En ce sens, ContextAA fait le choix de supporter des variations dans la qualité de service, en fonction de la richesse offerte dans l'environnement immédiat, privilégiant en retour une continuité de service.

Les éléments clés de ContextAA sont une architecture répartie reposant sur des composants Hôtes, à l'intérieur desquels logent des Agents standards, participant à l'offre de services de leur Hôte, et des Agents du domaine, chargés d'une mission et opérant sur la base d'une perspective individuée sur le Contexte, que nous nommons le Contexte-selon. Chaque Agent du domaine s'inscrit dans un cycle de vie, et plus spécifiquement dans un cycle opératoire, et peut migrer d'un Hôte à l'autre si cela lui permet de poursuivre sa mission.

Au cœur de ContextAA se trouve le Contexte, qui est à la fois langage et objet de langage, données, métadonnées et opérations sur le Contexte. De par le langage du Contexte et de par son architecture où les opérations sont basées sur le Contexte, ContextAA se présente comme une architecture capable d'accommoder des nœuds à faibles ressources, comme on en trouve par exemple dans le monde de l'Internet des objets, et où les Agents sont susceptibles de pouvoir poursuivre leur mission malgré la mobilité de leur Hôte, même lorsque cet Hôte se trouve en situation de connectivité limitée.

ContextAA est utilisé sur des projets concrets, comme base sur laquelle sont maintenant construits des outils de plus haut niveau, et permet à la fois de mieux profiter des espaces intelligents traditionnels et d'aborder les problèmes spécifiques à l'espace intelligent ouvert.

Remerciements

Ce fut une longue route, incluant un divorce, un mariage, une naissance, quelques décès, quelques années de syndicalisme actif, une grève étudiante d'une ampleur inattendue. Une vie familiale spectaculaire. Une carrière qui a beaucoup bouillonné. Une vie pleine de surprises.

Merci à Bessam Abdulrazak et à Yacine Belala. Je ne suis pas un étudiant facile à diriger, je sais, mais j'ai apprécié votre soutien et j'ai beaucoup de respect pour vous. Vous avez fait du bon travail, et je vous en remercie.

Merci particulier aux collègues Charles Gouin-Vallerand, qui a suggéré l'idée de visibilité du Contexte, et Benoit Fraikin pour son aide dans la sélection d'une notation pour la représentation des fonctions. De manière générale, les échanges informels avec les membres du DOMUS ont été féconds pour élaborer certaines des métaphores que vous rencontrerez dans ces pages. Merci à mon amie Martine Drapeau pour avoir suggéré le nom Agent du domaine. Merci à Björn Fahller pour une intuition pour la représentation des valeurs dans une sous-expression $eval$. Merci spécial à Victor Manuel Ponce Diaz pour sa collaboration et ses questions au cours des dernières années.

Merci à mes parents Lucille et Réal, à mes beaux-parents Jacqueline, Danielle et Jocelyn, de même qu'à mon ex beau-père Alain, qui m'ont tous soutenus ou aidés à leur manière pendant cette longue aventure. Je vous dois beaucoup et je vous aime.

Merci à mes collègues et ami(e)s, de même qu'à mes ami(e)s tout court. Vous êtes nombreuses et nombreux, et vous êtes importantes et importants pour moi. Je n'ai pas été aussi présent pour vous que j'aurais dû l'être, je le sais, mais je n'ai eu de vous que du soutien et de l'affection. Je suis reconnaissant. J'apprécie particulièrement vos critiques, vos commentaires, et cet irritant « pis, ton doctorat, ça avance? » qui grattait le bobo en ajoutant à l'envie de terminer au plus vite.

Merci au Collège Lionel-Groulx et à l'Université de Sherbrooke, pour vos encouragements et votre tolérance envers moi pendant cette aventure. En particulier, merci à mes étudiant(e)s, pour avoir stimulé mes neurones, mais surtout pour leur patience et (encore une fois) leur tolérance. Je sais que vous êtes conscients que je ne vous ai pas donné ce que je vous aurais donné en temps normal, et j'apprécie que vous ayez accepté de cheminer avec moi malgré tout.

À mes enfants. Mes quatre grandes filles : Marguerite, Calypso, Amandine et Vayda. À mon garçon Ludo, arrivé dans ma vie pendant ma rédaction. Je sais ce que vous avez enduré ces dernières années. Je vous entends. Je vous adore.

Surtout, à Za, Isabelle pour les moins intimes. Mon amour, ma belle. C'est beaucoup grâce à toi que j'y suis arrivé. Que nous y sommes arrivés, vraiment. La vie est plus belle parce que tu es là.

Merci.

Table des matières

Liste des abréviations

API	*Application Programming Interface*
BDI	*Belief-Desire-Intention*
COM	*Component Object Model*
CORBA	*Common Object Request Broker Architecture*
DCE	*Distributed Computing Environment*
EBNF	*Extended Backus-Naur Form*
FIPA	*Foundation for Intelligent Physical Agents*
ICE	*Internet Communications Engine*
IdO	Internet des objets (*Internet of Things*)
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IIOP	*Internet Inter-ORB Protocol*
ISO	*International Standards Organization*
JADE	*Java Agent Development Framework*
JCAF	*Java Context-Aware Framework*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
MQTT	*Message Queue Telemetry Transport*
NAT	*Network Address Translation*
OMG	*Object Management Group*
OSGi	*Open Services Gateway initiative*
OWL	*Web Ontology Language*
PARC	*Palo Alto Research Center*
ReST	*Representational State Transfer*
RFID	*Radio-Frequency Identification*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SOAP	*Simple Object Access Protocol*
STL	*Stepanov and Lee* (mieux connu sous le nom – erronné – de *Standard Template Library*)
URL	*Uniform Resource Locator*
URN	*Uniform Resource Name*

UUID *Universally Unique Identifier*

WSDL *Web Services Definition Language*

XDR *eXternal Data Representation*

XML *eXtensible Markup Language*

Liste des figures

Liste des tableaux

s

Chapitre 1
ContextAA – Introduction et motivation

« Il n'y a qu'une façon d'échouer, c'est d'abandonner avant d'avoir réussi ! » (Olivier Lockert / Hypnose)

L'informatique diffuse, suivant la vision énoncée par Mark Weiser [113], est cette tendance à faire en sorte que l'informatique se fonde dans le décor de la vie des individus. Plusieurs manifestations de cette vision existent au moment d'écrire ces lignes, allant des vêtements intelligents à la domotique; en même temps, le tout convergeant avec une démocratisation des ordinateurs prenant des formes de plus en plus conviviales (téléphones, tablettes et autres).

Un créneau important de la domotique est celui des espaces intelligents, enceintes capables d'offrir une gamme riche de services aux individus qui s'y trouvent. Les composants logiciels qui opèrent dans ces espaces sont en mesure de collaborer pour tirer profit de leurs forces mutuelles, partager de l'information et construire un contexte : une représentation logique de l'individu, de son milieu et de leur interaction.

Un espace intelligent est traditionnellement un environnement immersif aux frontières fixes : une chambre, un appartement, une maison. Le cas type de l'environnement intelligent est un domicile adapté aux besoins spécifiques d'un habitant déterminé, qu'il s'agisse de difficultés motrices, d'une forme de démence, de problèmes reliés au vieillissement, p. ex. : risques de chute menant à des blessures, et besoin de réagir rapidement si un tel événement survient. Dans un espace intelligent adapté à ses besoins, l'individu est accompagné, suivi et, en quelque sorte, protégé.

Lorsqu'un individu quitte un tel espace, toutefois, les services associés à l'espace ne peuvent plus lui être offerts. Ceci entraîne une discontinuité de service, sans qu'il n'y ait eu en contrepartie une discontinuité des besoins. Pour assurer la continuité des services dont dépend l'individu, il importe de faire en sorte que ces services ne soient pas liés à un lieu fixe, et soient plutôt en mesure d'accompagner l'individu dans ses déplacements.

Une convergence de technologies contemporaines ouvre des portes en ce sens : la connectivité accrue des appareils, qui tend vers ce que l'on nomme aujourd'hui l'Internet des objets (IdO); le fait que des ordinateurs grand public mais de format convivial soient désormais fortement répandus; la disponibilité de plus en plus grande de réseaux WiFi; etc. Assurer la continuité de services offerts à un individu mobile, à l'intérieur comme à l'extérieur des espaces intelligents traditionnels, devient véritablement possible.

1.1 Problématique

L'offre matérielle et logicielle existante dans le domaine des espaces intelligents tend à viser la résolution de problèmes concrets, comme la détection de chutes chez les personnes âgées, l'adaptation d'un lieu aux paramètres favorisant le confort d'un individu, ou encore la mise en place d'un suivi de prise de médicaments. Cette offre tend à être circonscrite dans l'espace : on parle typiquement de maisons intelligentes, par exemple, ou d'appartements intelligents.

La réalité contemporaine des individus ayant des besoins particuliers ne se limite toutefois pas à leur logis. Plusieurs se déplacent, vaquent à diverses occupations, travaillent à temps partiel, ont des loisirs. Les besoins des individus sont vastes, diversifiés, et ne s'estompent pas lorsqu'ils franchissent le pas de la porte de leur logis.

C'est pour cette raison que nous avons conçu ContextAA, que nous décrivons par la présente. Tirant son nom des Agents contextualisés (*Context-Aware Agents*) qui y opèrent, ContextAA vise à résoudre le problème de la continuité de service dans un contexte d'usagers mobiles à travers un environnement hétérogène où la qualité de la connectivité peut varier.

1.2 Proposition d'ensemble

Nous proposons donc par le présent document ContextAA, une solution qui se veut générale au problème de l'intelligence ambiante. Par intelligence ambiante, nous entendons d'abord un espace intelligent classique : une enceinte offrant des services aux individus qui s'y trouvent. Nous dénoterons parfois ces environnements par le vocable « espace classique ». Nous enrichissons cette acception classique des espaces intelligents pour en faire un espace sans frontières prédéterminées, dans lequel tout individu et tout agent est présumé capable de mobilité. Nous donnons à cet espace le nom d'espace intelligent ouvert, ou simplement « espace ouvert ».

Réaliser l'espace ouvert est une entreprise ambitieuse, nous le savons, puisqu'elle combine à la fois les défis des espaces intelligents classiques, ceux de la mobilité, et demande de faire certains choix, certains compromis dont nous discutons plus loin. Notre proposition pour réaliser cet objectif repose sur des Agents disséminés dans l'environnement, où chaque Agent doit accomplir une mission. Les Agents et ce sur quoi ils opèrent sont décrits dans un langage commun : le Contexte. L'emploi de majuscules est volontaire pour Contexte et Agent, et distingue l'acception technique de ces termes pour nos travaux du sens plus général généralement accordé à ces termes.

Notre démarche est motivée de prime abord par un constat : bien qu'il soit possible de construire des environnements intelligents capables de prêter assistance de manière très spécialisée à des individus en fonction de leurs besoins, ces espaces sont typiquement délimités, et il se trouve que les besoins d'un individu bénéficiant d'un environnement intelligent ne deviennent pas caducs lorsque l'individu se déplace au-delà des frontières de cet environnement.

L'incapacité d'agents liés à un lieu spécifique d'offrir leurs services à un individu hors de ce lieu entraîne une discontinuité de services pour le bénéficiaire. Nous sommes d'avis qu'il est important, au moins pour certains services, de mettre en place les mécanismes nécessaires pour prévenir cette discontinuité. Pour cette raison, nous mettons de l'avant une démarche maximisant la continuité de service (§4.7.4), que nous quantifions à partir de la capacité qu'a un Agent d'accomplir pleinement sa mission (§3.4.3). En contrepartie, conscients que ce ne sont pas tous les environnements qui pourront offrir à l'usager une gamme de services aussi riche que celle offerte dans un espace intelligent, nous sommes disposés à accepter une fluctuation de la qualité de service à travers le temps et l'espace. Privilégier la continuité de service plutôt que la qualité de service est un compromis que nous acceptons de faire.

Certains services requièrent le support d'un environnement intelligent riche. Pour cette raison, nous positionnons les espaces intelligents classiques comme des zones incluses dans l'espace intelligent ouvert. Nous préconisons qu'un Agent puisse utiliser les services d'un espace intelligent classique lorsque ceux-ci sont disponibles.

1.3 Survol de ContextAA

Nous appuyons notre proposition d'ensemble (§1.2) d'une architecture logicielle conçue pour rencontrer les défis de l'espace intelligent ouvert (§3.5) :

- nous supposons que l'environnement contient des nœuds, qu'il s'agisse d'appareils mobiles, d'ordinateurs fixes, de composants matériels dédiés, … Les technologies et les ressources de ces nœuds sont présumées hétérogènes, en quantité comme en qualité, mais les nœuds participant à notre démarche doivent pouvoir communiquer entre eux. Notez que nous ne supposons pas une connectivité universelle, avec tout nœud et en tout temps; il nous suffit en principe que la connectivité avec chaque nœud soit possible;
- nous déployons sur chaque nœud participant à notre architecture un composant Hôte (§3.5.1), adapté aux ressources qui y sont disponibles;
- des Agents (§3.4) sont déployés sur chaque Hôte. Un Hôte prend en charge les Agents qui y sont déployés et assure la présence d'une gamme minimale de services pour ceux-ci, en particulier l'interaction avec des entités sur d'autres nœuds, incluant des composants de technologies tierces (§1.5.4);
- chaque Agent cherche à accomplir une mission (§3.4). Un Agent connaît les contraintes, matérielles ou autres, auxquelles il est assujetti, et est en mesure de les communiquer à travers les services offerts par son Hôte. Chaque Agent est porteur d'un cadre ontique (§4.8); informellement, le cadre ontique d'un Agent constitue la vision subjective de ce qu'est un Contexte pertinent *pour lui*, donc de ce que sont les Contextes qui lui importent et les relations entre ces derniers;
- un Agent est porteur des opérations qu'il doit réaliser pour accomplir sa mission. L'Agent est pour nous une entité autonome et dynamique, capable de s'adapter à son environnement, mais capable aussi de s'associer à d'autres au besoin dans la poursuite de sa mission. Un Agent peut migrer d'un Hôte à l'autre si cela lui permet de mieux poursuivre sa mission;
- des entités satellites, nommées objets autonomes, gravitent autour d'un Hôte et réalisent les tâches qui ne s'inscrivent pas naturellement dans son cycle d'opérations, par exemple les entrées/sorties bloquantes;
- le Contexte est le langage qui sous-tend notre proposition. Par le Contexte, nous décrivons les états de la réalité enrichie, les comportements des Agents, leur mission, leur cadre ontique, leurs besoins et contraintes en termes de ressources… Le Contexte, jouant à la fois les rôles de langage et de métalangage, se trouve au cœur de notre démarche.

1.4 Survol des principales orientations

Pour tenir compte des réalités matérielles des nœuds, dont les ressources sont susceptibles d'être limitées, la plateforme répartie faite de nos composants Hôtes est légère : presque tout s'y décrit dans un seul langage simple, et chaque Agent y opère selon une perspective locale, dans l'optique de favoriser une action autonomique (au sens de Kephart et Chess [58]) si nécessaire.

Par cette approche, nous adoptons certaines orientations que nous estimons pragmatiques :

- la première est d'encourager la continuité de service (§4.7.4). Tel que mentionné dans §1.5.2, nous acceptons de ne pas pouvoir garantir en tout temps un niveau de qualité de service analogue à celui possible dans un espace classique. Nous privilégions en retour une adaptation des Agents à l'environnement : les services riches d'un environnement intelligent classique peuvent être utilisés par nos Agents lorsque cela s'avère possible. En l'absence de tels services, notre optique est qu'un Agent doit d'abord assurer la continuité de ses services, quitte à réduire la gamme de services offerts à l'ensemble de ceux qui peuvent être assurés dans un espace aux ressources plus limitées;
- la seconde est de préférer un cadre ontique subjectif et une syntaxe commune à une ontologie partagée. Nous supportons d'office des Agents autonomes et mobiles. Par autonomes, nous n'entendons pas solitaires ou isolés, mais bien capables d'action indépendante des autres Agents, et agissant d'abord en fonction de leur propre mission et de leur propre « savoir ». Nous mettons l'accent sur la capacité d'un Agent à demeurer opérationnel même s'il se trouve isolé des autres. Plutôt que d'imposer une ontologie commune aux Agents et de présumer la disponibilité de tiers pour soutenir l'Agent dans son raisonnement, nous amenons une approche subjective, le « Contexte-selon » (§3.6.2).

De par cette seconde orientation, l'Agent devient porteur d'une perspective propre, parfois même singulière, sur le Contexte. Le passage d'une ontologie partagée à un cadre ontique individuel influence notre définition d'un Agent. Pour éviter une cacophonie de perspectives sur le Contexte, nous automatisons la mise en correspondance des besoins d'un Agent, exprimés sous la forme de Contexte, avec le Contexte disponible pour cet Agent (§5.4).

1.5 Justification des choix architecturaux

Notre proposition tient compte de notre compréhension de certains mouvements technologiques contemporains majeurs, et met en relief les solutions que nous apportons à certains problèmes clés de l'espace ouvert tel que nous le concevons.

1.5.1 Diversification et hétérogénéité des espaces intelligents

Les espaces ubiquitaires sont de plus en plus diversifiés, de moins en moins homogènes. En milieu urbain, la variété de marques d'appareils, de technologies, de formats de données, etc. est en croissance soutenue, ce qui s'accompagne de l'émergence de recherches dans ce qu'on nomme les espaces ubiquitaires urbains [120], capables d'appréhender cette nouvelle réalité.

Nous abordons cette diversité et cette hétérogénéité en mettant en place des mécanismes pour maximiser l'adaptation des Agents à leur environnement. Comme le montre la figure 1, un Agent a est isolé du voisinage réseau par son Hôte h (§3.5.2), et ne connaît de l'espace ouvert que le contenu de son propre espace contextuel S_a (§3.4.4). Un Hôte, quant à lui, transige avec les autres Hôtes et avec les entités tierces (§1.5.4) à travers des objets autonomes Obj (§3.5.5); chacun des objets autonomes chargé d'interagir avec une entité tierce joue un rôle d'intermédiaire entre la technologie tierce en question et le monde pur Contexte que nous proposons.

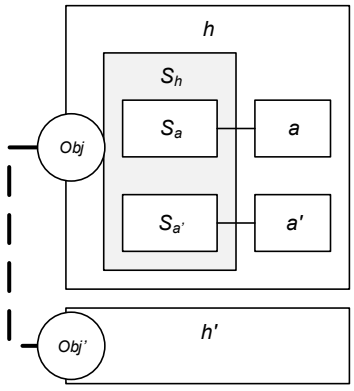


Figure 1 - Isolation d'un Agent du réseau

La diversification des nœuds s'accompagne d'une connectivité croissante avec une multitude d'appareils de la vie courante, ce qui laisse envisager une nouvelle réalité, souvent nommée Internet des objets (*Internet of Things*[1]), que nous abrégerons par IdO : un espace ubiquitaire constitué d'une vaste quantité d'appareils capables de communication. Ces nœuds, aux ressources souvent limitées, sont autant de possibilités pour déployer des Hôtes et loger d'éventuels Agents cherchant à accompagner un individu mobile. Pour être en mesure de profiter même des nœuds aux ressources limitées, notre proposition vise à être peu coûteuse en termes de ressources, comme en font foi nos choix technologiques et notre approche axée sur le seul langage du Contexte.

La croissance de la diversité des nœuds et de leur potentiel de connectivité ne garantit pas qu'un Agent soit en tout temps capable d'interagir avec son environnement. Nous estimons probable qu'un Agent se trouve occasionnellement dans des lieux où la connectivité et imparfaite ou nulle, ou encore là où les nœuds avoisinants ne correspondent pas à ses besoins, peu importe la raison. Conséquemment, la capacité pour un Agent d'agir de manière autonome nous semble être une caractéristique importante d'une conception pour l'espace intelligent ouvert. Notre approche pour la gestion de la discontinuité de connectivité passe par un mécanisme d'isolation, décrit à §5.6.3, à partir duquel la connectivité n'est pas un phénomène directement observable par les Agents.

Même entre Hôtes, donc entre entités opérant sur la base de Contexte, nous préconisons de réduire au minimum les exigences de conformité. Le choix d'appliquer un cadre ontique subjectif à chaque Agent plutôt que d'imposer une ontologie commune à tous les Agents, tout comme le fait de faire reposer l'interopérabilité des Agents sur la syntaxe du Contexte plutôt que sur sa sémantique, découlent de notre acceptation du caractère limité de ce que connaît *a priori* un Agent ou un Hôte : nous mettons l'accent sur la capacité d'apprendre, et d'oublier, du Contexte, de même que sur l'évaluation de la proximité (ou de la distance) sémantique entre deux Contextes.

1.5.2 Connectivité imparfaite et exigence d'autonomie

Un Agent capable de s'adapter à son environnement est, à notre avis, mieux équipé pour faire face à la diversité décrite ci-dessus et en tirer profit. C'est pourquoi nous mettons de l'avant une exigence d'autonomie chez les Agents, exigence que sous-tend notre proposition architecturale dans son ensemble. Dans notre approche, faire en sorte que l'Agent du domaine n'interagisse qu'avec son espace contextuel isole cet Agent des fluctuations du voisinage réseau et permet un raisonnement ne reposant que sur du connu pour cet Agent.

Déplacer le raisonnement vers l'espace contextuel de l'Agent permet de restreindre les impacts d'une connectivité imparfaite. Techniquement, même si son Hôte est incapable d'entrer en communication avec d'autres Hôtes ou avec des entités tierces, l'Agent demeure actif sur la base de son connu jusqu'alors.

[1] Ce terme est attribué à Kevin Ashton, par diverses sources dont http://www.rfidjournal.com/articles/view?4986

Ce choix architectural met en relief l'équilibre entre continuité de service et qualité de service : sur la base des choix que nous avons faits, en particulier celui d'isoler l'Agent du voisinage réseau, un Agent ayant pour mission d'assurer un environnement adéquat à un individu ne cesserait pas de travailler en ce sens s'il est coupé des capteurs lui décrivant l'état « actuel » de l'environnement de cet individu; l'Agent poursuivrait au contraire sa mission, et chercherait à guider les actuateurs en fonction des objectifs visés, tout comme l'Agent sur chaque actuateur chercherait à répondre aux demandes reçues tout en s'assurant que ses paramètres de sécurité soient respectés. Le Contexte-selon et la poursuite par chaque Agent d'une mission qui est la sienne peuvent mener à des conflits, ce dont tient compte notre proposition.

1.5.3 Simplification des mécanismes de raisonnement et de déploiement

Nous faisons le choix de représenter les Agents du domaine, les données et les opérations sur des données par un seul et même format, soit le Contexte. Ceci permet entre autres de raisonner sur le Contexte et de gérer les Agents à partir d'un seul et même moteur, tout comme cela permet de faire migrer un Agent entre deux Hôtes aussi simplement qu'en faisant migrer son espace contextuel.

Cette façon de faire est pour nous une simplification des mécanismes de raisonnement et de déploiement. Nous disons du raisonnement qu'il est simplifié car il ne repose que sur un seul format de données, le Contexte, et ne nécessite pas de cas particuliers pour les données, les Agents, les opérations, etc. Ceci ouvre la porte à une réduction de la taille du processus Hôte dans son ensemble, et au déploiement d'un Hôte sur un nœud aux ressources limitées.

Nous disons du déploiement des Agents qu'il est simplifié du fait qu'outre l'encadrement du processus, qui implique entre autres de détecter les conditions propices à la migration d'un Agent, de repérer une destination adéquate pour ce dernier et de s'assurer que l'Hôte destinataire est disposé à l'accueillir, le transfert en soi de l'Agent est un transfert de Contexte, un problème « réglé » dans notre architecture parce que nécessaire *a priori* à son bon fonctionnement.

1.5.4 Intégration à l'existant

La plupart des cadriciels d'agents répartis et d'entités contextualisées répertoriés à la section §2.4 proposent des infrastructures stratifiées et procèdent par offre et consommation de services, comme le montrent entre autres [29, 42, 101]. En ce sens, notre approche « pur Contexte », qui repose sur l'automatisation de la circulation de Contexte en fonction de la mise en correspondance de Contextes publiés par divers Agents, est un peu hors-normes.

Bien qu'étant hors-normes, notre proposition se doit de permettre l'interaction avec l'existant, avec les technologies tierces qui adressent des problèmes connexes à ceux qui nous intéressent.

En pratique, hors de son propre écosystème pur Contexte, un Hôte est susceptible d'interagir avec :

- des infrastructures existantes, telles que Jade, Jini ou des composants OSGi pour n'en nommer que quelques-unes;
- la plateforme sous-jacente, donc le nœud en tant que tel, typiquement à travers son système d'exploitation. Ceci inclut entre autres l'interaction des Agents avec des capteurs et des actuateurs. Nous en faisons mention ici du fait que les modalités pour permettre l'interaction entre le nœud et les Agents sont semblables à celles permettant l'interaction entre l'Hôte et les technologies tierces; et

- des formats de données, par exemple XML ou JSON, et des protocoles spécialisés tels que FIPA ou Java RMI.

La plupart des technologies permettent aux agents d'exposer et de consommer des services. Les interfaces de ces services sont typiquement compréhensibles par le consommateur, que ce soit par réflexivité ou par analyse d'un descriptif d'interface, WSDL ou autre. Pour assurer une interaction avec les composants de technologies tierces, il faut à tout le moins que notre proposition puisse découvrir des services, assurer l'interopérabilité avec ceux-ci, de même que consommer et produire des données dans des formats autres que le Contexte.

Le Contexte tel que nous le définissons peut en grande partie être converti de manière automatique dans des formats tels que XML ou JSON, et il est souvent possible de faire la transformation inverse (annexe J). Pour que les Agents dans notre modèle demeurent « pur Contexte », et assurer l'homogénéité opératoire de notre modèle, nous réalisons la jonction entre le fonctionnement « pur Contexte » et les technologies tierces en périphérie d'un Hôte, par l'interaction de deux mécanismes :

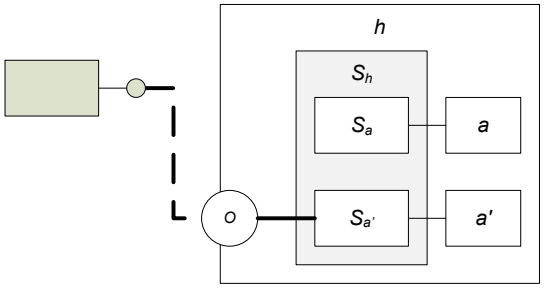


Figure 2 - Interaction avec une technologie tierce

- les Agents standards, par exemple a' dans la figure 2, implémentent les actions qui peuvent se faire sans nuire au cycle d'opération des Agents. Dans ce cas, l'Agent prend sur lui de représenter par du Contexte les états pertinents du nœud sous-jacent et les actions qu'il est possible d'y effectuer, et accepte sous forme de Contexte des demandes d'actions à poser qu'il pourra réaliser par la suite; et
- les objets autonomes, par exemple O dans la figure 2, dans le cas où l'interaction se fait avec un nœud ou un service externe à l'Hôte et au nœud sous-jacent, ou encore lorsque l'interaction demande d'entreprendre des actions qui nuiraient au cycle d'opération des Agents.

Bien que l'automatisation de la génération des interfaces avec des services tiers soit possible, ce volet de ContextAA fait partie de travaux en cours et sera réalisé dans un projet à part entière. Voir l'annexe L pour quelques avenues à explorer en ce sens.

1.6 Motivations d'une démarche novatrice

L'impulsion derrière nos travaux tient à la réalité d'espaces intelligents existants, où nous faisons déjà face au problème de discontinuité de service pour les usagers mobiles et où nous avons noté l'absence de solution générale à ce problème.

De prime abord, nous avons fait le constat que bien qu'il soit possible d'accompagner des individus dans un espace délimité tel qu'un environnement intelligent classique, la mobilité des individus leur permet de ne pas demeurer confinés à un tel espace.

Conscients du fait que les besoins individuels couverts par un environnement intelligent ne disparaissent pas nécessairement chez un individu qui quitte un tel espace, nous avons réfléchi à une solution basée sur des agents mobiles, capables d'accompagner l'individu dans ses déplacements. Nous souhaitions une approche peu coûteuse en termes de ressources, pour exploiter un maximum de nœuds possibles, tout comme nous souhaitions une solution qui ne soit pas rattachée à un seul langage de programmation ou à une seule plateforme.

Un examen des solutions existantes, fait à la section §2.4, a permis de constater qu'il serait préférable de mettre au point notre propre infrastructure, du fait que les plus populaires des infrastructures existantes (en particulier [5, 54]) reposaient sur une machine virtuelle commerciale (la JVM de Java), ce qui est avantageux sur le plan des services offerts *a priori* mais trop coûteux en termes de ressources pour certains des nœuds sur lesquels nous pouvions envisager déployer nos composants. Nous avons plutôt fait le choix de développer un composant Hôte, qui jouera le rôle d'une machine virtuelle sans en être une, et de le faire dans un langage qui donnerait un contrôle plus complet sur la gestion des ressources.

Notre perspective sur ce qui deviendrait l'espace intelligent ouvert impliquait plusieurs *a priori* architecturaux : la connectivité non-garantie et l'accent que nous avons choisis de mettre sur la continuité de services ont permis de mettre de l'avant l'intérêt d'avoir des agents autonomes, peu ou pas dépendants en pratique de tiers spécialisés ou de nœuds clés (§3.4). La continuité de service suggérait aussi l'intérêt pour un Agent d'être en mesure de poursuivre sa mission, donc de migrer d'un substrat à l'autre si les ressources venaient à manquer. L'objectif d'autonomie pour les agents, joint à celui d'une infrastructure peu coûteuse en termes de ressources, nous a amenés à envisager ce qui deviendrait le cadre ontique, sorte d'ontologie propre à un agent, donnant naissance au Contexte-selon qui sous-tend notre démarche.

L'un des aspects où nous estimions pouvoir innover, réduire la consommation de ressources et gagner en flexibilité fut le choix d'un langage spécifiquement pensé pour le Contexte (§4.3). Suite à un examen des catégories traditionnelles du Contexte (temps, lieu, environnement, usager, etc.), et au constat dans la littérature existante de nombreuses applications utilisant des formats spécialisés (§2.2), nous avons fait le choix d'un format minimaliste permettant de maximiser la capacité de représentation du contexte, tout en restant dissocié de la sémantique applicative de cas d'espèces existants. Les travaux sur ce format nous ont permis de constater que la dissociation du langage et de sa sémantique facilitait l'expression d'opérations fécondes, comme la mise en correspondance structurelle et sémantique de Contextes sur la seule base d'une syntaxe commune, ou encore l'automatisation du calcul de la distance entre deux Contextes. Éventuellement, il est devenu clair que le Contexte permettait même de représenter les Agents et leurs opérations, et qu'une conséquence directe de ce constat était que faire migrer un Agent d'un Hôte vers un autre était devenu l'équivalent technique de transmettre un Contexte entre eux.

Les fondements théoriques de nos travaux (chapitre 4) ont évolué côte à côte avec la conception d'un composant Hôte (§3.5.1), lieu où sont déployés les Agents. Nous avons fait progresser les volets conceptuel et technique main dans la main, validant nos idées de manière concrète et assurant du même coup que notre démarche demeurait pragmatique, même pour des systèmes embarqués et aux ressources restreintes.

Une partie de notre proposition nait de réflexions quant au caractère multidisciplinaire de la conception des espaces intelligents. Certains éléments architecturaux tels que les objets autonomes (§3.5.5) et les canaux de Contexte (§3.5.6) visent en partie à simplifier le raisonnement quant au comportement d'un Agent, sans compromettre les caractéristiques opératoires d'un Hôte. Isoler les Agents du réseau et des problèmes associés à la concurrence évacue ces questions de l'élaboration des tâches que les Agents doivent accomplir, ce qui vise à faciliter la mise à contribution de spécialistes non-informaticiens dans la conception d'un Agent.

Le résultat de cette démarche est ce que nous présentons ici :

- le chapitre 2 met en relief l'état des connaissances dans quelques-uns des créneaux connexes à nos travaux dans l'espace intelligent ouvert;
- les chapitres 3, 4 et 5 visent à décrire ce qu'on pourrait qualifier du « cœur » de ContextAA :
  - le chapitre 3 établit l'usage que nous faisons de certains termes clés, en particulier dans l'acception spécifique et technique que leur confèrent nos travaux. Ainsi, ce chapitre montre comment ContextAA permet d'adresser les problèmes de l'espace intelligent ouvert;
  - le chapitre 4 décrit le modèle théorique qui sous-tend notre démarche. Y sont entre autres décrits le langage qu'est le Contexte, les opérations sur le Contexte, la mise en correspondance de Contextes et les Agents de ContextAA;
  - le chapitre 5 détaille les éléments architecturaux par lesquels nous avons réalisé ce modèle, incluant les composants Hôtes, les objets autonomes et les interactions entre Hôtes;
- le chapitre 6 fait état de la démarche de validation en cours pour ContextAA;
- enfin, une conclusion permet de faire le point sur l'état de nos travaux et de décrire les directions vers lesquelles nous sommes engagés pour le proche futur.

Chapitre 2
État des connaissances

« Les grandes connaissances engendrent les grands doutes » (Aristote / Les entretiens – IV^e s. av. J.-C.)

Ce chapitre a pour objectif de situer l'état des connaissances dans les domaines ayant trait aux points d'intérêt de nos travaux, soit les systèmes répartis, les systèmes multi-agents, le Contexte et la contextualisation, de même que les espaces intelligents. Chacun de ces sujets étant vaste, nous limitons notre survol aux parties de ces champs d'étude qui convergent dans une direction connexe à la nôtre, touchant au passage certains créneaux proches du nôtre mais que nous avons choisi de ne pas explorer.

Il deviendra rapidement clair que les environnements intelligents existants offrent des services riches aux individus qui les habitent, mais ont des défauts conséquents avec leurs qualités, c'est-à-dire que la richesse de leur offre de services tient à leur situation, soit celle d'un lieu délimité dans l'espace et dans lequel les outils disponibles sont de grande qualité. Les systèmes pensés pour ces environnements profitent de leurs forces mais ont une portée délimitée par les limites de ces environnements.

Dans cette section, tout en décrivant l'état des connaissances, nous prenons soin de cibler là où nous pensons offrir, par un changement de perspective et une approche technologique différente, des outils qui permettront de transcender les limites des espaces existants pour enrichir l'offre de services qu'ils rendent disponible, tout en poursuivant la collaboration avec leurs services lorsque cela s'avère opportun. Les constats que nous ferons serviront d'assise pour la conception de ContextAA telle que décrit au Chapitre 3.

2.1 Environnements intelligents

Les postes de travail virtuels, mieux connus sous le nom d'ordinateurs personnels aujourd'hui et depuis l'époque où ils furent pensés au Palo Alto Research Center (PARC) comme depuis la commercialisation du premier appareil Macintosh d'Apple, en 1984, proposent aux usagers une représentation interactive d'entités logicielles à travers des métaphores du « monde réel » : des dossiers, qui contiennent des fichiers; une corbeille à recyclage; une interaction basée sur le principe du pointage et de la sélection par un ou plusieurs clics; le glissement d'un document d'un « endroit » à l'autre en le glissant à l'écran… Cette métaphore est d'ailleurs amenée plus loin encore avec les écrans tactiles contemporains qu'on retrouve sur plusieurs appareils contemporains tels que les téléphones intelligents et les tablettes.

Nous utilisons informellement le vocable **espace physique** pour faire référence à l'espace dans lequel vivent les humains. Nous nommons tout aussi informellement **espace logique** ce dans quoi s'exécutent les composants logiciels. Nous ne considérons pas l'un ou l'autre de ces espaces comme « plus réel » que l'autre; en effet, les interactions entre l'espace logique et l'espace physique sont si nombreuses que ces deux espaces sont devenus interdépendants.

Il est possible de voir l'interface entre les humains et la machine comme une entité jouant un double rôle, soit celui de représenter et filtrer l'information d'une part et celui de valider les entrées de données et les opérations entreprises d'autre part. Offrir une métaphore qui adapte le logiciel aux attentes des humains, tout en formalisant les deux extrémités de cette interface, eut un impact considérable sur l'acceptation de la machine par les humains, de même que sur l'avènement de l'ère de l'information [15].

Malgré ce rapprochement, l'espace physique et l'espace logique sont longtemps demeurés séparés, distincts, chacun de son côté de l'interface décrite par les périphériques d'entrée/ sortie que sont la souris, l'écran, les haut-parleurs, etc. Un impact de leur intégration graduelle fut d'accroître fortement la productivité des individus et l'interactivité entre composants de ces deux espaces [27]. Suivant [93], nous nommons **espace intelligent**, ou **réalité enrichie**, l'espace synthétique résultant de la jonction des espaces physique et logique et de leur interaction.

Si la métaphore du bureau de travail virtuel a transformé l'ordinateur en commodité courante accessible aux non-spécialistes, et si les téléphones intelligents, tablettes et autres appareils contemporains ont accéléré ce processus d'acceptation sociale de la technologie, l'informatique ubiquitaire (*Ubiquitous Computing*) et les espaces intelligents poussent plus loin encore la métaphore en cherchant à intégrer les réalités physique et logique sous un angle différent. Plutôt que de proposer une interface à travers des appareils spécialisés comme des ordinateurs, l'informatique ubiquitaire passe plutôt par de nombreux (et souvent petits) appareils disséminés dans l'espace physique, appareils qui ne ressemblent pas nécessairement à des ordinateurs.

Depuis ses débuts, l'un des principaux objectifs de l'informatique ubiquitaire est de faire en sorte que les interfaces manifestes entre les espaces logique et physique disparaissent, se fondent en quelque sorte dans le décor, là où elles demeureront utiles sans par ailleurs perturber les humains dans leurs activités, pour dépasser la métaphore du bureau de travail virtuel [113].

La vision mise de l'avant par Weiser [113] se rapproche de celle de la réalité augmentée : plutôt que de chercher à intégrer les humains dans un espace logique, cette vision préconise d'intégrer les composants du monde logique dans l'espace physique, de manière à les rendre essentiellement invisibles pour les humains qui l'habitent.

Puisque les espaces logique et physique sont distincts mais interdépendants, ceux-ci interagissent à travers une interface. En informatique ubiquitaire, cette interface est constituée d'un ensemble d'appareils, chacun existant en quelque sorte à la fois dans l'espace logique, interagissant avec les autres appareils, et dans l'espace physique, participant à l'environnement des humains. Les capteurs saisissent l'information de l'espace physique et l'exposent au monde logique dans un format susceptible d'y être utile, alors que les actuateurs transforment des requêtes provenant du monde logique en actions sur le monde physique.

Certains prétendent que dans un monde idéal, cette interface entre les mondes physique et logique ne devrait pas être perceptible par des humains, sauf peut-être dans le cas de composants conçus expressément dans cette optique : « *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*. »[2] [113]. En ce sens, l'informatique ubiquitaire est parfois aussi qualifiée de « technologie calme ».

[2] Les technologies les plus riches sont celles qui se fondent dans le décor, au point d'en faire partie. (Traduction libre)

> « *Technologies encalm as they empower our periphery. This happens in two ways. First, […] a calming technology may be one that easily moves from center to periphery and back. Second, a technology may enhance our peripheral reach by bringing more details into the periphery. […] The result of calm technology is to put us at home, in a familiar place.* »[3] [114]

La vision proposée par Weiser est empreinte de défis, et a donné une impulsion à quantité de recherches en informatique ubiquitaire, en particulier dans les créneaux des systèmes intelligents prêtant assistance aux humains de manière réactive ou proactive. Cette vision a bien sûr ses critiques, en particulier ceux qui, à la manière de Chmiel et al. [23], suggèrent que la recherche soit orientée vers l'amplification de l'intelligence humaine plutôt que vers l'intelligence artificielle. À titre d'exemple, selon Rogers [89] :

> « *[some argue] for a significant shift from proactive computing to proactive people; where UbiComp technologies are designed not to do things for people but to […] enable people to do what they want […]. Instead of embedding pervasive computing everywhere in the environment it considers how UbiComp technologies can be created as ensembles or ecologies of resources, that can be mobile and/or fixed, to serve specific purposes and be situated in particular places.* »[4] [89]

En résumé, l'informatique ubiquitaire peut être perçue comme une manière non-intrusive de fondre les espaces physique et logique de manière à ce que l'espace résultant soit une réalité enrichie, accroissant par le fait-même le potentiel de chacun.

La mouvance de l'IdO, devenue populaire pendant que nos travaux sur ContextAA suivaient leur cours, préconise la dissémination dans l'environnement d'objets connectés et chargés d'une forme d'intelligence. Cette mouvance suppose une connectivité pleine et entière des objets, et permet à ces derniers de participer aux tâches quotidiennes des humains : un réfrigérateur qui supervise les dates de péremption des aliments qui y sont entreposés et propose une liste d'épicerie adaptée en conséquence, tenant compte des rabais à l'épicerie de prédilection de ses propriétaires, est un exemple d'un tel objet.

---

[3] Les technologies calment quand elles enrichissent la périphérie de notre environnement, que ce soit parce qu'elles se déplacent aisément entre le centre et la périphérie ou parce qu'elles étendent notre portée en accroissant le détail de la périphérie. Une technologie calme évite les bris de familiarité. (Traduction libre)

[4] Certains préconisent un changement de perspective, passant de l'informatique proactive aux humains proactifs, où les technologies ubiquitaires seraient conçues non pas en fonction de faire des choses pour les gens mais bien en fonction d'aider les gens à faire ce qu'ils souhaitent faire. Plutôt que d'intégrer les composants ubiquitaires partout dans l'environnement, ceux-ci estiment que les technologies ubiquitaires devraient constituer un ensemble de ressources, mobiles ou fixes, jouant des rôles définis et placés à des endroits précis. (Traduction libre)

Dans un survol de l'IdO, Rose [90] relate :

> « *The term "Internet of Things" (IoT) was first used in 1999 by British technology pioneer Kevin Ashton to describe a system in which objects in the physical world could be connected to the Internet by sensors. Ashton coined the term to illustrate the power of connecting Radio-Frequency Identification (RFID) tags used in corporate supply chains to the Internet in order to count and track goods without the need for human intervention. Today, the Internet of Things has become a popular term for describing scenarios in which Internet connectivity and computing capability extend to a variety of objects, devices, sensors, and everyday Items* »[5] [90]

L'IdO est typiquement pensée comme un enrichissement de l'environnement immédiat des humains, et s'inscrit donc dans la conception d'espaces intelligents traditionnels, mais dans une optique de démocratisation de ces espaces : la production accrue d'appareils capables de participer à l'IdO a un impact sur les prix, de telle sorte qu'il est possible de faire l'acquisition d'appareils participant à cette mouvance en ligne ou dans des commerces conventionnels [3]. Comme le mettent de l'avant les travaux de Tao et al. [108], l'IdO est aussi une approche visant à faciliter l'interopérabilité d'appareils *a priori* hétérogènes :

> « *IoT does not refer to a single technology but, instead, to a new paradigm characterized by the pervasive presence around us of a variety of objects (referred as'things') participating into the domestic activities […] connected (sic.) each other [and] able to interact and cooperate in order to achieve common home-related goals* »[6] [108]

Cette démocratisation des espaces intelligents rejoint certains objectifs de ContextAA, qui facilite par design la détection des participants à l'espace intelligent et fait passer les échanges avec ceux-ci par le langage du Contexte. Contrairement à l'IdO, toutefois, ContextAA ne suppose pas une connectivité pleine et entière, et met l'accent sur l'adaptation du système à l'environnement plutôt que sur l'adaptation de l'environnement aux besoins d'un individu. En ce sens, les deux approches sont complémentaires.

---

[5] La paternité du terme "*Internet of Things*" (Internet des objets, IdO) est attribuée à Kevin Ashton en 1999 et visait à décrire un système où les objets du monde physique pourraient être connectés à Internet par voie de capteurs. Ashton utilisait le terme pour illustrer la puissance des étiquettes RFID pour tracer des objets de consommation par Internet sans intervention humaine. Aujourd'hui, IdO est un terme populaire pour décrire des scénarios où la connectivité avec Internet s'étend à une vaste gamme d'objets de la vie courante. (Traduction libre)

[6] L'IdO n'est pas une technologie spécifique mais bien un nouveau paradigme charactérisé par la présence autour de nous d'une variété d'objets (de 'choses') participant aux activités domestiques, interconnectées et capables de collaboration dans la poursuite d'un objectif commun. (Traduction libre)

Une autre mouvance connexe à ContextAA et à l'IdO est celle des environnements ubiquitaires urbains, où la caractéristique première est la diversité technologique et protocolaire [99]. À titre d'exemple de tels environnements, certaines villes se disent désormais « intelligentes », bien que ce qualificatif ait un sens variant fortement selon les lieux [79]. Pour plusieurs villes, cette intelligence vise surtout l'offre de services pour identifier les lieux où il est possible de garer un véhicule ou un signalement aux intéressés de retards possibles dans le réseau de transport en commun. Généralement, les villes intelligentes accroissent l'offre de réseaux sans fils pour accroître la connectivité des individus et faciliter l'accès aux services offerts. S'ajoutent à cela des capteurs pour améliorer la fluidité des déplacements des gens (indications juste-à-temps du trafic, feux de circulation adaptatifs et autres), de la luminosité intelligente dans les rues, une meilleure détection de l'amoncellement de déchets, etc.

Constat 1 : un environnement ubiquitaire urbain se présente comme une offre diversifiée (certains diraient disparate) de services. L'évolution des villes qui se veulent de plus en plus intelligentes annonce un accroissement de la présence de ces environnements.

Les espaces intelligents traditionnels sont typiquement des environnements riches en services mais délimités dans l'espace : « *Intelligent Space […] denotes an environment containing a set of interacting Context providers and consumers* »[7] [93]. La richesse de leur offre de services fait de ces espaces un milieu privilégié d'expérimentation et de développement de solutions visant en particulier l'amélioration des conditions de vie des individus. Ils ne sont toutefois pas les seuls types d'environnements intelligents possibles en pratique.

Des études ont été faites sur les **espaces intelligents nomadiques**, ensembles fédérés d'espaces intelligents entre lesquels se déplace un usager, et ont de ce fait abordé la question du transfert de données d'un espace intelligent à l'autre. Yang [117] offre un plaidoyer pour assurer une continuité de services pour des usagers mobiles, et soutenant cette approche :

> « *As people get used to new technology, they come to expect its availability, and dependency on the technology grows. This is especially true of smart spaces because their goal is to embed technology into living environments, which can greatly impact many daily activity routines. Consider an elder person with early-stage Alzheimer, whose independent living depends on a cognitive assistance service that cues her to finish tasks she has started. When she leaves her apartment to go to the grocery store, she still counts on receiving the same kind of help. This is why we must have nomadic pervasive computing, so that critical services are not lost just because the user leaves her home space* »[8] [117]

[7] L'espace intelligent identifie un environnement contenant un ensemble de producteurs et de consommateurs de Contexte en interaction les uns avec les autres (Traduction libre)

[8] En s'habituant aux nouvelles technologies, les gens en viennent à dépendre de leur disponibilité. Ceci est particulièrement vrai des espaces intelligents, ce qui impacte plusieurs activités du quotidien. Prenons pour exemple une personne souffrant de la maladie d'Alzheimer, qui dépend de l'assistance de son environnement pour terminer les tâches qu'elle entreprend; cette personne a encore besoin d'aide lorsqu'elle fait son épicerie. C'est ce qui explique le besoin d'espaces intelligents nomadiques (Traduction libre)

La vision nomadique des espaces intelligents vise à assurer une transition du profil usager (§2.2.3) d'un espace intelligent à l'autre pour un usager en déplacement, avec les échanges d'information que comprend un comportement nomadique [12]. Bien que cela constitue un pas en avant dans l'accroissement de la liberté de mouvement des usagers, le support qui leur est apporté par l'environnement demeure discontinu, n'étant pas offert pendant le déplacement.

La principale limite des espaces intelligents traditionnels est en effet leur caractère fini, délimité, qui tend à constituer un obstacle pour ce qui est d'assurer la continuité des services qui y sont offerts. Notre démarche prend racine dans le souci d'assurer cette continuité de services pour les composants logiciels et les individus mobiles, capables de franchir ces frontières.

Constat 2a : il existe des solutions spécifiques, pensées pour les espaces intelligents traditionnels, tout comme il existe des solutions pour des espaces dits « nomadiques » qui visent à assurer une préservation du profil usager entre deux espaces intelligents traditionnels. Assurer la continuité de services lors de déplacements entre deux espaces intelligents traditionnels demeure problématique.

> « *We are moving from the Personal Computer age (i.e., a one computing device per person) to the Ubiquitous Computing age in which a user utilizes, at the same time, several electronic platforms through which he can access all the required information whenever and wherever needed* »[9] [21], en référence à [113]

En comparaison avec les espaces intelligents traditionnels, qui existent depuis des décennies, l'espace intelligent ouvert est une préoccupation relativement récente, à propos de laquelle la littérature est moins touffue. On rencontre entre autres : une prise de position dans Zalavsky [119] portant sur l'importance des agents mobiles dans un espace ouvert; une discussion dans Jayaputera [55] sur des agents mobiles réalisant individuellement des tâches simples qui, prises ensemble, constituent des missions plus complexes; et Pratistha et al. [83] qui discute de l'intégration d'agents mobiles et de services Web. Plus récemment, Bosse [10] met de l'avant une infrastructure multi-agents autonomes pour des appareils munis de fureteurs Internet, solution toutefois plus lourde que celle que nous mettons de l'avant avec ContextAA. La solution proposée par Bosse est stratifiée : pour atteindre des objectifs semblables à ceux de ContextAA, elle utilise pour chaque nœud une base de données (espace peuplé d'uplets), un support intrinsèque du protocole http, une couche RPC, une couche isolant le système de fichiers du nœud hôte, une couche DNS, etc. alors que ContextAA rend la plupart de ces composants optionnels à l'aide d'objets autonomes.

Nous ne sommes pas seuls à constater le besoin pour un espace intelligent ouvert. Prenons par exemple la question de la tolérance aux pannes dans un système réparti et mobile :

[9] Nous quittons l'ère de l'ordinateur personnel, où l'on trouve un ordinateur par personne, pour entrer dans celle de l'informatique ubiquitaire, où chaque individu accède en tout temps à l'information requise par le truchement de plusieurs appareils (Traduction libre)

> « *[…] due to variable nature of the hosting and network environments (both intranet and internet) executing the distributed applications, changing their topology could overcome both the resulting performance and reliability shortcomings. Examples of these shortcomings include system failure, insufficient resources, and network disruptions. By supporting parts of the application to relocate to a different host, some of the performance and reliability problems could be solved either temporarily or permanently* »[10] [83]

Les espaces intelligents traditionnels peuvent être considérés comme des espaces contextualisés « macro », où le système est examiné comme un tout, ou « micro », où chaque Agent approche le Contexte de manière individuelle et « subjective ». Une discussion des forces, faiblesses et contextes d'application de ces deux stratégies apparaît dans Abdulrazak et al. [2].

Les environnements intelligents, tout comme les technologies qui participent à la conception de ces environnements, résultent des travaux de plusieurs laboratoires de recherche. Certains développent leurs propres environnements, d'autres contribuent à l'enrichissement technologique des laboratoires existants. Un survol rapide de quelques-uns des lieux de recherche plus connus parmi ceux-ci suit, accompagné d'une brève explication du positionnement de ContextAA en lien eux.

Laboratoire	**Positionnement de ContextAA**
Au MIT se trouve le laboratoire AIRE[11], pour Agent-based Intelligent Reactive Environments, qui comprend entre autres l'Intelligent Room[12], un environnement ubiquitaire nomadique où sont mis au point des outils de collaboration entre humains et de nouvelles métaphores d'interfaces personne/ machine. AIRE fait partie d'un projet de plus grande envergure nommé Oxygen[13], où l'accent est mis sur l'humain d'abord, et ensuite seulement l'environnement dans lequel l'humain est immergé. La reconnaissance vocale semble y occuper une place de choix dans les interactions entre l'humain et ce qui l'entoure.	Dès le début, ContextAA a été pensé de manière à pouvoir accompagner des humains, mais n'a jamais été conçu dans ce seul objectif; à titre d'exemple, le Profil, dont nous discutons sommairement à la section §2.2.3, est dans ContextAA un Contexte parmi d'autres, du moins sur le plan technique. Pour cette raison, un système comme ContextAA pourrait être utile à titre de plateforme soutenant des travaux comme ceux de ce laboratoire, mais ce serait pour nous un domaine d'application.

[10] En changeant leur topologie, les applications réparties peuvent compenser des déficiences de stabilité et compenser pour l'instabilité du voisinage réseau. La capacité de relocaliser des parties d'une application vers un autre hôte peut compenser ou résoudre ces problèmes de stabilité. (Traduction libre)

[11] http://www.ai.mit.edu/projects/aire.orig/

[12] http://people.csail.mit.edu/brooks/papers/aizu.pdf

[13] http://oxygen.csail.mit.edu/Overview.html

Laboratoire	Positionnement de ContextAA
Microsoft Research comprend un groupe de recherche nommé *Adaptive Systems and Interaction*, ou ASI, qui met l'accent sur l'interaction et la visualisation de données adaptées au contexte. Ce groupe s'intéresse entre autres au « *[…] long-term dream of fluid conversation between people and computers* »[14] et au développement de capteurs et d'actuateurs novateurs. Une partie de leurs travaux porte sur le diagnostic et la réparation automatiques, un sujet en lien direct avec les agents autonomiques de Kephart et Chess [58].	Pour ContextAA, la visualisation de données est un domaine d'application. La manière la plus probable d'intégrer un tel volet à ContextAA serait de créer une application transigeant du Contexte avec un objet autonome et utilisant en quelque sorte un Hôte de ContextAA à titre de serveur de Contexte. En retour, en intégrant l'auto-diagnostic des Agents en continu, ContextAA se prête bien au développement d'Agent autonomiques; il s'agit d'un objectif que ContextAA partage avec les projets de ce laboratoire.
À Telecom SudParis, le groupe Ambient Intelligence & Pervasive Systems[15] présente le contexte comme émergeant de « traces de pas » numériques, qu'ils collectent en vastes quantités et étudient pour être en mesure de les mettre en relation. Les environnements intelligents qu'ils étudient incluent des entités mobiles, par exemple un taxi, et les environnements urbains.	Un système comme ContextAA, où l'échange de Contextes entre Hôtes sur la base des besoins et des missions des Agents, pourrait selon nous server à titre de plateforme de développement et de déploiement pour les projets de ce laboratoire.
L'Autonomous and Intelligent Systems Group, du City University de Londres[16], où se fait de la recherche sur les environnements intelligents, les agents et l'apprentissage automatique, avec un intérêt particulier pour le créneau de l'auto-optimisation telle que mentionnée par Kephart et Chess [58] comme condition de l'autonomie des agents. Le volet « environnement intelligent » de la recherche qui s'y fait a comme particularité le recours au jeu et aux sociétés artificielles comme métaphore des interactions entre individus.	Bien que ContextAA soit agnostique quant aux champs d'application, il y a recoupement entre les efforts d'auto-optimisation de ce laboratoire et la dynamique d'entretien de l'espace contextuel sur la base de méta-Contextes dans ContextAA.

[14] […] rêve d'une conversation fluide entre l'humain et l'ordinateur (Traduction libre), tiré de http://research.microsoft.com/en-us/groups/adapt/

[15] http://www-public.it-sudparis.eu/~zhang_da/

[16] http://www.city.ac.uk/department-computer-science/ais-group

Laboratoire	Positionnement de ContextAA
Le laboratoire A☆STAR[17], à Singapour, où s'accomplit de la recherche portant sur la contextualisation sous plusieurs formes, allant de la voiture intelligente aux panneaux publicitaires contextualisés, en passant par les vêtements intelligents. Ce laboratoire met l'accent non pas sur les environnements intelligents en tant que tels, mais plutôt sur des technologies susceptibles de participer à l'intelligence d'environnements développés par des tiers.	ContextAA pourrait mainfestement servir à titre de plateforme de développement et de déploiement pour les projets d'un laboratoire tel que celui-ci.
À Georgia Tech, le projet AHRI, pour *Aware Home Research Initiative*[18], s'intéresse tout particulièrement à la question de la santé des individus dans un environnement intelligent. Prendre le virage vers la mobilité y est présenté comme une avenue envisagée, pas une réalité du moment. L'une des préoccupations périphériques de l'approche des gens qui y œuvrent est l'économie d'énergie et de ressources de manière générale.	ContextAA pourrait aussi servir à titre de plateforme de développement et de déploiement dans un laboratoire comme celui-ci. En particulier, la mobilité est inhérente à notre modèle.
Le projet CAMP, pour *Context Aware Mobility Project*[19], est mené par le centre de recherche d'*Information Communications Technology*, en Australie. Leur principale préoccupation semble être l'interconnexion automatique des individus par les meilleurs canaux possibles en fonction de la tâche qu'ils cherchent à accomplir.	Nous sommes d'avis que ce que ContextAA fait en faisant circuler les requêtes et les réponses de Contexte entre les Hôtes pourrait servir à titre de soutien à des travaux comme ceux de ce projet.

[17] http://www.i2r.a-star.edu.sg/

[18] http://awarehome.imtc.gatech.edu/

[19] http://nicta.com.au/research/archive/business_areas/mobile_systems_and_services/context_aware_mobility_project

Laboratoire	Positionnement de ContextAA
À l'Université Essex d'Angleterre, le *Computational Intelligence Centre*[20] mène des projets sur les agents intelligents, l'intelligence ambiante et l'apprentissage automatique. Leurs travaux touchent aux environnements intelligents dans une acception large, comprenant entre autres les moteurs de navires et les appareils sous-marins.	Nous pensons que ContextAA pourrait servir dans une situation comme celle-ci, les sujets d'étude se rejoignant, mais ContextAA est un projet d'ordre général, voué à soutenir le fonctionnement des applications, alors que ceux de ce laboratoire visent à résoudre des problèmes plus spécifiques.
Le laboratoire DOMUS, de l'Université de Sherbrooke, fait d'un appartement complet à l'intérieur des murs de l'institution et où une partie importante de la recherche touche les questions de la santé et du vieillissement. Ses objectifs sont de « mettre en place les fondements théoriques, logiciels et matériels pour la conception et la réalisation de services mobiles et diffus »[21]. Parmi les créneaux de recherche qui y sont poursuivis, on trouve le déploiement de composants dans des environnements diffus, l'expérimentation de prototypes novateurs en grandeur nature d'environnements diffus et d'applications mobiles, et l'étude multidisciplinaire des impacts de l'informatique diffuse et de l'informatique mobile sur la vie quotidienne et la société.	ContextAA contribue à certains projets de DOMUS, entre autres Ponce et al. [81, 82]
Le Gator Tech Smart House[22], de l'Université de la Floride, dont la recherche vise à maximiser l'indépendance des personnes âgées ou souffrant d'un handicap, et le maintien d'une bonne qualité de vie pour elles. Ce laboratoire met l'accent sur le développement d'un support aux tâches quotidiennes des individus par les services de l'environnement intelligent.	Il y a des similitudes entre ce laboratoire et DOMUS. ContextAA pourrait sans doute servir au développement d'applications à cet endroit.

[20] http://cic.essex.ac.uk/

[21] http://www.usherbrooke.ca/sciences/recherche/centres/informatique/domus-domotique-et-informatique-mobile/

[22] http://www.icta.ufl.edu/gt.htm

Laboratoire	Positionnement de ContextAA
À l'Université de l'Illinois, GAIA[23] est décrit comme un méta système d'exploitation sur lequel reposent des environnements intelligents programmables, qu'ils nomment « *Active Spaces* ». Ce logiciel est déployé dans un environnement intelligent haut-de-gamme, mais peut, selon ses développeurs, être déployé dans des environnements plus diversifiés.	Contrairement à ce que nous mettons de l'avant avec ContextAA, le projet GAIA mise sur la qualité de service, mais se prête à un espace aux frontières définies *a priori*. Les espaces nomadiques forment habituellement un graphe dont chaque noeud est un espace intelligent; ContextAA s'occupe aussi de ce qui se passe sur les arêtes de ces grapes.
Le *Mobile & Pervasive Computing Group*[24], ou MPC, situé à l'Université du Texas, œuvre sur des questions telles que les environnements intelligents et les applications mobiles opportunistes. Leur préoccupation première porte sur les applications de la contextualisation et du *situation-awareness* à la conception d'applications s'adaptant à l'environnement et aux besoins individuels. Plusieurs de leurs publications, par exemple Maxfield et Julien [66] ou Roy et al. [92], portent sur le problème de la recherche d'information sur la base de l'information disponible en un lieu et à un moment donnés.	L'approche basée sur le Contexte-selon de ContextAA serait, selon nous, intéressante à appliquer dans le contexte des travaux de ce laboratoire. La mobilité y est inhérente, et le volet opportuniste pourrait se valider sur une base Agent par Agent.
UCL, pour *Ubiquitous Computing Laboratory*[25], de Kyung Hee University en Corée du Sud, se préoccupe à la fois de systèmes ubiquitaires, d'applications mobiles, de télésanté et de ce qu'on nomme communément le *Big Data*.	ContextAA a été pensé à la fois pour couvrir les exigences de nœuds à faibles ressources et pour profiter des capacités de nœuds plus riches. Rien de notre modèle ne semble *a priori* s'opposer au traitement de *Big Data*, mais ce n'est pas un champ d'application que nous avons exploré jusqu'ici.

[23] http://gaia.cs.uiuc.edu/

[24] http://mpc.ece.utexas.edu/

[25] http://uclab.khu.ac.kr/

Ces environnements partagent un certain nombre de caractéristiques : ils tendent à appuyer l'humain dans ses activités quotidiennes ou à en enrichir les possibilités, mais ils ont pour la plupart des frontières au-delà desquelles il devient difficile d'offrir les services de l'environnement à ses usagers. Dans certains cas, les frontières sont fixes mais l'environnement intelligent est mobile (p. ex. : un navire, un taxi). Les villes intelligentes prises dans une acception élargie prennent forme, par exemple avec l'avènement de projets comme Digital Skin décrit par Rabari et Storper [84] qui vise toutefois en priorité la collecte et l'analyse des données sur la ville et ses habitant, et moins à offrir des services à ces derniers; pour ce qui est d'appréhender l'espace intelligent ouvert dans son ensemble, il nous semble que le terrain demeure vierge en grande partie.

ContextAA vise à résoudre des problèmes connexes à ceux auxquels s'attaquent les travaux de ces divers laboratoires, mais avec une approche distinctive : indépendance *a priori* du substrat matériel, faible coût en ressources, peu de dépendances envers des logiciels tiers, le tout en mettant l'accent sur la continuité de service plutôt que sur la qualité de service.En ce sens, ContextAA pourrait interagir avec les infrastructures de plusieurs des laboratoires susmentionnés. Sans mettre l'accent sur un contexte d'exécution spécifique, comme le font par exemple AIRE (interaction avec les humains), l'Autonomous and Intelligent Systems Group (métaphore axée sur le jeu et les sociétés artificielles) ou le Gator Tech Smart House (bien-être des personnes ¸âgées), et sans viser une famille de cas d'applications spécifique comme le font l'Ambient Intelligence & Pervasive Systems ou UCL (étude de données massives), ContextAA se veut une solution générale infrastructurelle à partir de laquelle des outils plus spécifiques peuvent être construits, mais en visant la continuité de service plutôt que la qualité de service comme le fait par exemple GAIA.

Constat 2b : bien qu'il existe plusieurs solutions à des problèmes associés aux espaces intelligents classiques, et bien qu'il existe quelques solutions de type « nomadique » (voir plus haut dans cette section) au problème de la migration des individus entre deux espaces intelligents classiques, sur il n'existe pas véritablement de solution appréhendant l'espace intelligent ouvert.

2.2 Contexte

Les environnements intelligents ne sont pas qu'un assemblage d'appareils et de composants logiciels entourant un individu; pour qu'il soit vraiment réalisé, l'environnement intelligent doit devenir un environnement ubiquitaire, où les mondes logique et physique s'intègrent et se fondent l'un dans l'autre.

Pour en arriver à une telle intégration, à tout le moins du point de vue des composants de l'espace logique, il ne suffit pas d'y assurer l'accès aux informations décrivant l'espace physique. Il faut aussi faire en sorte que les composants de l'espace logique puissent faire face à une description de l'espace physique qui soit :

- riche de données chargées sémantiquement, et qu'il importe de traiter en tenant compte de la situation dans laquelle elles ont été obtenues, produites, des raisons pour lesquelles elles ont été obtenues, du savoir *a priori* du composant consommateur, etc.; et
- susceptible de changer à tout moment, dû à des circonstances sur lesquelles le composant consommateur de ces données n'a que peu ou pas de contrôle.

En effet, les états de l'espace physique changent dû aux circonstances comme aux actions de ses habitants, incluant les actuateurs. Ainsi, un composant de l'espace logique doit être capable de réagir aux circonstances et de s'adapter au changement. De plus, les composants de l'espace logique influencent de plus en plus les habitants de l'espace physique; à titre d'exemple, pensons seulement à des services d'assistance aux conducteurs de véhicules automobiles [85] ou tout encore à des véhicules qui se conduisent eux-mêmes, comme le *Google Self-Driving Car*[26]. Le domaine que constitue la représentation de l'espace physique est immense, et doit être abordé en conséquence, comme l'explique Schilit et al. :

> « *One challenge of mobile distributed computing is to exploit the changing environment with a new class of applications that are aware of the context in which they are run. Such context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time. A system with these capabilities can examine the computing environment and react to changes to the environment* »[27] [100]

Lorsque des données sont acquises de l'espace physique et introduites, parfois avec transformations, dans l'espace logique pour participer à la réalité enrichie, le fruit de ces transformations et du raisonnement sur les données résultantes est ce que Winograd [115] nomme **contexte** :

> « *Context is an operational term: Something is context because of the way it is used in interpretation, not due to its inherent properties. The voltage on the power lines is a context if there is some action by the user and/or computer whose interpretation is dependent on it, but otherwise is just part of the environment* »[28] [115]

Nous écrivons **Contexte** avec un 'C' majuscule pour référer à notre acception technique de cette entité au centre de nos travaux, et pour distinguer cette écriture de celle utilisée pour faire référence au « contexte » dans son acception générale. Ce choix orthographique que nous faisons n'est pas une convention universelle, mais il nous permet de lever certaines ambiguïtés sémantiques. Bien que ce choix soit propre à notre démarche, nous l'appliquons également dans ce survol de l'état de la connaissance dans le but de clarifier le propos.

---

[26] https://plus.google.com/+GoogleSelfDrivingCars/videos

[27] Un défi de l'informatique mobile repartie est de tirer profit d'un environnement changeant avec une nouvelle famille d'applications, conscientes du contexte dans lequel elles sont exécutées. Un logiciel ainsi contextualisé s'adapte au lieu d'utilisation, à l'ensemble des gens à proximité, des appareils qui lui sont accessibles, de même qu'aux changements à des choses au fil du temps. Un système avec ces caractéristiques peut constater les changements dans son environnement, et réagir en conséquence. (Traduction libre)

[28] Le contexte est un terme opératoire : une chose est du contexte de par l'utilisation qui en est faite pour fins interprétatives plutôt que dû à des propriétés inhérentes. Par exemple, le voltage sur une ligne d'alimentation électrique est du contexte si une action doit être posée sur la base de son interprétation, sinon cette information fait simplement partie de l'environnement. (Traduction libre)

La définition du contexte donnée par Winograd ci-dessus pose la question de la dichotomie entre le contexte comme representation du monde et la perception ou l'interprétation qui en est faite. Avec ContextAA, le Contexte est représentation par la médiation des capteurs ou par celle d'Agents, consommé sur la base subjective du Contexte-selon. La representation par voie de Contexte est subjective, étant réalisée par un Agent, et il en va de même pour ses choix de consommation e Contexte.

ContextAA propose le Contexte-selon, un Contexte dont la sémantique n'est pas intrinsèque, dependant plutôt des Agents qui le produisent et le consomment. Le Contexte dans ContextAA est donc neutre sur la question de la dualité entre représentation et perception; ce sont les Agents qui, sur une base individuelle, proposent par le Contexte-selon une réponse qui leur est propre. Ce qui tient lieu de perception dans ContextAA, avec le Contexte-selon, est le traitement fait par un Agent sur le Contexe mis à sa disposition à partir des règles subjectives logées dans son candre ontique.

Le Contexte est souvent décrit comme étant une représentation opérationnelle de l'espace physique, mais il n'y a pas de contrainte inhérente au Contexte qui empêcherait de l'utiliser pour décrire l'espace logique, donc de servir à titre d'outil opératoire pour décrire les éléments de la réalité enrichie dans son ensemble. C'est d'ailleurs ce que nous faisons avec ContextAA.

Le Contexte peut être considéré comme du Contexte brut (*Raw*) [59], fruit d'un enrichissement sémantique appliqué sur des données brutes (p. ex. : transformer la valeur numérique résultant de la lecture d'un capteur en une température exprimée en °C), ou comme du Contexte synthétique, construit par agrégation de Contextes, raisonnement sur du Contexte ou transformations appliquées à du Contexte, ce que Henricksen et al. [42] nomme *Context Processing* et exprime en termes de sources, drains (*Sinks*) et canaux.

2.2.1 Définitions

Il est difficile d'en arriver à une définition formelle du Contexte, puisque les définitions proposées dans la littérature varient sensiblement selon les sources. Ces variations tiennent à plusieurs aspects.

Certains considèrent le Contexte comme étant une propriété inhérente de l'environnement logique ou physique, alors que d'autres estiment qu'il s'agit d'une propriété émergente de ces environnements, prenant par exemple racine dans l'agrégation de Contextes bruts ou dans la production de Contexte synthétique. À titre d'illustration, dans un article portant principalement sur les capteurs de lieu (*Location Sensors*), [46] place les capteurs dans trois groupes : les capteurs physiques, qui saisissent des données sur le monde physique; les capteurs virtuels, qui saisissent des données sur le monde logique, et les capteurs logiques, qui combinent les données prises de capteurs des deux autres groupes.

La constitution du Contexte varie aussi selon les sources. Des exemples typiques du Contexte dans la littérature incluent :

- les propriétés de l'environnement physique, que ce soit celui d'un agent ou d'un humain. Parmi les plus populaires de ces propriétés, on trouve les catégories « traditionnelles » de Contexte que sont le temps, le lieu, la proximité (avec quelque chose ou avec quelqu'un) et l'activité humaine. La majorité des définitions du Contexte admettent aussi des caractéristiques environnementales telles que la température, l'éclairage, l'humidité ambiante, etc. Un tel Contexte peut être inhérent ou émergent, comme dans le cas du constat qu'une porte a été ouverte par l'action d'un individu, ou celui qu'une personne semble en train de se cuisiner un petit plat sur la base des actes que cette personne a posés. Le Contexte de ContextAA, de manière inhérente, ne se limite donc pas à la description d'un constat d'état sur le monde et sa situation (§4.3 explore cette question en détail);
- les propriétés de l'environnement logique, comme par exemple le Profil (§2.2.3), l'identité d'un appareil, ou encore le Contexte synthétique produit par des agrégateurs ou d'autres composants. Un tel Contexte peut être émergent, par définition, mais peut aussi être inhérent, comme dans le cas de l'examen de la durée de vie résiduelle de la pile d'un appareil.

Qu'il décrive une réalité logique, physique ou un hybride synthétique de ces deux catégories, le Contexte apparaît comme une forme de métadonnée sur la réalité enrichie. La définition du Contexte qui semble la plus connue et la plus citée est celle de [30], une définition en deux temps :

> « *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. […] A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task* »[29]

Cette définition a plusieurs qualités, étant flexible et relativement ouverte, mais elle met l'accent sur l'usager en positionnant la pertinence du Contexte en fonction des tâches que ce dernier doit accomplir. Il faut toutefois souligner que Dey et Abowd [30], dans ses exemples, montre le Contexte aussi comme dépendant des composant logiciels, s'écartant de manière pragmatique de sa définition au sens strict. Cette acception du Contexte est sans doute la plus répandue, et certains textes tels que Nawrocki et al. [70] placent les services offerts aux usagers au cœur du Contexte sans même expliquer pourquoi, tenant cette acception du terme comme implicite.

Constat 3 : une acception possible (et répandue) du Contexte place l'humain au centre de ce qui en détermine la pertinence. Ceci peut s'avérer moins à propos pour certaines applications où le rôle de l'humain est plutôt périphérique que central.

Il s'avère que [29] propose aussi une catégorisation du Contexte en quatre groupes, soit le lieu, le temps, les états/ l'activité, et l'identité. Les deux premiers situent le Contexte, le troisième décrit son utilité, et le quatrième permet de distinguer les Contextes les uns des autres.

Constat 4 : il existe des efforts de catégorisation *a priori* du Contexte. Ces catégorisations simplifient la définition de certains problèmes spécifiques, et aident par le fait-même à la mise en place de solutions pour ceux-ci.

---

[29] Un système est contextualisé s'il utilise le contexte pour offrir de l'information ou des services pertinents à l'usager. La pertinence a trait aux tâches de l'usager. (Traduction libre)

Notez que même dans le cas de Contextes comme le temps et l'espace, qui surviennent dans la majorité des systèmes contextualisés (§3.4.6) et sont donc connus et traités de manière relativement routinière, il existe de nombreuses variations dans la forme et le sens que prend le Contexte. Des représentations absolues dans un référentiel donné (coordonnées euclidiennes ou polaires; dates de calendrier; temps d'horloge; etc.) ou relatives (près de, à gauche de, en-dessous de, entre, …) sont aussi susceptibles l'une que l'autre d'être valides et pertinentes à titre de Contexte.

Nous lisons dans cette variation de forme selon le contexte un besoin pour le Contexte d'être flexible et capable d'adaptation. En particulier, pour un Agent mobile, la capacité de réaliser certaines transformations de référentiel de manière automatique (§4.5.3) et d'adapter dynamiquement le raisonnement de l'Agent aux règles qui prévalent dans son milieu nous semblent des *a priori* pour des systèmes contextualisés dans l'espace intelligent ouvert.

La contextualisation est décrite par Ryan et al. [97] comme :

> « *[…] a term that describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user identity. This information can be used not only to tag information as it is collected in the field, but also to enable selective responses such as triggering alarms or retrieving information relevant to the task at hand.* »

Cette définition est aussi en deux parties : la première décrit le Contexte en soi, alors que la seconde balise ce que les systèmes contextualisés peuvent être. Le critère de l'utilité du Contexte pour une tâche est présent ici aussi, mais sans toutefois être associée à la tâche accomplie par un usager.

La définition du Contexte proposée par le laboratoire Technology for Enabling Awareness [109] va comme suit :

> « *The notion of Context-awareness for devices itself can be split up into three components: activity, environment and self. The activity describes the task the user is performing at the moment, or more generally what his or her behaviour is. […] The environment describes what the status is of the physical and social surroundings of the user. […] Finally, the self component contains the status of the device itself. This third point of view on Context-awareness has not been researched as much as the other two, but is a very interesting one in the scope of cognitive sciences.* »

L'introduction d'un « soi » pour l'Agent dans la définition du Contexte et de la contextualisation soulève des questions en apparence anthropomorphique : « qui » est contextualisé? Pris au sens anglais de *Context-Aware*, ou conscient du Contexte, nous écririons « Qui est conscient? », « conscient dans quel sens? », ou encore « conscient de quoi? Ces considérations sont au cœur de ContextAA et s'expriment par ce que nous nommons le Contexte-selon.

Pour sa part, Fujii [36] décrit le Contexte comme suit :

> « *[…] a system obtains information from numerous sensors arranged in the home, workplace, cities, and other spaces where people engage in everyday activities. This is used to determine the details of the user's actions and the constantly-changing condition of the environment, that is, "context."* »[30].

Les actions des usagers sont au cœur de cette définition du Contexte; d'ailleurs, les mêmes auteurs catégorisent le Contexte en *Resource Context*, en lien avec des images ou des clips vidéo par exemple; *User Context*, qui a trait aux intentions de l'usager, à son comportement; et *Situational Context*, décrivant les conditions de l'environnement de l'usager. Le « *User Context* » de Fujii est ce que plusieurs nomment le profil usager, que nous nommons simplement Profil.

La définition de Neovius et al. [71] est plus succincte : « *[…] we interpret context as a setting in which an event occurs […]* »[31], liant de fait contexte et occurrence d'événement – un « Contexte-où-et-pourquoi ». Le choix du terme *setting* est intéressant, étant ouvert à l'inclusion des usagers, conditions environnementales, Agents, etc. Un glissement sémantique apparaît toutefois, car le Contexte n'est pas ici un élément d'un ensemble déterminé par l'environnement, mais bien quelque chose de plus englobant. D'autres comme Olaru et al. [73] font de même et donnent une définition courte mais large du Contexte : « *Context is defined as the interrelated conditions in which something exists or occurs* »[32], pour ajouter ensuite une précision qui rejoint les acceptions anthropocentriques plus répandues : « *In the case of Ambient Intelligence, [context] denotes those elements that affect the interaction between the user and the system* »[33].

Constat 5 : une acception possible du Contexte tient au lien entre l'occurrence d'un événement et la situation dans laquelle cette occurrence survient.

Pour Chen et Kotz [18], le Contexte est : « *the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user* »[34]. Cette définition insiste d'abord sur l'impact du Contexte sur le système contextualisé, en ensuite sur l'usager. Les Contextes passifs, pertinents sans être critiques pour une application, y sont considérés distincts des Contextes actifs, qui influencent le comportement d'une application. La question de savoir s'il est possible qu'un Contexte soit à la fois passif pour un Agent et actif pour un autre n'est pas adressée par l'auteur.

---

[30] […] un système obtient de l'information de plusieurs capteurs disposés partout où les gens sont actifs. Ceci permet de déterminer le détail des actions individuelles et les conditions continuellement changeantes de l'environnement, c'est-à-dire « le contexte » (Traduction libre)

[31] Nous définissons le contexte comme le cadre où se produit un événement […] (Traduction libre)

[32] Le contexte est défini comme les conditions interreliées dans lesquelles quelque chose existe ou se produit. (Traduction libre)

[33] Dans le cas de l'intelligence ambiante, [le contexte] dénote les éléments qui impactent l'interaction entre l'usager et le système. (Traduction libre)

[34] L'ensemble d'états environnementaux déterminant le comportement d'une application, ou dans lesquels un événement d'intérêt pour l'usager se produit (Traduction libre)

Le Contexte est défini de manière très large par Lieberman et Selker [64] comme « *everything that affects the computation except the explicit input and output* »[35]; il s'agit donc d'une définition du Contexte centrée sur les tâches des composants logiques plutôt que sur sa pertinence pour un usager. Cette définition suggère que le Contexte ne soit pas ce qui est lu de l'environnement par un capteur ou appliqué à l'environnement pas un actuateur, mais plutôt ce qui est généré suite à la lecture par un capteur ou ce qui sera traduit en opérations par un actuateur.

Constat 6 : une acception possible du Contexte en fait tout ce qui affecte le traitement outre ce qui relève des entrées / sorties.

Constat 7 : certaines acceptions du Contexte, par exemple celles ciblées par les constats 5 et 6, sont très larges peuvent mener à une cardinalité importante de l'ensemble des Contextes.

En résumé, sur la base de ces quelques définitions du Contexte utilisées en pratique, certains constats peuvent être faits :

- les définitions proposées dans la littérature tendent à être pragmatiques et ciblées, convenant au problème adressé par les auteurs plutôt que cherchant à encadrer et à définir ce qu'est le Contexte de manière générale. Même la définition proposée par Dey [30], sans doute la plus connue d'entre elles, met l'accent sur les besoins d'un usager, excluant les applications contextualisées qui ne dépendraient pas d'un usager;
- les définitions de Ryan et al. [97] et de Fujii [36] sont fortement associées à l'environnement physique, mais sont difficiles à adapter pour inclure l'espace logique;
- les définitions de [64, 71, 73] sont très inclusives, acceptant une vaste gamme de données en tant que Contexte. Si elles ne souffrent pas d'une association trop forte avec le monde physique, elles mènent à un ensemble de Contextes de cardinalité considérable.

Un point sur lequel la littérature ne permet pas de conclure sans équivoque, comme en font foi les définitions relatées plus haut, est celui de la nature du Contexte, à savoir si le Contexte est : un type, une abstraction; une instance, au sens de la réification d'une abstraction; ou quelque chose de différent. Pour opérationnaliser le Contexte et pour le représenter formellement, cette question doit être éclaircie.

Le Contexte peut prendre plusieurs formes. À titre d'exemple :

- il est possible de représenter un lieu sous plusieurs formes : des positions dans l'espace Euclidien à partir d'une origine choisie, ce qui peut être adéquat si la surface représentée est relativement petite; des coordonnées polaires, ce qui sied aux grandes surfaces; des zones logiques (la cuisine, le salon, la salle de bains, …); etc.;

[35] Tout ce qui impacte un calcul, outre les entrées et sorties explicites (Traduction libre)

- il est possible de représenter le temps au sens d'un ordonnancement d'événements sous la forme de la relation *happens-before* [62], du moins dans certains cas, tout comme il est possible de déterminer un historique global des événements [17] dans un système clos. Dans l'espace ouvert, de manière générale, l'ordonnancement du Contexte sur une base chronologique de manière à déterminer le Contexte le plus récent dans un ensemble donné est une question à laquelle il est difficile d'apporter une réponse unique, et les Agents sont appelés à discriminer entre ces Contextes sur la base de critères qui leur appartiennent : le Contexte-selon remplace un *happens-before* par un *happens-before-according-to*. La logique temporelle [7] peut servir d'outil pour résoudre ces questions;
- s'ajoute à ces questions de forme le fait qu'il importe souvent plus d'établir des relations entre les Contextes que de les identifier et de les modéliser. Sur la base du temps ou du lieu, déterminer si un Contexte est inclus dans un autre Contexte, sous un autre Contexte, près d'un autre Contexte, ou encore si un Contexte précède un autre Contexte chronologiquement, sont des problèmes auxquels de nombreux projets ont été confrontés, et pour lesquels plusieurs solutions spécialisées ont été proposées.

2.2.2 Modélisation

Il est généralement admis qu'un modèle de Contexte doit être défini pour être en mesure d'en automatiser le traitement. Couvrir la totalité des Contextes possibles dans une ontologie générale est un objectif ambitieux, que certains ont cherché à atteindre, par exemple Abdulrazak et al. [1]. Plusieurs approches plus circonscrites ont aussi été tentées [107] au fil des années.

Parmi les principales formes que prend la modélisation du Contexte, on trouve :

- les paires {nom,valeur}, qui sont parmi les plus simples et servent souvent à identifier les caractéristiques des services de manière à faciliter leur découverte à partir des besoins des agents. Cette modélisation apparaît dans la plupart des bases de données NoSQL [98] de même que dans des systèmes multi-agents tels que Jade;
- les modèles basés sur la logique formelle, pour représenter le Contexte comme des affirmations sur l'existant puis raisonner sur la base de faits et de règles, comme par exemple celui mis de l'avant par McCarthy et Buvač [67];
- les modèles hiérarchiques, par exemple ceux décrits à l'aide de balises. On trouve entre autres plusieurs modèles de cette forme dans les technologies associées au Web sémantique, en particulier dans les ontologies qui constituent une catégorie en soi [77];
- les modèles à base d'ontologie, justement, approche recommandée par Strang et Linnhoff-Popien [107] à titre de type de modèle le plus expressif du lot. Pour encadrer le design d'une ontologie de Contexte, Korpipaa et al. [61] recommande entre autres de viser la flexibilité dans l'ajout de nouveaux éléments, la simplicité des relations et la généricité pour couvrir une plus vaste gamme de concepts;
- les modèles graphiques, par exemple ceux résultant d'une modélisation UML appliquée à la contextualisation de services Web [103]; et
- les modèles orientés objet, qui utilisent l'arsenal de ce modèle de programmation pour modéliser le Contexte et les services. Un exemple d'application de cette approche est Hofer et al. [45].

Comme le souligne Castillejo et al. [16], « *The first attribute that defines context is its variability, capability of change, or dynamism. This makes modelling context especially troublesome* »[36]. Peu importe sa forme, le Contexte est habituellement décrit à partir de diverses propriétés, par exemple le type ou la catégorie (temps, vitesse, température, etc.) et la valeur, souvent associée aux données brutes récoltées d'un capteur et exprimée en fonction de l'unité de mesure d'origine. Ces propriétés sont fréquemment enrichies de métadonnées, par exemple le moment où la donnée a été captée (suivant la perspective du capteur), les paramètres opératoires du capteur, et le degré de confiance associé au Contexte. Vu sous cette perspective, un format autodescriptif du Contexte semble souhaitable.

Constat 8 : en pratique, le Contexte doit fréquemment être enrichi de métadonnées pour jouer le rôle descriptif et autodescriptif qui est le sien.

Pris sous un angle plus large encore, sur un plan opératoire, le Contexte doit être :

- décrit de manière ouverte et extensible. Il serait suspect de croire que la totalité des Contextes possibles ait été envisagée de prime abord; par conséquent, prévoir un format pour le Contexte qui permette d'en élargir les possibilités est sage;
- auto-décrit, pour faciliter la découverte dynamique de sens, en particulier dans une optique de Contexte-selon;
- composable, pour construire des Contextes complexes à partir de Contextes plus simples;
- concis, pour économiser les ressources des nœuds les plus limités;
- identifiable de manière unique, pour qu'il soit possible d'établir à la fois l'équivalence et l'identité entre deux Contextes; et
- comparable, de manière à identifier les similitudes et des différences entre Contextes.

À titre d'exemple, les formats à base de balises tels qu'OWL [77] offrent plusieurs de ces caractéristiques mais souffrent du point de vue de la concision. La capacité de comparer deux Contextes pour établir en quoi ils se ressemblent et en quoi ils diffèrent permet à un Agent de faire des choix lorsque placé devant plusieurs Contextes susceptibles de convenir à ses besoins; que le Contexte permette de réaliser un tel examen de similitude sans recours à un composant tiers facilite l'autonomie des Agents.

2.2.3 Profil

Le profil, ou « *User Profile* », décrit traditionnellement le Contexte associé à un individu. Ce Contexte est parfois catégorisé de manière plus précise, comme dans [36] qui distingue le « *User Context* », ayant par exemple trait au dossier médical d'un individu, des préférences individuelles de cet individu. Pour éviter toute ambiguïté, nous écrivons Profil lorsque nous utilisons ce terme dans son acception technique.

Constat 9 : le Profil est une information au cœur de plusieurs systèmes contextualisés, en particulier ceux qui placent l'usager au cœur de la définition du Contexte.

[36] La principale caractéristique du contexte est sa variabilité, son dynamisme. Ceci complique singulièrement sa modélisation. (Traduction libre)

Parmi les formes que prend le Profil, selon les applications, on trouve un amalgame de considérations propres à l'individu, incluant ses préférences individuelles, qui permettent par exemple d'offrir à l'usager les médias qui l'intéressent au moment le plus opportun et sous la forme qui lui sied le mieux; un descriptif de ce dont l'individu a besoin; un descriptif évolutif de l'état physique et émotif de l'individu; etc. Comme l'explique Panagiotakis [78], « *The contextual information can be encoded in various related profiles such as, the user preferences profile and the terminal, ambient, network, and service profiles. The combination of all these profiles constitutes the User Profile* »[37]

En pratique, le Profil est à la fois personnel et personnalisé : les besoins varient selon les individus, et il est pragmatique, surtout avec des composants mobiles aux ressources quelque peu limitées, de ne conserver sur un nœud donné que les éléments de Profil pertinents à la mission des Agents (§3.4).

2.3 Agents

Dans la littérature, un **agent** est une entité qui accomplit une tâche pour le bénéfice d'une autre. Par exemple, selon Nwana : « *we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user* »[38] [72]. Des liens peuvent être faits entre les agents dans un système réparti et la théorie des acteurs de Hewitt et al. [44], ces derniers y étant définis comme « *a computational agent which has a mail address and a behaviour. Actors communicate by message-passing and carry out their actions concurrently* ».

La programmation par agents, ou *Agent-Oriented Programming*, est un terme mis de l'avant par Yohav Shoham [104]. Notant que « *The original sense of the word, of someone acting on behalf of someone else, has all but been lost in AI* » [39], l'auteur propose sa définition, soit « *An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments* » [40]. Cette définition générale sied au concept général tel que mis en pratique dans ContextAA, où le cadre ontique (§4.8) tient lieu de perspective propre à l'agent sur le monde, et montre en quoi cette perspective reposant sur des connaissances locales à l'agent sied à la perspective automonique mise de l'avant par Kephart et Chess [58].

Les approches de programmation par agents peuvent être groupées en familles [96], les principales d'entre elles étant :

---

[37] L'information contextuelle peut être encodée dans des profils variés tels que les préférences de l'usager et les profils de services, réseaux, environnement ambiant, etc. Le profil de l'usager est une combinaison de tous ces profils (Traduction libre)

[38] Nous définissons le terme agent comme un composant logiciel ou une composante matérielle capable d'agir pour le bénéfice d'un usager. (Traduction libre) Notons que [72] insiste sur la difficulté de définir clairement le terme « agent », celui-ci étant devenu quelque peu galvaudé.

[39] Le sens original du mot, soit d'agir pour un tiers, a essentiellement été oublié en intelligence artificielle. (Traduction libre)

[40] Un agent est une entité dont les états sont en fait des composants mentaux tels que des croyances, des capacités, des choix et des prises de position. (Traduction libre)

- les agents procédant par réflexe simple (*Simple Reflex Agent*), qui basent leurs décisions sur les données disponibles au moment présent, sans tenir compte du contexte historique. Ces agents peuvent être sans états (*Stateless*), qualité qui leur confère un certain degré intrinsèque de tolérance aux pannes et d'interchangeabilité. En retour, il leur est difficile de gérer correctement des tâches qui impliquent un suivi des actions dans le temps, par exemple gérer un dérapage dans une situation de conduite automobile autonome;
- les agents procédant par réflexe basé sur un modèle (*Model-Based Reflex Agent*), qui combinent règles et modélisation de l'état actuel des choses pour prendre une décision. Un tel agent peut être représenté en partie par un automate déterministe, et est typiquement conçu pour adresser une gamme bien définie de problèmes;
- les agents orientés vers un but (*Goal-Based Agents*), qui cherchent à atteindre un objectif. Une caractéristique de ces agents est qu'ils tendent à planifier en fonction de leur objectif et à s'adapter aux circonstances, ce qui rend leur processus de décision plus adaptatif;
- les agents d'utilité (*Utility-Based Agents*), qui visent à maximiser une métrique composite (par exemple, maximiser le bonheur d'un individu). Une telle métrique est nommée en anglais *Utility Function*. Le comportement de tout agent en apparence rationnel équivaut à l'action d'un agent d'utilité; enfin,
- un agent apprenant (*Learning-Based Agent*), à la fois capable d'apprendre à partir de nouvelles situations et d'évaluer la qualité de cet apprentissage pour s'autocorriger et s'améliorer.

En pratique, un agent est déployé sur un substrat logiciel, lui-même déployé sur un substrat matériel, ou **nœud**. Un nœud peut être un capteur, un actuateur, un ordinateur à part entière, ou tout autre appareil participant à la réalité enrichie, incluant les téléphones intelligents et les vêtements intelligents.

Le recours aux agents intelligents dans les applications déployées sur des appareils mobiles tels que des téléphones intelligents n'est pas chose nouvelle en soi. À titre d'exemple, Rashvand et Hsiao [86] répertorie de nombreuses applications mobiles reposant sur des agents intelligents et destinées à combler, chacune à sa manière, les besoins d'un groupe d'usagers; cette revue de littérature met aussi en lumière que la majorité de ces applications reposent sur une architecture stratifiée somme toute classique.

Selon Nwana [72], il y a historiquement eu deux grandes vagues d'agents, la première débutant vers 1977 avec entre autres Hewitt et al. [44] et mettant l'accent sur l'intelligence artificielle répartie, avec des agents chargés de modèles symboliques internes riches, et la seconde débutant vers 1990 et plus axée sur l'action que sur le raisonnement, avec des agents dont l'intelligence est de faible à modérée.

Cette dichotomie entre d'une part, quelques entités riches en termes de ressources et, d'autre part, un grand nombre d'entités plus pauvres, trouve écho dans d'autres créneaux de l'informatique, par exemple les débats autour des vertus respectives des clients minces et des clients riches dans les années '90, ou dans le passage des superordinateurs aux grappes d'ordinateurs plus conventionnels.

Certains, en particulier Kephart et Chess [58], défendent l'idée d'agents autonomiques, au sens d'agents capables de s'auto-configurer, de s'auto-protéger, de s'auto-guérir et de s'auto-optimiser. De tels agents seraient capables de mieux s'adapter à un environnement mobile ou changeant. Il est démontré dans Abdulrazak et al. [2] qu'un Agent autonomique peut s'exprimer dans une infrastructure contextualisée « macro », telle qu'on en trouve dans les espaces intelligents traditionnels, comme dans une infrastructure « micro », appropriée pour l'espace intelligent ouvert (chapitre 3) : en raisonnant sur des états locaux, de manière indépendante de nœuds spécialisés, un Agent dans une infrastructure « micro » est moins sensible aux fluctuations de qualité de service dans son environnement et plus susceptible de s'adapter au changement, tout en mettant de l'avant l'importance de s'adapter au contexte « macro » lorsque c'est possible pour accroître la qualité des services offerts. En particulier, ne pas dépendre de nœuds spécialisés facilite, par définition, l'opération d'un Agent en situation de connectivité faible ou nulle, ou en cas de panne d'un tel nœud spécialisé.

Le caractère autonome et adaptatif attendu des agents dans un espace ouvert est mis en relief par Lee [63], qui rappelle, dans un texte mettant l'accent sur l'importance de réagir à temps aux événements :

> *« The physical world, however, is not entirely predictable. Cyber-physical systems will not be operating in a controlled environment, and must be robust to unexpected conditions and adaptable to subsystem failures. […] But no component is perfectly reliable, and the physical environment will manage to foil reliability by presenting unexpected conditions »* [41][63]

Les agents répartis peuvent servir à modéliser des systèmes complexes de composants interreliés. Par exemple, Bosse et al. [11] présente un modèle de contagion émotive sur la base d'agents en interaction, et montre comment il est possible de modéliser même des comportements de groupes humains à partir d'un modèle simplificateur à base d'agents.

Le recours à des agents peut servir de mécanisme d'accompagnement individué dans un système contextualisé; d'ailleurs, comme en fait état [87], le recours à des agents ayant en apparence un « comportement social » peut accroître leur acceptation par un humain, et contribuer à une bonification de l'offre de services. La contextualisation contribue à enrichir l'acception que se fait l'agent de son environnement et de la situation des entités avec lesquelles il interagit.

[41] Le monde physique n'est pas pleinement prévisible. Les systèmes hybrides n'opèrent pas dans un environnement contrôlé et doivent être robustes et adaptables. Aucun composant physique n'est totalement fiable, et l'environnement physique est toujours susceptible de présenter des conditions inattendues. (Traduction libre)

La littérature portant sur les agents contextualisés mobiles et opérant dans l'espace intelligent ouvert inclut Zalavsky [119], qui prend position quant à la pertinence des agents mobiles, balise ce qui est attendu de tels agents en lien avec le Contexte (le découvrir, l'extraire, le manipuler, l'interpréter, le disséminer et l'utiliser), et décrit ce que nous qualifierons d'espaces ouverts dans une perspective nomadique (déplacement d'un espace intelligent à un autre tout en maintenant une forme de service de la part des agents); Jayaputera [55] qui décrit des agents mobiles créés (assemblés, en fait) sur demande pour répondre à des besoins découverts et accomplir une mission, réalisant ainsi automatiquement des tâches qui, prises ensemble, constituent des missions complexes; et Pratistha [83] qui propose d'intégrer services Web et agents mobiles en appliquant aux services Web des caractéristiques inspirées du monde des agents mobiles, en particulier en offrant une infrastructure facilitant la reconfiguration dynamique et la mobilité des services Web, ce qui permet de mieux intégrer ces deux familles de technologies.

Les agents contribuent sur le plan de la mécanique opératoire des systèmes répartis. Par exemple, le protocole SLP, pour *Service Location Protocol* [41], de l'Internet Engineering Task Force (IETF), repose sur trois types d'agents, soit les *User Agents*, qui réalisent la découverte de services pour un usager; les *Service Agents*, qui assurent la publication des services offerts sur un nœud, et les *Directory Agents*, qui ont pour tâche l'agrégation d'information sur les services avoisinants.

Constat 10a : dans la plupart des systèmes multi-agents, les agents jouent un rôle essentiel et définissent en quelque sorte le rôle du système auquel chacun participe.

Constat 10b : permettre à un agent de s'adapter à un environnement hétérogène comme l'espace intelligent ouvert est une condition préalable à l'atteinte d'une forme d'autonomie pour cet agent.

2.3.1 Contextualisation (*Context-awareness*)

Le fruit de raisonnement réalisé par un Agent sur du Contexte est connu sous le nom anglais de *Context-awareness*, pour lequel nous utilisons le terme **contextualisation**. Nous utilisons aussi **contextualisé** à titre de terme technique pour *Context-aware*.

Il existe plusieurs champs d'application pour la contextualisation. Par exemple :

- mieux comprendre le comportement d'un humain :

  > « *[…] such a [biomonitoring] system needs to be context-aware not only to help better assess the criticality of the subscriber's medical condition but also to better determine who to contact, what information to include, and what is the best way to contact that third party* »[42] [57]

- accompagner un individu dans ses activités quotidiennes;
- superviser un individu pour reconnaître un comportement inapproprié, détecter des signes de violence, un déplacement dans une zone interdite, ou simplement un bris de routine :

[42] Un tel système de biosurveillance se doit d'être contextualisé, non seulement pour mieux évaluer le la gravité de la condition médicale de l'individu surveillé, mais aussi pour savoir qui contacter, quelle information lui transmettre, et comment cet individu devrait idéalement être rejoint. (Traduction libre)

> « *Although the systems only work for some of the people, some of the time, studies have consistently shown that context-sensitive prompting can motivate behavior changes […] To present messages at just the right time requires computational sensing that can infer context from sensor data* »[43] [47]

- adapter l'environnement aux besoins et aux désirs individuels :

> « *Your [Context-aware, personal network] is aware of the fact that you are reading your e-book and that you sit in a moving car. Consequently, it increases the font size in order to increase your reading comfort* »[44] [116]

- diagnostiquer des défectuosités ou des pannes dans d'autres composants logiciels;
- etc.

Certains, par exemple Erfani et al. [32], estiment que la contextualisation , bien qu'insuffisamment répandue pour y arriver, serait une des principales clés de ce qui deviendrait le Web 3.0 : « *While domains such as Web 3.0 – the next generation of the web – have made context-awareness a main requirement of their solution space, the software engineering domain still lacks the same rate of adoption* »[45]. Les travaux décrit dans Grubert et al. [40] sur la jonction entre contextualisation et réalité augmentée donnent dans la même direction en suggérant l'importance croissante de la contextualisation dans les applications contemporaines; cet article discute d'ailleurs de ce qu'il nomme réalité augmentée omniprésente (*Pervasive Augmented Reality*), présenté comme « *a continuous and pervasive user interface that augments the physical world with digital information registered in 3D, while being aware of and responsive to the users context* » [40], et compare ce modèle à celui de la réalité augmentée « traditionnelle » qui demande des actions d'un usager. La réalité augmentée omniprésente a de particulier qu'elle est toujours potentiellement active, et entre en scène en fonction de la situation, du contexte dans lequel un usager est plongé. Les auteurs présentent principalement l'intérêt de la réalité augmentée omniprésente comme reposant sur la présentation d'information ajoutée au réel pour des usagers, par exemple ceux devant réaliser des tâches spécialisées.

Selon Henricksen et al. [42], « *pervasive systems need to be* context-aware*, i.e., aware of the state of the computing environment and also of the requirements and current state of computing applications.* »[46] Il est possible de décrire un Agent contextualisé tel que ceux sur lesquels repose ContextAA comme utilisant le Contexte pour réaliser sa tâche.

---

[43] Bien que les systèmes en question ne fonctionnent que pour certains individus, et encore pas tout le temps, il a été démontré qu'un signal contextualisé peut induire un changement comportemental […] Présenter un message au moment opportun requiert que l'on déduise du contexte des données tirées de capteurs. (Traduction libre)

[44] Votre réseau contextualisé personnel est conscient que vous lisez un livre numérique tout en étant assis dans un véhicule en mouvement; conséquemment, il en accroît la taille de la police de caractères pour maximiser votre confort (Traduction libre)

[45] Bien que des domaines d'application tels que le Web 3.0 fassent de la contextualisation un de leurs requis, celle-ci n'a pas encore été pleinement adoptée par le monde de l'ingénierie. (Traduction libre)

[46] Les systèmes ubiquitaires doivent être contextualisés, c.-à-d. conscients de l'état de leur environnement, de même que des exigences des applications et de l'état de leur exécution » (Traduction libre)

Plusieurs systèmes contextualisés ont pour caractéristique de se fondre dans le décor, condition mise de l'avant par Weiser et Brown [114], pour interférer le moins possible avec la vie normale de l'usager. La capacité pour le Contexte d'accompagner l'usager dans ses déplacements permet aussi à ce dernier d'avoir une vie la plus normale possible. Cette « normalité » peut être une condition de validité de certains Contextes : un système contextualisé participant à un diagnostic de maladie mentale gagnera sans doute en valeur si son action laisse l'usager agir de manière habituelle, sans interférer avec ses actions au quotidien.

Cette non-interférence avec les activités du quotidien implique aussi en quelque sorte la capacité des systèmes contextualisés de répondre aux stimuli en temps réel, ce terme étant ici pris au sens de « faible latence ». Les réponses offertes aux actions humaines sont alors plus naturelles, moins intrusives.

Certains mettent de l'avant une distinction entre un composant contextualisé et un composant dépendant du Contexte. Entre autres, selon Neovius et al. [71], « *A context-dependent part of the system depends on a Context provider to supply the metrics for fulfilling its declared service.* »[47]. En effet, dépendre du Contexte n'implique pas nécessairement une capacité de raisonner sur le Contexte, ce que laisse entendre le vocable anglais *Context-aware*. Un exemple simple de production de Contexte synthétique est le relais, par lequel un Agent publie un Contexte tel qu'il a été consommé, sans y appliquer de transformations. Dans un tel cas, on pourrait prétendre que l'Agent est dépendant du Contexte sans toutefois raisonner sur le Contexte.

Constat 11 : traditionnellement, une distinction est faite entre *Context-aware*, au sens de « utilise le Contexte et raisonne sur celui-ci dans l'exercice de ses fonctions » et *Context-dependent*, au sens de « dépend du Contexte dans l'exercice de ses fonctions ».

La contextualisation d'Agents mobiles comporte ses propres particularités. D'office, Chen et Kotz [18] décrit la programmation de composants contextualisés comme « *a mobile computing paradigm in which applications can discover and take advantage of contextual information* »[48]. Il laisse entendre que la mobilité est un élément essentiel de systèmes contextualisés, et que les applications contextualisées (*Context-aware*) devraient aussi être « mobilisées » (*mobility-aware*). Un peu à la manière de [100], il place le Contexte en quatre catégories, soit :

- le Contexte de calcul (*Computing Context*), reposant strictement sur les états de l'espace logique;
- le Contexte de l'usager (*User Context*), ce que plusieurs nomment le profil, ou Profil, dans ContextAA (§2.2.3);
- le Contexte physique (*Physical Context*), qui décrit les conditions environnementales; et
- le Contexte de temps (*Time Context*), qu'il recommande d'ajouter aux définitions classiques.

Dans une analyse de Lieberman et Selker [64], Bisgaard indique que ce dernier :

[47] Un composant dépendant du Contexte requiert qu'un fournisseur de Contexte lui offre les paramètres dont il a besoin pour offrir son propre service (Traduction libre)

[48] […] un paradigme de programmation mobile dans lequel les applications découvrent et tirent profil d'information contextuelle. (Traduction libre)

> « *[…] proposed another definition of context. […] [T]his definition is quite technical and is centred on the application […] [s]o the context is any information that the system senses beyond the direct commands and which have an effect on the state of the system instead of the user* »[49]. [8]

À cette définition du Contexte, Bisgaard voit à la fois des avantages et des inconvénients :

> « *This approach offers the opportunity to design systems that act semi-independent based on the actions of the users. The belief is that the system can sense changes to the environment and the action of the user. […] This approach differs from the user centred in that it does not need the user's authorisation to execute commands […] this many (sic.) cause confusion in the user […]. Another disadvantage is that the performance of the system is very much dependant on the interpretation of the context* »[50] [8]

Comme le fait encore remarquer Bisgaard : « *It is very difficult include (sic) every aspect of the context in a given situation and when context is seen in connection with the field of computer science it may not be every aspect of the context that is important to a research project* »[51]. Sur le plan de la continuité de services pour composants mobiles, il ajoute :

> « *Continuity through mobility is based on the idea that a rapidly changing environment and a dynamic context of use set a new level of required continuity in mobile devices. The prevalent view is that mobile systems must be able to accommodate interruptions in work, concurrency in work and a stop and go method of working. This puts a heavy burden on the would-be designer of a context-aware system; how do we maintain continuity in the interface when the interactions are as dynamic [8]* »[52]

Ce n'est pas tant le caractère dynamique des systèmes contextualisés mobiles que la discontinuité inhérente à la réalisation de leurs tâches qui est mise en relief dans ce passage : l'auteur dit voir un défi dans la jonction entre la discontinuité des tâches accomplies et le souhait de continuité dans l'interface de l'application. Pour ContextAA, permettre la continuité de service est plus qu'un tenir compte : il s'agit d'un objectif-clé (§4.7.4).

---

[49] [Ce dernier] a proposé une autre définition du Contexte. Cette définition est plutôt technique, et centrée sur l'application […] dont le Contexte est toute information détectée par le système sans être une commande lui étant directement adressée, et qui affectent l'état du système plutôt que l'usager. (Traduction libre)

[50] Cette approche permet de concevoir des systèmes au comportement semi-indépendant sur la base des actions des usagers. […] Cette approche diffère d'une approche centrée sur l'usager du fait qu'elle ne requiert pas l'autorisation d'un humain pour exécuter des commandes. […] Ceci peut entraîner de la confusion chez l'usager. Un autre désavantage de cette approche est que la « performance » du système dépend fortement de l'interprétation faite du Contexte. (Traduction libre)

[51] Il est difficile de tenir compte de tous les aspects du Contexte dans une situation donnée, et il se peut que ce ne soit pas tous les aspects d'un Contexte qui importent à un projet spécifique. (Traduction libre)

[52] La continuité par la mobilité repose sur l'idée qu'un environnement et un Contexte d'utilisation changeants accroissent le besoin de continuité des appareils mobiles. Le point de vue dominant sur les systèmes mobiles veut que ces systèmes doivent s'adapter aux interruptions, à la concurrence et au redémarrage de leurs tâches. Ceci place un fardeau sur les épaules des concepteurs de systèmes contextualisés, à savoir : comment maintenir la continuité dans l'interface usager quand les interactions sont dynamiques? (Traduction libre)

Plusieurs systèmes contextualisés reposent sur les services offerts par un intergiciel. Dans une présentation d'intergiciels pour systèmes contextualisés, Kjaer [60] indique qu'un intergiciel pour systèmes répartis et dynamiques comme ceux rencontrés dans un environnement ubiquitaire tend à :

> « *[…offer] suitable abstractions for dealing with heterogeneity and distribution without hiding them, and in some cases even provide information about distribution and heterogeneity as context information* »[53].

De même, Kjaer insiste sur la contextualisation sur la base de Contexte décrivant le lieu et la proximité, sans se restreindre à ces deux idées, et distingue le Contexte « externe », décrivant le monde physique, du Contexte « interne », décrivant le monde logique.

Constat 12 : certaines acceptions tracent une distinction entre le Contexte ayant trait au monde physique et le Contexte portant sur le monde logique.

Pour Baldauf et al. [4], le caractère calme et discret des technologies contextualisées est l'une de leurs caractéristiques essentielles :

> « *The term 'pervasive' […] refers to the seamless integration of devices into the users (sic.) everyday life. Appliances should vanish into the background to make the user and his tasks the central focus rather than computing devices and technical issues […] Context-aware systems are able to adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account* »[54].

Pour sa part, Schmohl et Baumgarten [101] suggère que la contextualisation distingue le Contexte « direct », émis directement des capteurs, du Contexte « indirect » qui est acquis par une application et est susceptible d'avoir été transformé. Dans une réflexion sur la modélisation du Contexte, ce texte distingue aussi le « *Context Theoretic Modeling* », influencé par les actions et les situations et réalisé *a posteriori*, du « *Context Conceptual Modeling* », réalisé *a priori* sur la base des relations entre les Contextes. À propos des composants contextualisés répartis, Schmohl et Baumgarten indique :

> « *[A] single node in the network is usually not interested in the complete context in the network, but rather only in relevant contextual information in its vicinity, which consist of the contextual data available at the proximate nodes. Thus, each node may define its context boundary, depending on its individual preferences* ».[55]

---

[53] […] offrir des abstractions appropries pour faire face à l'hétérogénéité et à la répartition, sans toutefois les cacher, et parfois même décrire la répartition et l'hétérogénéité sous forme de Contexte (Traduction libre)

[54] Le terme « pénétrant » fait référence à l'intégration harmonieuse des appareils au quotidien des usagers. Les appareils devraient se fondre dans le décor de telle sorte que l'usager et ses tâches deviennent le centre d'attention. […] Les systèmes contextualisés peuvent adapter leurs opérations au contexte sans intervention explicite d'un usager, et gagnent en efficience en tenant compte de l'environnement et du Contexte. (Traduction libre)

[55] Pris isolément, un nœud du réseau n'a typiquement pas besoin du Contexte entier de ce réseau, pouvant se limiter au Contexte pertinent dans son voisinage et disponible sur les nœuds à proximité. Ainsi, chaque nœud peut déterminer ses frontières contextuelles sur la base de ses préférences. (Traduction libre)

Constat 13 : certaines acceptions tracent une distinction entre le Contexte tiré directement d'un capteur et le Contexte résultant d'un traitement.

Constat 14 : certaines acceptions mettent de l'avant l'intérêt de faire varier les contours du Contexte sur la base des nœuds et des agents qui y logent.

2.3.2 Raisonnement réparti

Il est tentant d'estimer que le Contexte, en particulier s'il est brut, est une donnée factuelle, mais une telle position omettrait plusieurs considérations importantes du processus de contextualisation. La contextualisation dans une optique de Contexte-selon implique une perspective que l'on pourrait qualifier de « subjective », au sens où elle est teintée par l'Agent en laquelle elle se réalise :

- tout Contexte, même brut, résulte du travail d'un Agent, du moins si nous suivons une définition du Contexte inspirée de celle de [115];
- l'Agent fournisseur d'un Contexte peut avoir produit ce Contexte à partir de données incomplètes ou simplement fausses, découlant par exemple d'une panne matérielle ou d'une défectuosité chez les Agents desquels il a tiré ses sources;
- l'Agent fournisseur d'un Contexte lui-même peut être en panne, que ce soit dû à une erreur logique ou à une défectuosité matérielle (une mémoire déficiente, par exemple);
- pour des raisons qui ne tiennent à aucune défectuosité, deux fournisseurs de Contexte distincts peuvent offrir des Contextes distincts l'un de l'autre en réponse à une même requête. Pour qu'une telle situation se produise, il suffit que les moments où les données ont été saisies diffèrent, que l'un des deux capteurs ait une résolution plus fine que l'autre, que les lieux de saisie des deux capteurs diffèrent, ou même que les stratégies de production du Contexte soient différentes sur la base des mêmes sources.

Constat 15 : puisque le Contexte émerge d'un capteur ou naît de transformations réalisées sur des états physiques ou logiques « perçus » par un Agent en des moments précis, et puisque la transmission des données n'est pas instantanée, tout Contexte rendu disponible par un Agent est (relativement) vieux, et peut ne pas être la « meilleure » version de ce Contexte dans l'espace intelligent pour un autre Agent.

2.4 Architecture

Il existe bien entendu plusieurs architectures matérielles et logicielles avec lesquelles ContextAA partage des particularités. Nous examinons brièvement ici quelques-unes des plus connues d'entre elles, pour voir en quoi elles se ressemblent, mais aussi pour mettre en relief la raison d'être de ContextAA et de son modèle distinctif.

De prime abord, rappelons qu'un composant logiciel (incluant les Agents) s'exécute sur un composant matériel, parfois à travers une machine virtuelle ou, pour ContextAA, un Hôte (§3.5.1). Nous nommons **nœud** un tel composant matériel ou, par extension, la machine virtuelle qui l'occupe. Un nœud peut être un ordinateur traditionnel, connecté à Internet, comme il peut s'agir d'un appareil mobile aux ressources restreintes.

Un exemple représentatif des architectures contextualisées typiques est donné par [101], et reproduit à la figure 3, qui trace une distinction entre les composants matériels que sont capteurs et les actuateurs, et les interfaces de capture de Contexte. Ces dernières sont, dans son vocabulaire, des interfaces de bas niveau qui traduisent les lectures de l'espace physique en valeurs destinées à l'espace logique ou inversement. Il inclut aussi parmi les composants importants d'un système contextualisé :

- les dépôts de Contexte, ou « *Context Repository* », pour entreposer le Contexte;
- les raisonneurs, ou « *Context Reasoners* » (Moteur d'inférence dans la figure), générant du Contexte indirect à partir du Contexte direct;
- l'API de Contexte, qui permet la construction d'applications contextualisées;
- les applications contextualisées en tant que telles; et
- les interfaces de communication entre tous ces composants.

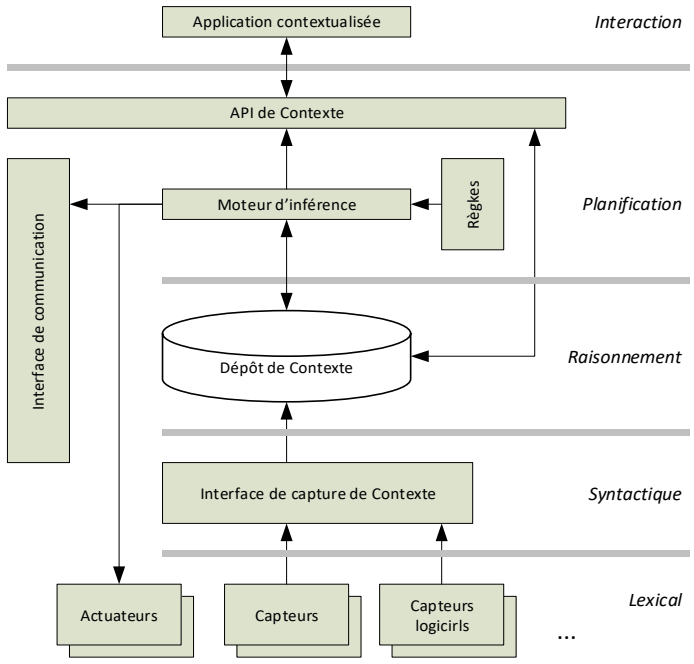


Figure 3 Architecture générale d'un système de gestion du contexte (reproduit de [101])

Schmohl et Baumgarten en sont arrivés à cette représentation type sur la base de leur rescension de l'existant. Bien que différente de celle que propose ContextAA, certains points en commun peuvent être identifiés (interface de communication avec d'autres applications; couche permettant d'interagir avec capteurs et actuateurs, existence d'au moins un dépôt de contexte, etc.). En retour, certains aspects de cette représentation (moteur d'inférence unique, dépôt de contexte unique) ne conviennent pas au Contexte-selon préconisé par ContextAA, ce qui explique certaines différences architecturales de notre proposition.

2.4.1 Modalités architecturales

Les modalités d'acquisition du Contexte et d'échange de Contexte entre Agents varient selon les projets, mais il est possible de regrouper ces façons de faire en un nombre restreint de catégories. Ces catégories ont bien entendu des conséquences importantes sur le plan architectural, influençant en particulier le degré de cohésion et le degré de couplage entre les composants de l'architecture.

À la suite de Chen et al. [20] et de Winograd [115], Baldauf et al. [4] insiste sur l'importance de séparer la détection et l'utilisation du Contexte, et met de l'avant sa préférence pour atteindre cet objectif à partir d'une architecture stratifiée (*Layered Architecture*). Dans une architecture stratifiée, chaque strate a un rôle à jouer, et les composants d'une strate communiquent typiquement qu'avec les strates voisines (bien qu'il existe des variantes de cette approche), ce qui localise à la fois les responsabilités des composants et les impacts de changements apportés à l'un ou l'autre d'entre eux.

Pour sa part, Chen et al. [20] propose un système de courtier (*Broker*) de Contexte ayant pour rôle de tenir à jour un modèle partagé du Contexte pour toutes les entités susceptibles d'en demander l'accès, de même qu'une ontologie partagée pour raisonner sur ce Contexte, et privilégie qu'un accès direct aux capteurs soit donné aux Agents dans le respect de certaines politiques de sécurité assurées par une médiation initiale à travers les services du courtier. Ceci permet à un Agent de traduire les données brutes en Contexte convenant à ses besoins. L'idée derrière cette approche est de supprimer des intermédiaires une fois l'accès initial établi, ce qui peut accélérer les échanges. Un coût évident de cette façon de faire est une réduction de flexibilité dû à un couplage plus direct entre le code client et les données.

Constat 16 : les architectures stratifiées et les architecture donnant aux agents un accès direct aux capteurs sont deux approches architecturales fréquemment rencontrées dans les systèmes contextualisés.

L'accélération potentielle du temps de traitement résultant du retrait d'intermédiaires est plus ardue à réaliser qu'il n'y paraît, du moins en général. Le recours à des intermédiaires spécialisés, définis par des experts du domaine, fait habituellement plus que compenser leur présence en tant qu'intermédiaires. Cependant, permettre l'accès direct au matériel lorsque nécessaire représente une porte de sortie pour des cas « niches », ou lorsque le contexte permet aux développeurs d'envisager faire mieux que ne le font les intermédiaires plus génériques.

Sur le plan de la modélisation du Contexte, Schmohl et Baumgarten discute aussi de la stratification d'architectures contextualisées, menant à un regroupement logique d'opérations contextualisées typiques, soit : la strate lexicale, qui interface directement avec les appareils; la strate syntaxique, où se produit la transformation des données lues en données répondant aux besoins de l'espace logique; la strate de raisonnement; la strate de planification; et la strate d'interaction.

Cette catégorisation met en relief que les architectures contextualisées les plus répandues sont faites d'un ensemble de composants ou de strates ayant des rôles clairement définis, et s'inscrivant dans une relation d'interdépendance sur la base des besoins des uns et des services offerts par les autres. Pour aborder l'espace intelligent ouvert, où la connectivité potentielle entre les composants peut varier en fonction du lieu et du moment dû à la mobilité présumée des nœuds, il est légitime de se questionner sur un modèle comme celui-ci, où les composants ne sont pas *a priori* indépendants les uns des autres.

Parmi les approches stratifiées, Chen et al. [20] discute d'accès au Contexte à travers un intergiciel. Cette approche stabilise le code client en le faisant passer par une strate uniforme facilitant l'ajout de composants dans un système contextualisé, mais est susceptible d'entraîner des coûts non-négligeables pour un système embarqué :

> « *A middle-ware (sic.) design trades computation resources for development convenience. In order to maintain a generic programming interface between the high-level applications and the low-level sensors, certain amount of computation resources (e.g., CPU power, memory, network bandwidths) must be allocated for the middle-ware operations. While this might not create a problem for devices that have rich computing resources, however, it will lead to resource contention problem in devices that have less resources (e.g., cellphones and embedded devices).* »[56] [20]

Cette mise en garde nous rappelle l'importance pour un intergiciel destiné à une plateforme embarquée d'être frugal dans sa consommation de ressources, car celles qu'il consomme dans l'écosystème de ces nœuds réduisent tout autant ce qui est disponible pour les applications. Cela dit, les intergiciels déployés dans les systèmes contextualisés tendent à offrir beaucoup plus qu'une simple interface de services, allant jusqu'à supporter la découverte automatique de services [37] ou à offrir des services spécifiques au domaine d'application, par exemple des composants spécialisés dans l'acquisition ou l'agrégation de Contexte [33].

On trouve aussi, dans Chen et al., une description de l'accès au Contexte à travers des serveurs de Contexte. Un serveur de Contexte est un composant spécialisé dont le rôle est d'alléger la tâche des Agents en rendant le Contexte disponible à partir d'un seul lieu, leur évitant d'accéder directement chaque fournisseur sur une base individuelle, et en réalisant sur le Contexte une part de traitement préalable. Comme en fait état Roy [93], bien qu'utiliser un tel nœud soit pertinent, dépendre d'un tel nœud n'est raisonnable que dans la mesure où un accès continuel à ce nœud peut être garanti, ce qui n'est pas le cas dans l'espace ouvert.

Constat 17 : le recours à des nœuds spécialisés tel des serveurs de Contexte est une pratique architecturale répandue dans les systèmes contextualisés.

De son côté, Winograd [115] propose une approche en quatre parties :

- des « widgets », composants enrichissant l'interface des pilotes des capteurs et des actuateurs;
- un gestionnaire de « widgets » pour assurer leur déploiement et leur gestion une fois ceux-ci déployés;
- des services réseaux pour faciliter la découverte des « widgets », ce qui découple quelque peu les « widgets » des agents; et
- un tableau noir (*Blackboard*) où publier du Contexte et d'où le consommer.

[56] Le design d'un intergiciel donne des ressources de calcul en échange d'une facilité de développement accrue. Pour maintenir une interface de programmation générique, une partie des ressources de calcul sont allouées aux opérations de l'intergiciel. Ceci peut ne pas affecter les nœuds riches en ressources, mais peut nuire aux nœuds plus limités. (Traduction libre)

Une ardoise est habituellement un média centralisé, commun à tous les agents d'un système. La proposition de Windograd est de faire en sorte que l'ardoise offre un système d'abonnement (approche *Publish/ Subscribe*), de nature événementielle, mais d'autres métaphores (en particulier le sondage, ou *Polling*) sont possibles. Certains systèmes commerciaux, par exemple JavaSpaces[57], appliquent une métaphore similaire à celle préconisée par Winograd. À titre de média centralisé, l'ardoise souffre du même inconvénient que les serveurs de Contexte de Chen et al., soit une obligation de disponibilité et de connectivité en tout temps, difficile à garantir hors d'un espace intelligent traditionnel.

Constat 18 : le recours à des médias centralisés de publication de Contexte est une pratique architecturale répandue dans les systèmes contextualisés.

Comme mentionné plus haut, une approche par stratification permet d'isoler les composants dans des couches bien définies, ce qui tend à réduire le couplage entre ces composants et à simplifier quelque peu la gestion du système, à tout le moins pour l'ajout, le retrait ou la mise à jour d'un composant. Cette approche contraint toutefois les composants à occuper des niches précises, et peut entraîner une consommation accrue de ressources sur des nœuds où celles-ci sont limitées, à moins que les composants par strate ne soient installés qu'au besoin ou de manière optionnelle; en effet, sur des nœuds à faibles ressources, il peut être pertinent de ne déployer qu'un ensemble minimal et nécessaire de composants pour fins d'économie.

Les strates ont aussi un impact sur le design d'un système, du fait qu'elles impliquent des interfaces clairement définies au préalable. À titre d'exemple, modifier l'interface entre les capteurs et les composants avec lesquels ils interagissent peut mener à un nombre important de mises à jour logicielles. Pour une technologie de composants répartis (COM, CORBA, ICE, etc.), les composants transigent à travers des intermédiaires (*Proxies*) générés à partir des interfaces; les intermédiaires doivent alors être recompilés lors de tout changement d'interface, ce qui mène à une recompilation de tous les clients et de tous les serveurs.

Il existe, surtout dans le monde du développement Web, des intermédiaires générés dynamiquement et moins fragiles lors de bris d'interfaces, mais ces derniers sont plus lents que leurs équivalents statiques, et demandent plus de puissance de calcul pour utiliser ou offrir des services équivalents.

Une approche mettant l'accent sur des nœuds spécialisés facilite la tâche des agents de plusieurs manières : les nœuds spécialisés délestent les agents de tâches qui demanderaient d'eux des ressources, en particulier du temps de calcul, et permet en quelque sorte d'insérer des îlots d'autorité sur le Contexte dans un système donné. Comme l'indique entre autres [4], offrir une gamme plus riche de services est libérateur pour l'Agent qui en profite. Assurer la disponibilité totale et la connectivité en tout temps avec ces nœuds peut toutefois être difficile à réaliser dans l'espace ouvert.

2.4.2 Systèmes multi-agents

Les systèmes multi-agents reposent typiquement sur :

- des cadriciels, ensembles intégrés d'outils conçus pour résoudre des problèmes complexes et porteurs d'une façon de faire pour résoudre ces problèmes;

[57] http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/

- des architectures, constituées de composants existants, de leurs propriétés et de leurs interactions mais agnostiques quant aux pratiques de résolution de problèmes;
- des bibliothèques, ensembles de classes et de fonctions regroupées selon certains critères et assemblées selon les besoins du code client; et
- des boîtes à outils (*Toolkits*), spécialisées pour des domaines d'application précis.

Ces catégories ont des frontières floues, et il serait possible dans bien des cas de placer un même outil dans plus d'une catégorie.

Parmi les outils voués au support de la contextualisation les plus connus, on trouve le Context Toolkit. Dans Dey et Abowd [28], le Context Toolkit se veut un cadriciel pensé pour concevoir des applications contextualisées capables d'utiliser une variété de contextes et une variété de capteurs.

Le premier requis mis de l'avant par Dey et Abowd pour le Context Toolkit est un mécanisme de spécification du Contexte, au sens d'un mécanisme permettant au développeur d'indiquer le Contexte dont dépend son application. Les autres requis importants de cet outil sont l'interprétation, l'entreposage et la gestion du Contexte, de même que la gestion transparente de la communication entre composants répartis et des mécanismes pour la découverte de ressources.

Le Context Toolkit met de l'avant à la fois une approche stratifiée et des composants spécialisés, en particulier pour rendre le Contexte en tout temps disponible pour les agents, ce que Dey et Abowd nomme *Constant availability of Context acquisition*. Avec le Context Toolkit, le Contexte est collecté par des « widgets », puis colligé dans des nœuds agrégateurs spécialisés, et déposé dans des *Situation Nodes*, composants de haut niveau associés à une entité.

Constat 19 : assurer la disponibilité du Contexte en tout temps est une pratique architecturale répandue dans les systèmes contextualisés.

Le Java Context-Aware Framework, ou JCAF [5], se présente comme une infrastructure orientée services reposant sur la subdivision de l'acquisition, de la gestion des de la distribution du contexte à travers un réseau de services. Dans le modèle mis de l'avant par JCAF, la recherche de services et d'entités sont des mécanismes dynamiques, et il en va de même pour leur intégration au système, ce qui permet d'ajouter ou de supprimer des agents sans redémarrer le système.

Le Contexte dans JCAF modélise des entités, contreparties logiques d'objets physiques; des relations (p. ex. : *located-in*); et des *Context Items* tels qu'un endroit spécifique, ce qui mène à des descriptions de Contexte ressemblant à des énoncés factuels (*Alice* is *located in* the *living room*) qu'il est par la suite possible de traiter par logique argumentative. Le fournisseur d'un Contexte l'accompagne d'affirmations quant à sa qualité (est-il sécuritaire? À quel point est-il précis?), forme de Contexte-selon que le consommateur doit évaluer pour déterminer le degré de confiance à leur accorder.

La communication dans JCAF se fait par sérialisation du Contexte en format XML, et la sérialisation se fait par les mécanismes RMI de Java [6]. La communication événementielle y est privilégiée. Ce sont à la fois des forces et des faiblesses pour JCAF : la flexibilité du format XML s'accompagne d'un poids sur les ressources disponibles, et il en va de même pour l'utilisation d'une machine virtuelle aussi puissante, mais aussi lourde que la JVM; de plus, bien qu'il soit possible de procéder autrement, privilégier la communication par RMI fait de JCAF une plateforme moins accueillante pour des technologies tierces et moins gourmandes en ressources.

Les composants JCAF peuvent être répartis sur plusieurs nœuds et sur diverses strates; la strate la plus près du matériel comprend des *Context monitors*, qui interagissent avec les capteurs de manière synchrone ou asynchrone, et des *Context actuators* qui opèrent des actuateurs. JCAF supporte des composants spécialisés pour la sécurité et la transformation de Contexte (incluant les agrégateurs et les traducteurs de Contexte) [5]. La découverte de services y passe par un bottin où les fouilles se font sur la base d'identifiants de services.

Le Java Agent Development Framework, ou JADE, est un cadriciel multi-agents implémentant les spécifications protocolaires du Foundation for Intelligent Physical Agents, ou FIPA [54], décrit sommairement dans §2.4.3. Le conteneur d'agents de JADE pour un nœud donné est une JVM. La communication s'y fait par une file de messages avec plusieurs modalités d'accès (bloquant, non-bloquant, par reconnaissance de modèle, etc.). JADE supporte aussi plusieurs protocoles de communication généraux, dont IIOP, http, le RMI de Java, etc. Une description de l'intégration de JADE avec OSGi est donnée par [105], alors qu'une description de JADE dans Gaia est offerte par [69].

Un agent dans JADE peut échanger de manière bidirectionnelle en mode point à point avec tout autre agent du même système. Les agents, pris individuellement, sont monoprogrammés, mais JADE peut exécuter plusieurs agents concurremment, à l'insu des agents. Il est possible de compiler JADE pour des plateformes à faibles ressources, ce qui permet de les déployer sur une variété de nœuds, mais JADE requiert tout de même une JVM pour fonctionner, ce qui entraîne un coût de base non-négligeable sur des nœuds à faibles ressources. Dans les limites de la plateforme Java, des tests de résilience en fonction de la charge laissent croire que les mécanismes de traitement de messages de JADE peuvent soutenir une charge de travail élevée [22].

La découverte de services avec JADE passe par un bottin où les fouilles sont faites à partir d'identifiants uniques d'agents. Certains services, par exemple la sécurité et la persistance des objets, y sont expérimentaux ou offerts par des tiers-partis tels que Vila et al. [112].

Un agent dans JADE est une instance d'une classe dérivée d'Agent, prise en charge par un moteur et intégrée dans un cycle de vie. La migration d'un agent passe par une suspension de son traitement, suivie de sa sérialisation vers le nœud de destination et son redémarrage une fois déployé à cet endroit.

Constat 20 : la migration d'un Agent entre deux nœuds peut se faire par sérialisation si le format dans lequel l'Agent est décrit s'y prête.

Enfin, Jini est un outil plus généraliste que les précédents, se voulant une architecture de services répartis pour exploiter Java dans l'optique de construire des systèmes constituant une fédération de services et de clients en réseau. En quelque sorte, Jini vise à adresser ce que Rotem-Gal-Oz [91] nomme, à la suite de [110], les huit faussetés de la programmation répartie. Cependant, bien que Jini permette de concevoir des systèmes répartis d'agents contextualisés, tel n'est pas son objectif premier.

Avec Jini, la découverte de services passe par un bottin où les services doivent s'inscrire; Jini élimine de ce bottin les services qui semblent ne plus répondre à l'appel. Jini met en place des mécanismes de communication sur la base de composants jouant le rôle de courtiers de messages. La communication avec Jini repose sur les mécanismes RMI de Java, sur lesquels circulent des objets Java sérialisés; comme pour JADE, ceci constitue une force à l'intérieur de programmes Java, mais complique l'interaction avec des technologies tierces qui doivent implémenter le nécessaire pour parler avec Java dans le respect des règles de ce langage, plutôt que d'implémenter le nécessaire pour assurer une forme d'interopérabilité sur la base d'un protocole plus neutre. Jini s'accompagne de JavaSpaces, un média d'entreposage partagé sur le réseau pour faciliter l'entreposage et l'échange de données.

2.4.3 Modalités de communication

Comme l'a mis en relief la section précédente, sur le plan protocolaire comme sur le plan des pratiques, plusieurs modalités de communication sont possibles pour un système d'agents répartis, et chaque technologie peut en supporter plusieurs. Parmi les principales, on trouve :

- implémenter un protocole maison, basé par exemple sur TCP ou UDP. Ceci offre une liberté maximale mais demande un effort important de programmation et de contrôle de qualité;
- utiliser des protocoles spécifiques à une technologie ou l'autre, comme ceux sous-jacents à RMI pour Java. Ces protocoles sont en général efficaces, testés et rodés, mais participent à des écosystèmes *a priori* fermés, ce qui demande un effort d'intégration pour qui choisit une autre technologie pour implémenter l'autre homologue du lien de communication. Comme nous l'avons vu plus haut, c'est le choix fait par JADE et Jini;
- utiliser un protocole RPC tout usage, comme il en existe plusieurs (DCE/RPC, RPC/XDR, SOAP, XML-RPC, etc.), variant en complexité et en format;
- utiliser un protocole d'une autre forme, par exemple une approche ReST [34] où chaque ressource correspond à une URL et où le serveur est présumé sans états, entraînant un transfert des états pertinents par le client avec chaque requête; ou encore
- avoir recours à un protocole spécialisé pour la communication entre agents, incluant par exemple des requêtes par un agent quant aux « croyances » d'un autre agent, ou encore une affirmation publique d'un « fait » par un agent.

Parmi les plus connus des protocoles spécialisés pour la communication entre agents, on trouve FIPA [35], pour *Foundation for Intelligent Physical Agents*, un standard IEEE implémenté entre autres par JADE, et en particulier son *Agent Communication Language* et son *Interaction protocol*, qui décrit des actes langagiers (*Speech Acts*) rappelant ceux mis de l'avant par McCarthy et Buvač [67] :

> « *Speech acts may be understood in terms of an intentional level description of an agent. An intentional description makes references to beliefs, desires, intentions & other modalities* »[58] [38]

[58] Les actes langagiers peuvent être comprise en termes d'une description intentionnelle d'un agent, faisant référence à ses croyances, ses désirs, ses intentions, etc. (Traduction libre)

> « *The theory of speech acts distinguishes between illocutionary acts, such as telling someone something, and perlocutionary acts, such as convincing him of it. […]. Thus there is an input distinction between hearing that and learning that analogous to the output distinction between telling and convincing* »[59] [67]

Les requêtes dans FIPA peuvent être exprimées en lien avec une ontologie, que Greenwood [38] décrit comme un vocabulaire commun de définitions consensuelles et de relations entre ces définitions sans l'objectif de décrire un domaine particulier. Le concept d'acte langagier, qui trace une distinction entre le factuel, tenu pour vrai, et ce qui est simplement affirmé par un agent, est fécond pour l'espace ouvert, et sous-tend une partie de notre modèle, bien que ContextAA n'adresse pas la question de convaincre et déplace la tâche de déterminer à quelle interprétation d'un Contexte adhérer du côté des Agents consommateurs

Un autre protocole inter-agents spécialisé très connu est Juxtapose, mieux connu sous le nom de JXTA, qui est en fait un ensemble de protocoles pour une communication point à point indépendante de la topologie réseau sous-jacente. JXTA définit un réseau virtuel distinct du réseau réel [13], et permet entre autres d'éviter des obstacles comme des murs coupe-feu ou des unités NAT (*Network Address Translation*). L'indépendance quant au réseau est une vertu dans l'espace ouvert, mais JXTA y parvient grâce à des nœuds spécialisés, une approche fragile dans un environnement mobile.

En résumé, les architectures et outils servant au développement et au support de systèmes répartis d'agents contextualisés sont parfois généralistes (Jini est un exemple bien connu), souvent spécialisés. La plupart de ces outils assurent la communication par des systèmes de messagerie, certains privilégiant par contre la communication point à point.

La plupart des outils reposent sur des nœuds clés, offrant des services spécialisés et dont dépendent les agents et les autres nœuds. Ces nœuds clés sont parfois spécialisés pour la contextualisation (p. ex. : des agrégateurs de Contexte, des moteurs de raisonnement), parfois d'ordre plus général (p. ex. : un bottin pour inscrire et découvrir des services).

Les pratiques architecturales principales varient selon les outils : le Context Toolkit insiste sur la spécification du Contexte; JCAF décrit le Contexte comme des actes langagiers; JADE insiste sur le modèle d'exécution des agents; Jini met l'accent sur la distribution de services. Pour JADE et Jini, les composants répartis sont au cœur des préoccupations alors que les considérations propres au Contexte sont pour l'essentiel du ressort des applications. De leur côté, le Context Toolkit et JCAF placent le Contexte au cœur du développement. La majorité de ces outils se disent ouverts, mais sont en pratique fortement associés à une plateforme (habituellement Java), à son modèle d'exécution et à ses formats de données.

Peu importe l'outil ou l'approche, certains services doivent être offerts à des agents, en particulier à des agents contextualisés, autonomiques et mobiles, pour qu'il soit possible de mettre au point des systèmes répartis d'agents contextualisés sans devoir réécrire le tout de rien à chaque fois. La véritable question est de savoir quels services, et sous quelle forme les offrir.

---

[59] La théorie des actes langagiers distingue les actes illocutoires, comme dire quelque chose à quelqu'un, des actes perlocutoires, comme convaincre quelqu'un de quelque chose. Ainsi, il existe une distinction entre entendre et apprendre, analogue à la distinction entre dire et convaincre. (Traduction libre)

Pour ce qui est de la gamme de services, les plus souvent rencontrés dans les systèmes répartis d'agents contextualisés sont ceux destinés aux fins suivantes :

- identification et nommage;
- équilibrage de la charge de travail;
- communication et messagerie;
- prise en charge des agents (activation, suspension, exécution);
- gestion des types et des algorithmes;
- gestion des ressources disponibles;
- découverte de services;
- récolte de Contexte;
- filtrage de Contexte;
- publication de Contexte.

Pour adresser les problématiques de l'espace intelligent ouvert, il importe que ces services soient offerts sans entraîner de dépendance envers des nœuds clés, tout en tenant compte du contexte (ressources souvent limitées sur les nœuds, Agents mobiles, environnement hétérogène, etc.). Puisque tous les nœuds auront à offrir ces services, il semble raisonnable de les offrir à même l'infrastructure logicielle sur laquelle seront déployés les Agents.

Enfin, les deux grandes familles traditionnelles de stratégies de coordination entre agents ou entre services, soit l'orchestration, où un agent particulier (l'orchestrateur) joue un rôle prépondérant dans les prises de décision, et la chorégraphie, où les agents conviennent d'une marche à suivre sur la base d'échanges de messages entre égaux, se rapprochant typiquement d'un processus démocratique. Ces modalités s'expriment habituellement par exposition et utilisation de services, ce qui ne sied pas au modèle de ContextAA (§3.5.7).

2.4.4 Ontologie

L'ontologie, bien que plus un élément de modélisation des systèmes contextualisés qu'un composant architectural, teinte directement l'architecture de plusieurs systèmes contextualisés, et il en va de même (dans une acception spécialisée) pour ContextAA. Les définitions proposées pour le terme ontologie varient, mais incluent :

- la spécification d'une conceptualisation partagée d'un domaine (traduit librement de [74]);
- une liste de termes formels, accompagnés de définitions et de contraintes formelles portant sur ces termes. Ainsi, l'ontologie inclut toute représentation ou spécification d'un domaine, outre les croyances quant à des objets spécifiques (traduit librement de [75]);
- une spécification formelle explicite pour représenter des objets, des concepts et d'autres entités dont l'existence dans un espace d'intérêt est présumée, de même que les relations entre ceux-ci (ibid.);
- une spécification explicite et formelle d'une conceptualisation partagée (traduit librement de [39]).

Ces définitions montrent que l'ontologie est généralement considérée comme décrivant à la fois des entités et des relations formelles entre ces entités. Chaque ontologie tend à se polariser autour d'un centre de gravité discursif, ou domaine d'intérêt, et la plupart des ontologies portent sur un domaine circonscrit, qu'elles décrivent de manière à permettre de l'analyser de manière automatique.

Certaines définitions proposées pour l'ontologie vont plus loin. Gruber [74] indique qu'en informatique, l'ontologie définit un ensemble de primitives (classes, propriétés, relations, …) modélisant un domaine de connaissances. Ces primitives incluent de l'information à propos de leur sens, de même que des contraintes quant à leur application logique cohérente.

Le site officiel du langage OWL[60], le Web Ontology Language, sans doute le plus connu pour ce qui est de décrire des ontologies, décrit l'ontologie comme définissant les termes utilisés pour décrire et représenter des connaissances d'un certain domaine, incluant des définitions opératoires de concepts de base dans ce domaine et les relations entre ces concepts, le tout dans une optique de réutilisation des connaissances. Les auteurs ajoutent qu'une ontologie est habituellement exprimée dans un langage logique, permettant de distinguer avec précision les classes, propriétés et relations des objets du domaine qu'elle représente, et mentionnent l'existence d'outils pour raisonner automatiquement sur une ontologie et offrir des services aux applications intelligentes

Ces définitions illustrent que l'acception générale du terme ontologie place à l'avant-plan son volet formel, décrivant une adéquation entre ontologie et énoncé logique, et insistant sur l'importance de faire concorder les actions d'un agent avec le formalisme que cette ontologie décrit.

Constat 21 : l'optique soutenue par OWL met de l'avant l'ontologie comme partie de l'automatisation d'une offre de services pour des tiers.

[60] http://www.w3.org/TR/webont-req/#onto-def (consulté le 4 mars 2015)

Chapitre 3
ContextAA – Conception

« Un concept est une invention à laquelle rien ne correspond exactement, mais à laquelle nombre de choses ressemblent. » (Friedrich Nietzsche / Posthumes)

Le présent chapitre est le premier de trois visant à décrire ce qu'on pourrait qualifier du « cœur » de ContextAA.

Les sections qui suivent présentent notre conception de ce qu'est un espace intelligent de manière à tenir compte de la perspective particulière qu'apporte l'espace intelligent ouvert; de ce qu'est le Contexte en tant que concept et en tant que langage, mais sans aller jusqu'à sa modélisation, pour des raisons exposées dans §3.3.1; et des Agents tels que ContextAA les conçoit. Nous complétons avec une définition des concepts architecturaux clés de ContextAA et des éléments les plus importants de la dynamique de contextualisation qui s'y opère.

Nous débutons tout d'abord par une revue des constats faits au chapitre 2 sur la base de notre revue de l'état des connaissances.

3.1 Pourquoi ContextAA – Les besoins

Le chapitre 2, portant sur l'état des connaissances, met en relief une gamme variée de solutions à un ensemble de problèmes associés aux espaces intelligents. Une lecture attentive de ce chapitre montre toutefois qu'il reste beaucoup à faire, comme l'indiquent les constats qui y ont été faits.

C'est à partir de ces besoins que se sont dessinés les contours de ce qui est devenu par la suite le design de ContextAA.

3.1.1 Sur le plan des environnements intelligents

Étant donné le constat 1, qui dénote la diversité de l'offre de services dans un environnement ubiquitaire urbain, il nous semble important d'offrir un système capable de naviguer la diversité technologique et de s'y adapter. ContextAA se veut un tel système, et son adaptabilité est une conséquence de son design.

Les constats 2a et 2b, à l'effet qu'assurer la continuité de service pour un individu mobile se déplaçant hors des espaces intelligents demeure difficile, révèle selon nous un besoin réel. Ce besoin gagne en importance pour qui considère les changements dans le mode de vie des humains, qui portent désormais souvent sur eux des ordinateurs « géolocalisés » et capables d'interopérabilité. Ceci met la table pour l'avènement de systèmes pensés pour l'espace ouvert, qui se distinguent de ceux conçus pour les espaces nomadiques en cherchant à maintenir l'offre de services même entre les espaces intelligents traditionnels, là où les ressources disponibles sont moindres. L'architecture de ContextAA est pensée depuis le début pour répondre à ce besoin, et nos travaux visent à appréhender ce que nous qualifions d'**espace intelligent ouvert** [93], un espace intelligent sans frontières, qui inclut les espaces intelligents traditionnels comme des îlots riches en ressources

3.1.2 Sur le plan du Contexte

Le constat 3 peut surprendre celles et ceux qui estiment que l'espace intelligent ne prend son sens qu'en présence d'humains, mais s'avère essentiel pour une perspective comme celle soutenue par ContextAA. En effet, dans ContextAA, chaque Agent cherche à accomplir sa mission; si certaines missions sont bel et bien centrées sur les besoins des humains, rien ne l'exige et, en pratique, plusieurs Agents accomplissent, sur la base de Contexte, des tâches qui ont surtout trait au bon fonctionnement du système dans son ensemble. Bien que les humains soient importants pour ContextAA, il demeure que ce système ne se centre pas de manière essentielle sur l'humain dans son environnement.

Le constat 4 est à l'effet qu'il existe des efforts pour catégoriser le Contexte. Les gains potentiels sur la base de tels efforts, manifestes pour la résolution de problèmes spécifiques, sont moins évidents pour des problèmes généraux et déterminés par le « Contexte-selon » (§3.3.1), comme c'est le cas avec ContextAA. Pour cette raison, la distinction des Contextes avec ContextAA est d'abord une question opératoire (§4.5).

En lien avec le constat 3, les constats 5 et 6 élargissent la définition du Contexte, le menant plus près de ce que ContextAA nomme l'espace contextuel (§3.4.4). Comme le met de l'avant le constat 7, trop inclure dans une même définition de Contexte peut mener à une explosion de la cardinalité de l'ensemble des informations à considérer; ceci explique en partie notre approche de Contexte subjectif, ou « Contexte-selon » (§3.3.1).

Le constat 8 explique en partie pourquoi, de manière inhérente, le Contexte dans ContextAA est un langage représentant à la fois données et métadonnées. Le choix que nous avons fait d'élargir le Contexte pour en faire un langage capable de représenter aussi des opérations sur le Contexte est expliqué dans les conséquences architecturale des *a priori* de ContextAA (§3.2.1).

Le constat 9, en conjonction avec le constat 3, explique pourquoi ContextAA diffère de la plupart des systèmes contextualisés dans son acception du Profil. En effet, sur le plan technique, du moins dans l'acception qu'en fait ContextAA, rien ne distingue fondamentalement le Profil d'autres Contextes. La raison pour laquelle le Profil se distingue en pratique des autres Contextes, et se mérite une dénomination particulière, tient à son rôle central dans un grand nombre de systèmes contextualisés (constat 3), position qui ne sied pas à une architecture comme celle de ContextAA.

3.1.3 Sur le plan des Agents

Comme le mettent en lumière les constats 10a et 10b, un système multi-agents est au moins en partie défini par les actions des agents qui l'habitent. Dans l'espace intelligent ouvert, pour lequel a été conçu ContextAA, un Agent peut jouer une multitude de rôles, du plus simple au plus complexe, comme par exemple :

- transformer le signal lu d'un capteur en données numérique dans un format particulier, passant par exemple d'une valeur dans l'intervalle [0..1] à un Contexte décrivant une valeur exprimée en °C;
- transformer une donnée d'un système de référence à un autre, comme par exemple transformer un Contexte exprimant une valeur en °C en son équivalent exprimé en °F;
- réaliser une agrégation de Contextes de diverses provenances pour les rendre disponibles à d'autres Agents, comme le suggère Chen et Kotz [19];

- publier un Contexte synthétique à partir d'un ou de plusieurs Contextes sources, comme par exemple générer un indice de confort à partir de Contextes décrivant l'environnement (température, humidité, illumination, etc.);
- raisonner sur du Contexte à des fins diagnostiques, pour générer un signal d'alarme lorsqu'un individu semble se placer en situation de risque ou lorsqu'un appareil semble défectueux, comme le propose Yu et al. [118];
- générer une commande vers un actuateur pour modifier les paramètres de l'espace physique. Un exemple de ceci pourrait être d'activer un déshumidificateur dans le but de réduire l'humidité ambiante à un niveau respectant les paramètres de confort d'un individu; etc.

Le Contexte est donc en partie une forme d'interprétation à partir d'une réalitée perçue par la médiation de capteurs ou à partir du raisonnement réalisé par un ou plusieurs Agents. Cependant, le Contexte occupe un rôle plus important encore dans ContextAA, pouvant aussi servir à titre de représentation des actions et opérations appliquées sur le Contexte et représentation des Agents entre autres choses. Dans ContextAA, l'interprétation du Contexte par un Agent se fait sur la base du cadre ontique de cet Agent.

Tout comme le font d'autres technologies multi-agents telles que Jade ou JINI par exemple, ContextAA déploie sur les nœuds des composants spécialisés, ou composants Hôtes, et c'est sur ceux-ci que les Agents eux-mêmes sont déployés, Ceci permet de concevoir des Agents d'une manière indépendante de la plateforme sous-jacente. Les Hôtes de ContextAA ont ceci de particulier qu'ils visent à une consommation très restreinte de ressources, et y arrivent en adoptant des modalités de traitement sur la base du seul Contexte.

3.1.4 Sur le plan de la contextualisation

Le constat 11 trace une distinction entre deux formes de contextualisation. En pratique, notre acception de la contextualisation avec ContextAA est plus proche d'une relation de dépendance envers le Contexte que de celle suggérant une obligation de raisonnement sur le Contexte; ContextAA admet et soutient le raisonnement sur le Contexte, sans toutefois l'obliger, mais son architecture est telle que tous les Agents consomment et produisent du Contexte dans l'exercice de leurs fonctions

Le constat 12 rappelle, par un autre angle que celui des constats 3 et 9, qu'il existe des distinctions dans la littérature entre les types de Contexte. ContextAA vise pour sa part la réunion de ces deux familles de Contexte dans une réalité enrichie, le tout dans un effort conjoint de simplification et de généralisation de ce qu'est le Contexte.

Le constat 13 fait remarquer une distinction entre le Contexte provenant directement d'un capteur et celui résultant d'un traitement. Sans que cette distinction ne soit importante pour ContextAA, nous la faisons sur le plan terminologique pour clarifier le propos. Ainsi, le Contexte émis directement d'un capteur est dit « Contexte brut » et le Contexte résultant d'un traitement réalisé sur un ou plusieurs Contextes est dit « Contexte synthétique ».

3.1.5 Sur le plan de la perspective sur le Contexte

Le constat 14 est un premier pas vers ce que nous nommons Contexte-selon (§3.3.1), et qui se situe au cœur de l'approche adoptée par ContextAA. Cette approche est de déterminer ce qui est du Contexte et posant la question du Contexte *pour un Agent*, plutôt que la question du Contexte *en soi*.

Le constat 15, quant à lui, sert de base pour notre définition du Contexte-selon. Puisque les besoins dépendent de chaque Agent et puisque la qualité relative des Contextes dépend aussi de chaque Agent, tout Agent doit inclure dans ses modalités de traitement une forme de mécanisme pour discriminer entre les Contextes. Cette réalisation du « Contexte-selon-lui » est, dans ContextAA, inscrite dans son cadre ontique (§4.8).

3.1.6 Sur le plan architectural

Le constat 16 met en relief le fait que plusieurs architectures de systèmes multi-agents se présentent sous un angle stratifié, pour fins de réduction de couplage, comme l'illustre d'ailleurs la figure 3, alors que certaines privilégient plutôt un accès plus direct aux capteurs, pour fins d'efficacité. D'une manière très circonscrite, ContextAA peut être vu comme stratifié du fait qu'un groupe d'Agents, les Agents standards, offre des services aux autres Agents (Agents du domaine), et du fait que tout Contexte obtenu par un Agent du domaine l'est en pratique par la médiation de son espace contextuel. L'acquisition de Contexte brut, qui se fait par accès direct aux capteurs, est une responsabilité donnée à des objets autonomes qui jouent le rôle d'intermédiaire avec les Agents.

Bien que le constat 17 soit à l'effet que la pratique d'utiliser des serveurs de Contexte ou d'autres nœuds spécialisés soit répandue dans les systèmes contextualisés, cette façon de faire ne convient pas à ContextAA. En effet, l'objectif de continuité de service est plus difficile à atteindre si les Agents dépendent de nœuds spécialisés, car avec une telle dépendance, un bris de connectivité compromettrait la capacité des Agents de communiquer avec ces nœuds spécialisés, et donc celle de mener à bien leur mission. Avec ContextAA, chaque nœud est en pratique un serveur de Contexte, préservant l'utilité du concept tout en supprimant les dangers qui lui sont associés.

De manière analogue, le constat 18 relève le caractère typique de la pratique qui consiste à utiliser des médias centralisés de publication de Contexte dans les systèmes contextualisés. Avec l'espace contextuel du composant Hôte, ContextAA offre en pratique plusieurs tableaux noirs, chacune étant d'usage local au nœud où il se situe. La circulation du Contexte entre Hôtes est quant à elle assurée par des objets autonomes, entités satellites non-assujetties aux contraintes de cycle de vie et d'exécution des Agents, et est donc décentralisée.

Le constat 19 décrit la propriété de plusieurs systèmes contextualisés qui vise à rendre disponible le Contexte en tout temps. Dans l'espace ouvert, offrir cette garantie est pour le moins difficile, pour les mêmes raisons que celles indiquées en lien avec le constat 17 et la dépendance envers des nœuds spécialisés, ce qui explique que ContextAA n'aille pas dans cette direction.

Comme le montrent ces constats, ContextAA a été conçu de manière à éviter toute dépendance envers des nœuds spécialisés. Ceci vise à faciliter l'adaptation aux variations de connectivité et aux caractéristiques pour le moins bigarrées de l'offre de service en milieu ubiquitaire urbain.

Le constat 20 fait état de la capacité qu'ont certaines architectures de permettre la migration par sérialisation d'Agents entre deux nœuds. En ce sens, ContextAA et JADE se ressemblent, à ceci près que les Agents de ContextAA sont représentés dans le format neutre, indépendant de toute technologie tierce, qu'est le Contexte alors que JADE privilégie une forme propre à un langage de programmation spécifique.

3.1.7 Sur le plan de l'ontologie

Le constat 21 positionne l'ontologie dans une acception répandue, celle d'OWL, comme une forme d'offre de services. Du point de vue autonomique de ContextAA, qui préconise le Contexte-selon, ceci est un atout sans être une nécessité. La section sur le cadre ontique des Agents montre ce que nous proposons pour adresser des questions telles que celles pour lesquelles l'ontologie est traditionnellement l'outil de choix, sans exiger de sémantique partagée entre les Agents, et sans aller jusqu'à entraîner une dépendance envers un service de raisonnement spécialisé.

3.2 Espace intelligent ouvert

Nous nommons réalité enrichie la jonction des réalités physique et logique (§2.1). L'espace intelligent est le lieu de la réalité enrichie. Un espace intelligent participe activement à la réalité enrichie et augmente les possibilités des humains, qui y sont situés ou qui ont un intérêt envers les événements qui y surviennent, de même que celles des composants logiciels, qui y gagnent à la fois données sur lesquelles agir et possibilités d'action.

Un exemple d'un espace intelligent est DOMUS, situé à l'Université de Sherbrooke, et présenté à la figure 4.

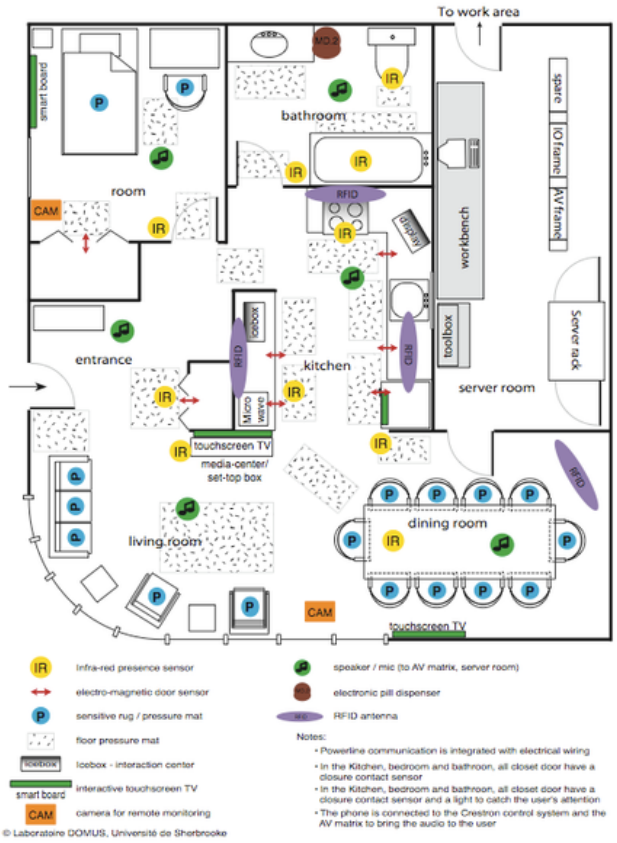


Figure 4 - DOMUS, espace intelligent de la Faculté des sciences, Université de Sherbrooke[61]

Les espaces intelligents traditionnels offrent typiquement une riche gamme de services, mais le font dans un espace délimité. Cette caractéristique fait entre autres des espaces intelligents traditionnels de bons laboratoires, en donnant à qui les administre beaucoup de contrôle sur l'environnement et ses paramètres. Cependant, le nombre croissant d'appareils intelligents mobiles, joint à une connectivité filaire ou sans-fil croissante des appareils fixes, permet aux humains et aux Agents d'aspirer à une mobilité pleine et entière. Avec un espace intelligent traditionnel, le moment où un humain ou un Agent quitte l'enceinte délimitant les frontières de l'espace est susceptible de mener à une discontinuité de service.

L'espace intelligent ouvert est pour nous l'ouverture du concept traditionnel d'espace intelligent vers une conception sans frontières *a priori*, où les individus et les Agents sont présumés libres de leurs mouvements; cette conception fait de chaque espace intelligent traditionnel une zone où l'offre de services est particulièrement riche dans cette entité plus large qu'est l'espace intelligent ouvert.

3.2.1 Conséquences

Pour aborder l'espace intelligent ouvert, nous faisons avec ContextAA un certain nombre de choix porteurs de conséquences architecturales :

---

[61] https://www.usherbrooke.ca/domus/fr/recherche/infrastructure/un-appartement-intelligent/ (consulté le 12 août 2017).

- nous supposons un environnement changeant pour chaque nœud, dans le temps et dans l'espace, du point de vue des Agents et des individus. Nous tenons le voisinage d'un nœud comme susceptible de se transformer, que ce soit parce que le nœud se déplace ou parce que d'autres nœuds mobiles entrent dans son environnement ou le quittent. Pour cette raison, nous proposons un support architectural où un composant Hôte isole les Agents des particularités et des fluctuations du voisinage réseau;
- nous proposons aussi des Agents autonomiques au sens de Kephart et Chess [58], *a priori* indépendants les uns des autres, et capables d'opérer au besoin même si le voisinage réseau est vide. Plusieurs aspects de notre démarche vont en ce sens, incluant la capacité de migration des Agents, le recours à un espace contextuel pour faciliter le raisonnement indépendant sur du Contexte connu localement, et le modèle conceptuel des Agents en tant que tel que nous nommons le Contexte-selon;
- nous supposons des nœuds dont les ressources peuvent varier, dû à la variété des substrats matériels et des plateformes ou pendant leur exécution. Pour cette raison, nous proposons une architecture dont le poids sur les ressources d'un nœud est faible, et des Agents capables de migration si leurs besoins en ressources dépassent ce que le nœud de leur Hôte peut leur offrir;
- nous supposons une hétérogénéité des technologies et des protocoles de communication, comme c'est déjà le cas dans les environnements ubiquitaires urbains. Pour cette raison, nous proposons une architecture orientée Contexte, où les interfaces avec d'autres technologies et d'autres protocoles sont prises en charge par des objets autonomes satellitaires du composant Hôte;
- la variation dans la richesse de l'environnement d'un Agent est un *a priori* de l'espace intelligent ouvert, et le restera du moins tant que la connectivité totale préconisée par la mouvance associée à l'IdO ne sera pas réalisée. Pour cette raison, nous faisons le choix de privilégier la continuité de service, en acceptant de ne pas pouvoir garantir en tout temps le même niveau de qualité de service. Nous proposons aussi un modèle selon lequel chaque Agent applique son propre cadre ontique sur le Contexte, menant à cette évaluation subjective de la qualité du Contexte et de sa pertinence qu'est le Contexte-selon; enfin
- puisque l'informatique ubiquitaire est un domaine pluridisciplinaire, nous supposons des Agents accomplissant des missions déterminées ou définies par des experts de domaines susceptibles de différer de l'informatique. Pour cette raison, nous proposons un modèle de programmation reposant sur le Contexte, et évacuant la complexité de la programmation concurrente et de la communication avec le voisinage réseau du point de vue de chaque Agent.

3.3 Contexte

Comme le montre le survol proposé à §2.2, le sens historique donné au mot « contexte » correspond à l'idée large de « descriptif du réel ». Plusieurs définitions plus précises sont données pour ce terme, en fonction des besoins et des projets, bien que celle de [30] semble la plus répandue, et la plus généralement citée.

La graphie « contexte » peut toutefois être ambiguë dans certaines circonstances, en particulier lorsqu'il y a lieu de distinguer le contexte au sens général comme « circonstances qui accompagnent un fait » [25] de celui qui, dans une acception plus technique, décrit formellement le réel enrichi. Conséquemment, nous distinguons contexte ('c' minuscule), terme pris dans son acception large, de Contexte ('C' majuscule), terme formel pour descriptif de réalité enrichie, ce qui nous permet entre autres de parler d'un Contexte pris dans son contexte.

Nous nommons Contexte brut, ou Contexte inhérent, le fruit du traitement réalisé par les capteurs, et introduit tel quel dans la réalité enrichie, alors que nous nommons Contexte synthétique, ou Contexte émergent, le Contexte résultant de traitement sur du Contexte existant. Nous n'imposons pas de domaine d'application précis pour le Contexte : celui-ci peut représenter les catégories traditionnelles comme le temps, le lieu, le comportement, les relations, etc. et peut s'appliquer à un humain dans son environnement, sans devoir s'y restreindre.

Nous faisons le choix de placer le Contexte au cœur de notre démarche, et d'en faire un descriptif du réel au sens large du terme, capable aussi de décrire des états purement applicatifs, des relations entre les composants logiques, des comportements, des opérations applicables à du Contexte, la mission d'un Agent, un modèle décrivant l'ensemble des Contextes demandés par un Agent, etc. Contrairement aux diverses approches décrites au chapitre 2, le Contexte dans ContextAA est autant un langage (et un métalangage) qu'il est un objet d'expression du réel enrichi.

Le formalisme de ContextAA pour le Contexte est donc un langage à part entière. Il couvre à la fois le Contexte inhérent et le Contexte émergent, sans imposer de contrainte quant aux objets qu'il est susceptible de représenter. Ce faisant, il est possible d'utiliser le Contexte tel que nous le proposons pour des cas de figure que certains pourraient ne pas considérer comme du Contexte, mais nous faisons le choix d'une définition inclusive, que les applications pourront contraindre en fonction de leurs besoins spécifiques. Notre position est que le Contexte pris au sens large ne doit pas dépendre des besoins d'une application contextualisée en particulier, mais qu'il doit plutôt être possible pour cette application d'utiliser le Contexte pour couvrir ses besoins; en particulier, cette position mène sur la possibilité du Contexte-selon (ci-dessous).

3.3.1 Contexte-selon

Nous poursuivons un idéal de représentation « pur Contexte » de la réalité enrichie et des Agents opérant sur celle-ci. Cet idéal comporte plusieurs avantages techniques, incluant la simplicité (un seul format) et la légèreté (un seul moteur de traitement). En ce sens, notre approche au Contexte participe à cette restriction que nous imposons sur le poids que notre architecture impose sur les ressources des nœuds.

L'autonomie des Agents nous amène aussi à proposer un modèle selon lequel la pertinence du Contexte est « subjective », variant selon l'Agent. Concrètement, nous mettons de l'avant un Contexte syntaxique, et c'est sur cette base que nous offrons un support architectural à la publication de Contexte, à la description sous forme de Contexte de requêtes pour du Contexte, et à la mise en correspondance des requêtes de Contexte avec le Contexte publié. Nous reléguons à l'Agent la responsabilité d'apposer une sémantique au Contexte, en acceptant que le Contexte soit alors un Contexte-selon (en anglais, un *Contexte-according-to*).

Le Contexte-selon sémantique propre à chaque Agent est distinct du Contexte syntaxique, ce dernier étant partagé par tous les Agents. Notre modèle permet à un Agent d'œuvrer à sa mission, et de communiquer avec les autres Agents en partageant une grammaire et un alphabet plutôt qu'une Ontologie. En ce sens, notre approche se distingue de plusieurs autres telles que [1, 29], ou [88] par exemple.

Le Contexte syntaxique admet bien sûr des constructions spécifiques. Nous distinguons le Contexte au sens large, qui est en quelque sorte une structure de données, de Contextes individuels, qui en sont des instances. Chaque Contexte sera un cas particulier du Contexte pris dans son acception générale, distinguant « un Contexte » et « le Contexte ».

Chaque Contexte porte un nom unique, qui permet de le distinguer des autres Contextes dans le contexte où il est utilisé. Ce nommage unique est important pour ContextAA : tel qu'expliqué plus en détail à la section §4.3, un Contexte peut être une structure très complexe, pour laquelle un examen en profondeur constitue une opération coûteuse en ressources; la capacité de comparer des Contextes par leur seul nom est une approche économique, qui rejoint les principes architecturaux décrits au chapitre 5.

L'unicité du nom d'un Contexte est atteinte en qualifiant celui-ci par son contexte. En effet, sur un Agent a pris en charge par un Hôte h, le nom d'un Contexte c portant un identifiant id propre à a sera $h{:}\, a{:}\, id$ lorsque c sera examiné par un tiers sur un Hote $h' \neq h$; le même Contexte c sera $a{:}\, id$ lorsque c sera examiné par un Agent $a' \neq a$ sur h; enfin, c sera simplement id lorsqu'il sera examiné par a lui-même. L'exigence d'unicité de c pour a est sous la responsabilité de a, alors que l'unicité globale de c tient à l'unicité de h et à l'unicité de a dans h. L'unicité des noms d'hôtes est donc un *a priori* important pour ContextAA, et peut être assurée par divers mécanismes, incluant le recours à des UUID [49].

Cette approche met en relief une différence entre le contexte dans la littérature, qui est typiquement un descriptif d'un réel complexe, et le Contexte de ContextAA qui est un langage décrivant entre autres le contexte dans son sens plus traditionnel, mais qui mène en plus au Contexte-selon.

Pour illustrer cette différence, supposons un Agent s'exécutant sur un téléphone intelligent et communiquant avec une application tierce à travers Internet pour guider un individu, et supposons une perte de connectivité à Internet alors que l'Agent cherche à réaliser sa tâche. Dans ContextAA, le Contexte-selon l'Agent est la partie du Contexte total qui semble pertinente pour l'Agent en fonction des règles qui sont les siennes, et la non-mise à jour de ce Contexte le force à opérer sur des données qui ne sont plus aussi à jour. Plusieurs options s'offrent à lui, incluant consulter d'autres Agents dans les environs ou passer par d'autres sources de données pour pallier l'irritant, mais ces choix dépendent de l'Agent, et il se peut que la qualité du service offert par l'Agent soit affectée. Toutefois, pour l'opération de l'Agent, le Contexte demeure ce qui est mis à sa disposition, et le service qu'il offre peut se poursuivre dans la mesure de ce qu'il parvient à faire sur cette base.

3.3.2 Modélisation

Tel que relaté dans §2.2.2, la modélisation du Contexte touche habituellement à la fois la syntaxe et la sémantique, mettant souvent l'accent sur la sémantique et insistant sur l'importance d'une ontologie commune aux agents pour automatiser le raisonnement.

ContextAA procède différemment :

- dans ContextAA, les Agents partagent un vocabulaire commun constitué d'un ensemble restreint de mots réservés (annexe A) pour faciliter l'opérationnalisation du Contexte;
- la grammaire du Contexte est commune à tous les Agents. Sa forme est récursive, au sens où un Contexte se définit par une paire {nom,valeur} où la valeur est un ensemble de Contextes, tous uniques dans ce contexte;

- la modélisation faite sur la base de Contexte varie en fonction de l'Agent. Une même idée peut être décrite de plusieurs manières différentes, en utilisant des mots et des unités de mesure variant selon la langue et la culture. ContextAA tient cette variation de structure et de vocables selon l'Agent, le Contexte-selon, pour un *a priori*; nous faisons le choix d'en tenir compte plutôt que chercher à imposer une ontologie commune ou une langue unique;
- ce qu'un Agent tient pour Contexte, donc ce qui est Contexte-selon cet Agent, réside dans ce que nous nommons son cadre ontique. C'est à partir de cette modélisation propre à l'Agent que ce dernier interrogera l'environnement à la recherche du Contexte pertinent pour lui, et c'est aussi à partir d'elle que l'Agent en viendra à choisir, parmi les Contextes correspondant à ce qu'il recherche, ceux qu'il utilisera réellement en pratique;
- pour cette raison, ContextAA met l'accent sur la mise en correspondance automatisée entre le Contexte publié et le Contexte demandé. Ce mécanisme est expliqué à §5.4.2 mais implique la description d'une requête pour du Contexte sous la forme d'un modèle de Contexte, en assurant la mise en correspondance de ces modèles avec le Contexte tel que publié, de même que la description sous forme de Contexte des règles de transformation d'un Contexte à un autre;
- ContextAA définit l'équivalence entre Contextes par l'égalité de leurs noms. Une question connexe est celle de la similitude (et, par le fait-même, de distance) entre Contextes. Cette similitude s'opérationnalise entre autres par le calcul de la distance entre leurs noms (§4.4.6.5); de la distance structurelle entre deux Contextes; de leur distance de contenu; et de leur distance sémantique;
- la modélisation d'un Agent se fait aussi sous forme de Contexte. Nous en discutons plus en détail dans §3.4.

3.3.3 Profil

Une catégorie récurrente de Contexte dans la littérature est celle du « profil » (*User Profile*), ou Contexte descriptif de la réalité de l'usager. Ceci tient en grande partie du fait que plusieurs applications contextualisées placent l'humain au cœur de leurs préoccupations [93], ce qui fait du profil un Contexte d'une importance manifeste.

ContextAA utilise aussi cette dénomination, bien qu'en l'écrivant Profil pour la distinguer d'autres acceptions du même mot, mais de manière informelle : un Agent dont la mission est en lien avec la situation ou le bien-être d'un usager utilisera du Contexte décrivant entre autres son Profil, et tiendra compte de besoins spécifiques à de tels Contextes (le besoin d'en respecter la confidentialité, par exemple).

Le Profil est un Contexte fréquemment rencontré dans les systèmes contextualisés; sur le plan éthique, en particulier, ce Contexte demande qu'une attention particulière lui soit accordée, et il est indéniable qu'il occupe une place de choix dans les tenir-compte d'une vaste gamme d'applications. Sur le plan technique cependant, au sens de ContextAA, le Profil est du Contexte, tout simplement.

3.3.4 Contexte partiel

Dans §2.2, nous mettons entre autres en relief l'immensité du domaine que représente la réalité enrichie. ContextAA prend acte de cet irritant et aborde le Contexte sous un angle subjectif, celui du Contexte-selon. Ceci permet de réduire la cardinalité de l'ensemble des Contextes pour chaque Agent, chaque Agent opérant sur un ensemble à la hauteur de ses besoins, sans plus. Ce qu'impose ContextAA aux Agents n'est pas une sémantique commune, mais bien une syntaxe commune, de même que des outils permettant d'unifier et de transformer les Contextes de chacun pour en arriver à raisonner sur les Contextes malgré ces différences.

Dans une approche où le Contexte est Contexte-selon, le Contexte descriptif d'une réalité commune à tous les Agents serait susceptible d'être aussi complexe que l'union ensembliste des Contextes selon chaque Agent l'observant, tout comme il serait susceptible de contenir des contradictions. Conséquemment, viser une description sous forme de Contexte qui soit commune à tous les Agents pour un réel donné peut mener à un Contexte démesurément lourd. La capacité pour un Agent de ne considérer que les parties convenant à son cadre ontique lui permet de n'opérer que sur un ensemble de Contextes d'une taille convenant à ses capacités et à ses besoins.

Pour faire face aux problèmes propres à l'espace ouvert, un Agent migrant vers un nœud mobile peut transporter avec lui le Contexte se limitant aux parties appropriées pour ses besoins opératoires. Tout Agent peut d'ailleurs agir ainsi, indépendamment de la richesse du nœud qui sous-tend son Hôte, mais cette capacité est surtout importante pour les Agents capables de migration.

Le Contexte-selon de ContextAA est avantageux ici, au sens où il permet à un Agent de restreindre son champ d'intérêt à des parties d'un Contexte autrement plus riche et plus complexe, d'extraire ces parties au besoin, d'opérer sur elles et, éventuellement, de permettre la fusion du fruit de son raisonnement sur ces parties à des Contextes tiers, potentiellement plus riches.

3.4 Agent

Dans ContextAA, nous nommons **Agent** une entité dynamique habitant l'espace logique, distinguant Agent avec un 'A' majuscule pour l'acception technique que nous en faisons d'agent avec un 'a' minuscule pour l'acception générale de ce terme. Sur le plan architectural, ContextAA supporte toutes les familles d'agents mentionnées dans §2.3, mais la vaste majorité des Agents seront au moins des agents orientés vers un but, chacun cherchant à atteindre sa mission.

Dans l'approche mise de l'avant pour ContextAA, un Agent construit un modèle de l'espace intelligent sur lequel il raisonne; pour cette raison, nous disons de lui qu'il est contextualisé. De même, chaque Agent ne considère qu'une fraction du Contexte disponible, guidé en ceci par son cadre ontique, ce qui entraîne la perspective du Contexte-selon qui distingue ContextAA des systèmes pour lesquels une Ontologie est un schème partagé par l'ensemble des agents et permet à un Agent de faire face à l'imposante quantité de Contextes possibles exposés dans l'espace intelligent ouvert.

Avec ContextAA, ce sont des composants Hôtes qui sont déployés sur des nœuds, alors qu'un Agent est déployé sur un Hôte. Sur un Hôte donné, ContextAA trace une distinction entre les Agents qui offrent des services spécifiquement pour leur Hôte, que nous nommons Agents standards, et les Agents qui cherchent d'abord et avant tout à accomplir une mission associée à un domaine d'application précis, que nous nommons Agents du domaine. À titre d'aperçu :

- un Agent standard assure une partie des services de son Hôte et y est lié. Pris ensemble, un Hôte et ses Agents standards jouent dans notre approche le rôle traditionnel d'un intergiciel;
- un Agent du domaine accomplit une mission exprimée sous forme de Contexte et peut, ce faisant, se déplacer d'un Hôte à l'autre.

Sans que ce ne soit une obligation, un Agent du domaine est typiquement décrit strictement par du Contexte, alors qu'un Agent standard tend à être programmé de manière plus conventionnelle. La raison pour cette convention est qu'un Agent standard est associé à un Hôte spécifique et tend à jouer un rôle architectural, analogue à celui de services intergiciels, ce qui explique qu'il bénéficie d'une intégration plus directe à son Hôte, alors qu'un Agent du domaine a un comportement dicté en grande partie par sa mission, et est susceptible de migrer d'un Hôte à l'autre dans cet objectif; conséquemment, ce type d'Agent bénéficie d'une représentation plus flexible et plus dynamique telle que le Contexte. Dans un cas comme dans l'autre, cette façon de procéder convient aux besoins des Agents.

La modélisation d'un Agent se fait sous forme de Contexte et décrit un uplet, incluant le Contexte qu'il a acquis, sous la forme de son espace contextuel, les ressources dont il a besoin pour réaliser sa mission lorsque cela s'avère applicable, son cadre ontique et le modèle de sa mission. Les opérations de mesure de proximité entre Contextes permettent d'établir une métrique quantitative de la continuité de service en fonction de la proportion dans laquelle la mission aura été accomplie au fil du temps.

Nous donnons à l'idée de mission une acception large, comme par exemple :

- assurer l'affichage, dans l'environnement et sur le bon écran, d'informations contribuant à la bonne santé d'un humain;
- consommer les données d'un capteur et les publier, une fois projetées dans un autre référentiel, pour fins de consommation par d'autres Agents;
- consommer des commandes exprimées sous forme de Contexte par un Agent et modifier en conséquence certains paramètres de l'environnement physique à travers l'action d'un actuateur;
- calculer un indice de confort à partir des données produites par d'autres Agents;
- signaler à un tiers que l'indice de confort susmentionné ne convient pas aux besoins décrits par le Profil de l'usager;
- surveiller le comportement d'autres Agents pour détecter des pannes ou des défectuosités; etc.

Les scénarios de tests décrits dans §Chapitre 6 mettent en relief certaines de ces acceptions du concept de mission, sans toutefois prétendre être exhaustifs.

Dans ContextAA, la capacité d'un Agent à participer au succès de la mission des autres Agents importe autant que sa capacité d'offrir un service aux humains. Notre approche soutient tout autant l'un que l'autre de ces objectifs. Comparer le Contexte attendu avec le Contexte, par exemple à l'aide du calcul de la distance sémantique entre eux, obtenu permet de quantifier la continuité de service pour un Agent.

3.4.1 Mobilité

La présomption de mobilité chez les Agents dans l'espace ouvert s'exprime sous deux angles : la mobilité des individus, incluant celle des nœuds mobiles qui les accompagnent, et la mobilité des Agents d'un nœud à l'autre. La mobilité d'un Agent, qui dépend de la connectivité entre les nœuds sur lesquels l'Agent transite ou réside, accroît sa capacité d'accomplir sa mission. Par exemple :

- si un individu se déplace d'une pièce à l'autre à son domicile, et si les nœuds y sont fixes, alors la mobilité de l'Agent lui permet de migrer d'un nœud à un autre et d'accompagner cet individu à l'intérieur des frontières de cet espace;
- si un individu quitte un espace intelligent, alors la mobilité de l'Agent lui permet de prendre avec lui la partie du Profil qui lui semble opportune, puis de migrer vers un nœud mobile qui accompagnera l'individu;
- pendant qu'un individu se déplace, si les ressources d'un nœud faiblissent de manière alarmante, un Agent peut migrer vers un autre nœud accompagnant l'individu, ou situé dans sa périphérie. Il est probable que certains nœuds ne soient pas en mesure d'accueillir un Agent de manière à lui permettre d'accomplir sa mission : à titre d'exemple, si un Agent dépend de la présence d'un accéléromètre sur son nœud pour détecter les chutes, alors seul un nœud mobile offrant ce type d'appareil pourra le satisfaire. Ceci explique que les besoins en termes de ressources d'un Agent fassent partie de sa modélisation;
- lorsqu'un individu fait son entrée dans un espace intelligent traditionnel, un Agent l'ayant accompagné dans ses déplacements peut migrer vers un nœud de ce nouvel environnement pour profiter des ressources potentiellement plus riches qui s'y trouvent.

À la manière de [58], ContextAA préconise pour l'espace ouvert des Agents autonomiques [2], et soutient cette autonomie potentielle par le Contexte, plus particulièrement par l'espace contextuel de l'Agent.

Sur un Hôte donné, un Agent est pris en charge et exécuté dans le respect d'un cycle de vie décrit dans §5.2.5. Ce cycle de vie assure entre autres l'activation et la suspension de l'Agent, de même que des étapes pour consommer, traiter et produire du Contexte. Chacune de ces étapes est optionnelle : un pur consommateur pourrait ne pas produire du Contexte mais activer un actuateur; un pur producteur pourrait intégrer sous forme de Contexte le fruit de la transformation de certains états descriptifs de la réalité physique, par exemple l'état de la pile d'un nœud mobile; enfin, un Agent servant de relais pourrait ne pas réaliser de traitement entre le Contexte consommé et le Contexte publié.

3.4.2 Services

Un Agent offre des services, et le fait en cherchant à accomplir sa mission. L'approche de ContextAA étant « pur Contexte », ces services y sont représentés comme du Contexte; lorsque survient le besoin d'interactions avec des tiers, ce Contexte est traduit en service par un objet autonome (voir §3.5.5 pour un bref descriptif). La continuité de service d'un Agent se calcule par le rapport entre les succès que rencontre l'Agent en cherchant à accomplir sa mission et les tentatives d'y arriver. Parmi les facteurs externes influençant la continuité de service d'un Agent, on trouve entre autres sa résilience et sa capacité de communiquer avec d'autres Agents, incluant sa capacité de migrer vers d'autres Hôtes.

Dans l'espace ouvert, l'Agent doit s'adapter en continu à un environnement hétérogène, ce qui complique le recours à des interfaces statiques et réduit les probabilités qu'une dépendance envers des composants spécialisés permette à l'Agent d'assurer la continuité de son offre de services. En retour, lorsqu'un Agent se trouve dans un espace aussi riche en services et en ressources que le sont les espaces intelligents traditionnels, il est inefficace de ne pas en profiter. Conséquemment, ContextAA permet de profiter des services des composants disponibles dans l'environnement, en particulier dans le contexte d'un espace intelligent traditionnel, mais préconise que les Agents évitent à tout prix de dépendre de nœuds spécifiques pour maximiser leur autonomie.

3.4.3 Mission

Un Agent dans ContextAA vise à accomplir une mission, ce par quoi nous entendons « faire en sorte que le Contexte visible pour l'Agent reflète une situation qui concorde avec un ensemble d'attentes » décrites sous forme de Contexte.

La description sous forme de Contexte des conditions attendues pour que les objectifs de l'Agent soient atteints permet à un Hôte d'évaluer quantitativement le degré d'atteinte de ces objectifs, et par le fait même de quantifier la continuité de service d'un Agent.

3.4.4 Espace contextuel

Dans Jacob et al. [53] apparaît un concept que l'auteur nomme *Context spaces* :

> « *[a] set of nodes[,] that commonly share information[, implying] a bounded locality of context allowing a distinction between local and global Context-aware applications: the local ones only access the contextual information in their context space, global context-awareness denotes the utilization of the global context consisting of all subordinate contexts in the respective context spaces* »[62].

Les *Context spaces* ainsi décrits tiennent le Contexte comme lié à un lieu, et leur intégration vise à permettre la construction d'un portrait global du Contexte pour un ensemble de lieux. Bien que nous n'adhérions pas à une démarche visant à déterminer un état systémique global dans l'espace ouvert, cette idée d'espaces de Contexte a inspiré un élément architectural de notre démarche, soit l'espace contextuel propre à chaque Agent.

Dans ContextAA, un espace contextuel est une zone dans laquelle sont entreposés les Contextes connus d'un Agent ou de son Hôte. L'espace contextuel d'un Agent a s'exprime S_a alors que celui d'un Hôte h s'exprime S_h. La modélisation sous forme de Contexte d'un Agent a est entreposée dans S_a, comme s'y trouvent les Contextes acquis par a, les requêtes pour du Contexte telles que soumises par a, les résultats de la mise en correspondance de ces requêtes avec les Contextes disponibles, etc.

[62] Un ensemble de nœuds partageant de l'information, impliquant un lieu délimité pour le Contexte permettant de distinguer les applications contextualisées locales de celles plus globales. Celles locales n'accèdent que l'information dans leur *espace contextuel*, alors que la contextualisation globale dénote le recours à un Contexte global fait des Contextes qui y sont subordonnés. (Traduction libre; les italiques sont de moi)

Plus précisément, l'espace contextuel d'un Agent est constitué de tous les Contextes qui lui sont directement accessibles; ceci permet à un Hôte donné de mettre à la disposition des Agents qu'il prend en charge des Contextes foncièrement partageables, par exemple des règles de transformation de référentiels, sans avoir à les dupliquer dans chaque espace contextuel. L'espace contextuel S_h d'un Hôte h est l'union ensembliste des espaces contextuels des Agents sous sa gouverne, ce à quoi s'ajoutent les Contextes connus de h seul.

Un Agent du domaine peut ajouter à son espace contextuel et peut consulter son espace contextuel, mais ne peut pas en éliminer lui-même des Contextes. Un Agent du domaine ne peut accéder aux espaces contextuels des autres Agents. Un Agent standard peut quant à lui accéder en lecture et en écriture aux espaces contextuels de tous les Agents du même Hôte.

Les règles d'entretien du Contexte dans l'espace contextuel d'un Agent donné sont exprimées sous forme de Contexte par cet Agent. Ces Contextes sont dits méta-Contextes; les règles qu'ils décrivent sont prises en charge par un Agent standard spécialisé à cette fin, et les Agents du domaine ne les considèrent typiquement pas lorsqu'ils formulent leurs requêtes. En ceci, l'espace contextuel partage certaines caractéristiques du tableau noir proposé par Winograd [115], mais diffère du fait que ce tableau noir est globalement accessible à tous les agents participants alors que l'espace contextuel d'un Agent ne concerne pas les autres Agents du domaine : là où Winograd propose un tableau noire public servant de média de diffusion centralisé pour le Contexte, l'espace contextuel S_a d'un Agent $a \in h$ n'est directement accessible qu'à a et aux Agents standards (§4.7.6) $a' : a' \in h$. Ce qui, dans ContextAA, se rapproche le plus d'un tableau noir public sur un Hôte h donné est la partie de l'espace contextuel de cet Hôte qui n'appartient pas en propre à l'un ou l'autre de ses Agents, ou $S = (S_h \cap \{\bigcup S_a : a \in h\})$ car ce Contexte est à proprement dit partagé entre tous les Agents sur h.

L'espace contextuel accroît l'autonomie d'un Agent. En effet, étant isolé du voisinage réseau, un Agent a ne connaît que ce que contient S_a, et agit chaque fois sur cette base. Ainsi, S_a devient la lorgnette par laquelle a observe le Contexte. Puisque S_a est mis à jour à un rythme indéterminé, en fonction du contexte, du voisinage réseau, du Contexte disponible, etc., l'Agent a prend à chaque fois qu'il s'exécute le contenu de S_a comme étant le Contexte mis à sa disposition, et opère sur cette base. Conséquemment, même si les circonstances ou la connectivité ont pour conséquence d'empêcher la mise à jour de S_a, a peut poursuivre ses opérations à partir du Contexte à sa disposition. De même, une autre conséquence de la mobilité d'un Agent est qu'il est susceptible d'être parfois mis en contact avec une grande variété d'Agents, et d'être essentiellement isolé de tous les Agents hors de son Hôte à d'autres moments. Cependant, pour a, il n'y a guère de différence entre ces deux situations, les interactions directes se limitant avec S_a qui est une ressource locale à son Hôte. Plus de détails sur la gestion de l'espace contextuel d'un Agent sont donnés à §4.7.5.

3.4.4.1 Visibilité du Contexte

Un Contexte peut être public, local ou privé. Ces qualifications constituent ce que nous nommons la visibilité du Contexte.

La visibilité du Contexte délimite ce à quoi un Agent a accès lorsqu'il soumet une requête pour du Contexte, et permet du même coup à l'Agent de réduire la portée de ses recherches. En pratique, dans ContextAA, un Agent du domaine ne réalise qu'indirectement les recherches associées à ses requêtes : toute requête pour du Contexte est prise en charge par un Agent standard ou par une fédération de tels Agents. Ainsi :

- une requête pour du Contexte privé permet à a de parcourir S_a, donc de limiter la portée de sa fouille au Contexte qu'il a lui-même amassé;
- une requête pour du Contexte local permet à $a \in h$ de parcourir S_h. Parmi les Contextes pertinents à h du point de vue de a, on trouve les caractéristiques de la plateforme sous-jacente, les services de traduction de Contexte disponibles, le contenu du voisinage réseau N_h, etc.;
- une requête pour du Contexte public permet à a de parcourir les Contexte publiés dans $\forall h' \in N_h$. Les requêtes pour du Contexte de l'environnement, incluant les données captées du monde physique et rendues disponibles par les Agents de niveau L_0 (§3.5), sont typiquement dans cette catégorie.

Dans la même veine, un Contexte privé dans $S_a, a \in h$ n'est accessible que par les Agents ayant accès à S_a, soit a et les Agents standards sur h. Un Contexte local dans $S_a: a \in h$ n'est accessible qu'à a et à $\forall a' \in h$. Enfin, un Contexte public dans S_h est accessible à $\bigcup(a \in h, a' \in N_h)$.

Le recours aux qualifications de visibilité dans ContextAA est jusqu'ici une convention. Une requête explicitant le souhait d'accéder à du Contexte qualifié de privé ou de local pourrait obtenir ce Contexte si elle était exprimée directement sur un Hôte donné. Ce qui réduit le risque de cette pratique est que l'essentiel des échanges de Contexte faits entre Hôtes passe par la médiation d'Agents standards, qui respectent les règles établies.

3.4.5 Cadre ontique

Sous ContextAA, l'ontologie en tant que formalisme partagé laisse sa place à un concept local à l'Agent, que nous nommons cadre ontique pour marquer la différence entre les deux concepts. En pratique, ainsi, une formule comme « l'ontologie » devient « son cadre ontique », ou « le cadre ontique de cet Agent ». ContextAA ne suppose au préalable aucune ontologie commune, sans exclure le recours à une ontologie partagée par un groupe d'Agents sur une base volontaire si un tel outil leur semble à propos, privilégiant le partage de cette structure d'abord grammaticale qu'est le Contexte et de mécanismes d'unification pour deux instances de cette structure.

Pour ContextAA, il importe que le cadre ontique soit partageable, donc que ceux de deux Agents puissent être mis en correspondance de manière à leur permettre de partager du Contexte sur lequel chacun pourra opérer et raisonner; pour cette raison, le cadre ontique d'un Agent se décrit lui aussi sous forme de Contexte.

ContextAA ne suppose pas qu'une ontologie commune soit une nécessité. En retour, ContextAA exige qu'il soit possible de décrire les besoins et les propositions de deux Agents sous forme de Contexte, dans le respect de leurs cadres ontiques respectifs, pour automatiser la mise en correspondance de ces besoins et ces propositions. Dans ContextAA, le cadre ontique est une entité opératoire subjective, élément clé du Contexte-selon : ce qui décrit les objets d'intérêt, de même que les relations entre ces objets, mais du point de vue d'un Agent particulier plutôt qu'à partir d'une perspective globale.

Ne pas recourir à une ontologie globale partagée réduit la dimension du problème adressé par chaque Agent dans la poursuite de sa mission aux Contextes d'intérêt pour lui : chaque Agent n'a qu'à considérer un sous-ensemble restreint des Contextes possibles, soit ceux qui sont décrits par son cadre ontique. Ceci rejoint l'objectif de frugalité en termes de consommation de ressources pour chaque nœud que ContextAA vise à atteindre : un Agent ne dépend pas d'un spécialiste externe du raisonnement, et peut se limiter à ne transporter que ce qui permet d'identifier le Contexte pertinent pour lui.

En résumé, le positionnement de l'ontologie comme un formalisme partagé sur un domaine d'intérêt commun se décline dans ContextAA comme des formalismes individuels dans la langue commune du Contexte, à partir de domaines d'intérêt distincts permettant à un Agent d'atteindre sa mission. L'ontologie, si on peut la nommer ainsi, émerge en quelque sorte implicitement plutôt qu'explicitement, par échange de Contextes et mise en correspondance de ce qui est publié et de ce qui est demandé.

3.4.6 Contextualisation

La contextualisation, décrit le comportement émergeant de la collaboration entre les composants matériels et logiciels dans un espace intelligent. Avec ContextAA, par conséquent, cette collaboration utilise du Contexte, et les événements n'y sont pas traités isolément mais bien considérés dans leur contexte. Le Contexte est un outil, alors que la contextualisation est un objectif : le Contexte permet de représenter ces données complexes et inter-reliées qui décrivent le réel enrichi, de manière à ce qu'elles puissent être consommées, traitées et produites par des Agents et, par le fait même, laisser émerger la contextualisation.

Pour notre démarche, les principaux composants contextualisés sont les Agents. Un Agent contextualisé peut jouer un rôle de consommateur de Contexte, de fournisseur de Contexte, ou une combinaison des deux, étant dépendant du Contexte au sens de Neovius et al. [71]. Dans ContextAA, tout Agent utilise le Contexte pour accomplir ses tâches, comme en fait foi sa représentation contextualisée.

En effet, notre approche avec ContextAA représente l'Agent sous forme de Contexte : sa mission M, son cadre ontique O, ses besoins en ressources R et son espace contextuel S. Étant Contexte, l'Agent est lui-même objet d'analyse pour un Agent, incluant ce que l'on pourrait qualifier de capacité d'introspection; que l'Agent puisse raisonner sur lui-même nous rapproche de l'idéal autonomique de Kephart et Chess [58] : l'Agent raisonne sur du Contexte dans le cadre de ses fonctions, il est lui-même Contexte, et devient par le fait-même capable de réflexivité, d'autodiagnostic, et d'adaptation à sa propre condition.

À titre d'exemple des avantages que comporte la contextualisation dans ContextAA, notons qu'en évaluant la distance contextuelle entre les objectifs de sa mission, M, et ce qui a été réalisé, logé dans S, l'Agent peut par introspection quantifier la proportion dans laquelle ses objectifs sont atteints, et quantifier sa propre continuité de service [94]. La continuité de service étant l'un des objectifs à la base de la démarche que sous-tend ContextAA, cette propriété nous semble particulièrement pertinente.

3.5 ContextAA – Architecture

ContextAA trouve ses racines dans une démarche plus vaste d'architecture multi-niveaux, $L_0..L_n$, intégrant les modèles de contextualisation micro et macro [1], tout en étant capable de se déployer dans le contexte plus large de l'espace intelligent ouvert.

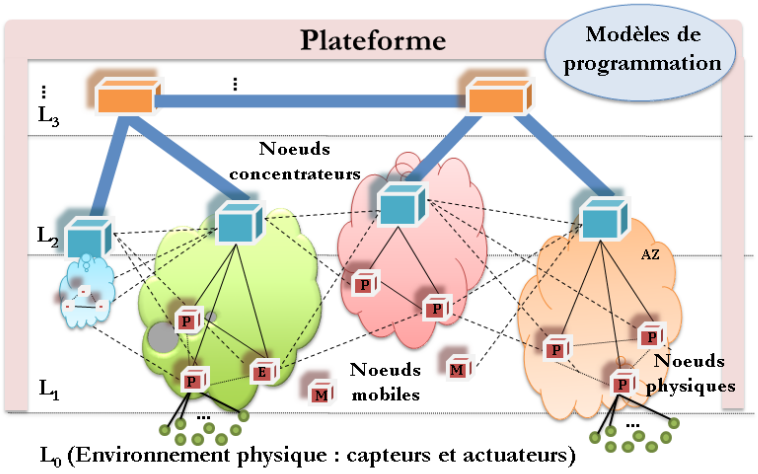


Figure 5 - Architecture globale soutenant à l'origine ContextAA

La figure 5 montre une vue à haut niveau de cette architecture originelle d'un point de vue macro, à partir de nœuds conçus sur mesure, qu'ils soient fixes ou mobiles. Les nœuds de niveau L_1 sont des abstractions de leur substrat matériel permettent d'interagir avec les nœuds du niveau L_0, ceux-ci correspondant aux capteurs et aux actuateurs. ContextAA intervient à chaque niveau de cette architecture, pour faciliter la conception d'applications sur la base de Contexte.

L'architecture en question se décline en zones, régions logiques d'influence regroupant des services sur la base d'un critère. Chaque nœud $L_i, i > 1$ sert de concentrateur pour les services offerts par les nœuds de niveau inférieur, $L_j, j < i$, dans une zone donnée. Les critères de regroupement de services dans une zone incluent : le partage d'un lieu commun (p. ex. : une pièce ou un étage d'un batiment), les besoins de traitement (p. ex. : un nœud $L_i, i > 1$ regroupe les services de nœuds L_{i-1}; un nœud participe à un algorithme de répartition de charge), des préoccupations de sécurité, de confidentialité ou d'éthique, etc. Deux zones peuvent se chevaucher; leur intersection peut être non-vide.

Cette architecture permet des scénarios hiérarchiques, où un appareil de niveau L_0 collecte des informations de divers capteurs pour en faire du Contexte. Ce Contexte peut alors être consommé par des appareils de niveau L_1 pour générer des fonctionnalités contextualisées en lien avec un composant et son voisinage : du Contexte brut. Par la suite, des nœuds de niveau L_2 regroupent des micro Contextes [9] saisis d'appareils de niveau L_1, construisant par le fait-même une vision plus globale du Contexte dans la zone, et menant à une contextualisation macro [1]. Enfin, aux niveaux L_3 et plus, les appareils regroupent le Contexte de niveaux inférieurs pour contribuer à la construction d'une vision plus globale de l'environnement.

ContextAA est conçu pour profiter de cette architecture originelle, de l'organisation inhérente de ses services et de la richesse du Contexte qu'elle peut mettre à sa disposition lorsque cela s'avère possible, tout en demeurant indépendante de cette architecture, au sens où ContextAA se compose simplement d'un système diffus de composants Hôtes interagissant à travers des objets autonomes et servant d'hôtes pour des Agents contextualisés.

C'est d'ailleurs là un objectif fondamental de ContextAA : savoir profiter d'un milieu riche en ressources et en services (approche « macro ») tout en étant capable de fonctionner dans un milieu plus « pauvre » en ressources et en services (approche « micro ») [2].Ainsi, dans un environnement riche, à l'image de celui décrit dans la figure 5, ContextAA profitera de l'ensemble des ressources disponibles, alors que dans une région plus pauvre en services que celles soutenues par l'architecture originelle décrite ici, il suffit de déployer des composants Hôtes sur des nœuds capables de communication, et idéalement d'en faire profiter quelques Agents du domaine, pour que ContextAA fonctionne.

3.5.1 Hôte

Que ce soit sur les nœuds de l'architecture originelle ou sur des nœuds pris dans un sens plus large, ContextAA déploie des composants spéciaux, nommés Hôtes pour les distinguer du terme général « hôte », ayant pour rôle de prendre en charge le cycle de vie des Agents et de leur offrir les services dont ils ont besoin. Un Agent est déployé sur un Hôte, et un Hôte est déployé sur un nœud dans l'environnement.

Les services qu'offre un Hôte à chacun de ses Agents incluent :

- la prise en charge de son cycle de vie;
- la représentation sous forme de Contexte des ressources de son nœud;
- la représentation sous forme de Contexte du voisinage réseau;
- protéger l'accès à son espace contextuel;
- l'isoler des risques associés à la concurrence;
- résoudre les mises en correspondance de modèles avec le Contexte disponible; et
- assurer la circulation du Contexte entre les Agents comme entre les Hôtes.

Concrètement, sur un Hôte donné, ces services sont implémentés en partie par l'Hôte lui-même, mais surtout par une fédération d'objets autonomes (§3.5.5) et d'Agents standards (§4.7.6).

3.5.2 Voisinage réseau

Parmi les principaux rôles d'un Hôte, on trouve ceux de transiger avec les Hôtes avoisinants et de permettre aux Agents de migrer d'un Hôte à l'autre au besoin. Pour cette raison, chaque Hôte h tient à jour à l'interne, sous forme de Contexte, une description du voisinage réseau de h, notée N_h. Seuls les nœuds directement accessibles par h sont considérés dans la construction et la mise à jour de N_h, pour éviter une explosion de sa cardinalité par voie de transitivité.

Pour h, N_h représente non seulement l'ensemble des nœuds directement avoisinants, mais aussi l'ensemble des Contextes accessibles à l'instant t pour lequel N_h est constaté par h, incluant l'ensemble des Agents pouvant participer à la résolution de requêtes soumises par des Agents sur h.

La mobilité présumée des nœuds, des Agents et des individus a pour conséquence directe de faire varier le contenu de N_h au fil du temps. Pour alléger l'écriture, on dira des Agents sur un Hôte donné que ce sont « ses » Agents, du moins pendant qu'ils sont placés sous sa gouverne. Les fluctuations de N_h expliquent qu'il s'agisse d'une entité construite et tenue à jour de manière dynamique par h.

Un Hôte h isole les Agents qu'il prend en charge de N_h et permet à ces Agents de n'opérer que sur la base de Contexte mis à leur disposition dans leurs espaces contextuels respectifs (§4.7.5). De cette manière, un Hôte isole ses Agents du voisinage réseau et participe à simplifier leur conception.

3.5.3 Nœuds mobiles

Dans la figure 5 apparaissent aussi des nœuds mobiles. Ces nœuds se prêtent à une approche micro de la contextualisation [2] : étant mobiles, ils sont présumés capables de se déplacer d'une zone à l'autre, entre autres dû aux déplacements des individus.

En se déplaçant dans l'espace et dans le temps, un Agent déployé sur un Hôte h occupant un nœud mobile voit N_h varier. Ceci met en relief une distinction fondamentale entre les systèmes macro, pensés pour profiter des espaces intelligents traditionnels, et les « systèmes » micro, qui ont pour cible l'espace intelligent ouvert : dans une approche macro, un Agent fait face à un système dont les composants sont en général disponibles, et occupent des rôles bien définis; en contrepartie, dans une approche micro, les rôles sont définis un Agent à la fois, et la perspective globale est remplacée par une perspective locale à chacun des Agents.

ContextAA, en adressant de manière générale la problématique des nœuds mobiles dans la perspective propre des questions propres à l'espace intelligent ouvert, suit une approche micro pour l'essentiel, tout en s'intégrant au modèle macro lorsque l'environnement le permet.

3.5.4 Architecture détaillée

ContextAA est donc une architecture logicielle répartie, dont nous décrivons sommairement ici les principes et principaux éléments constitutifs. Pour les besoins de notre description, nous tenons pour acquis que, pour participer à ContextAA :

- tout nœud peut être distingué des autres nœuds, par exemple à l'aide d'un identifiant qui lui est propre;
- les ressources d'un nœud sont finies;
- tout nœud peut accepter qu'on y déploie un Hôte si ses ressources le lui permettent;
- un nœud acceptant un composant Hôte est qualifié de « nœud participant »;
- la connectivité entre nœuds participants n'est pas garantie en tout temps, mais lorsqu'elle est possible, un nœud peut communiquer avec les autres nœuds qui lui sont accessibles.

Sur le plan architectural, les éléments clés de ContextAA (présentés à la figure 6) sont :

- un composant Hôte est déployé sur un nœud. Cet Hôte joue pour ContextAA un rôle analogue à celui d'une machine virtuelle : il offre un environnement d'exécution pour les Agents, et il isole ces derniers (en particulier les Agents du domaine) du matériel comme du réseau;

- sur un Hôte sont logés *a priori* des Agents que nous nommons Agents standards. Chaque Agent standard est présumé disponible sur tout Hôte, et offre pour lui une gamme de services de base. Par conception, il est peu probable qu'un Agent standard ait recours aux mécanismes de migration d'Agents, sauf peut-être pour fins de mises à jour;
- sur un Hôte sont aussi logés des Agents du domaine. Un Agent du domaine vise à accomplir une mission, dépend d'un certain nombre de ressources, et opère dans le respect d'un cadre ontique décrivant le Contexte pertinent à ses fins, les modalités par lesquelles il « raisonne » sur ce Contexte, les opérations qu'il y applique, etc. Chaque Agent du domaine a pour principal objectif d'accomplir sa mission, et peut migrer d'un Hôte à l'autre pour y arriver si cela semble opportun;
- en périphérie de chaque Hôte se trouvent des objets autonomes. Un objet autonome est une entité s'exécutant concurremment avec l'Hôte lui-même et prenant en charge des tâches qui ne conviennent pas à une intégration dans le cycle d'opération des Agents. En effet, l'exécution des Agents s'inscrit dans un cycle pris en charge par l'Hôte. Pour cette raison, certaines tâches non-cycliques (p. ex. : des entrées/ sorties bloquantes) sont mieux prises en charge par des entités dont l'exécution est concurrente à celle de l'Hôte;
- seul un Agent standard peut interagir avec un objet autonome. Cette interaction passe sous forme de Contexte par un média synchronisé nommé canal de Contexte.

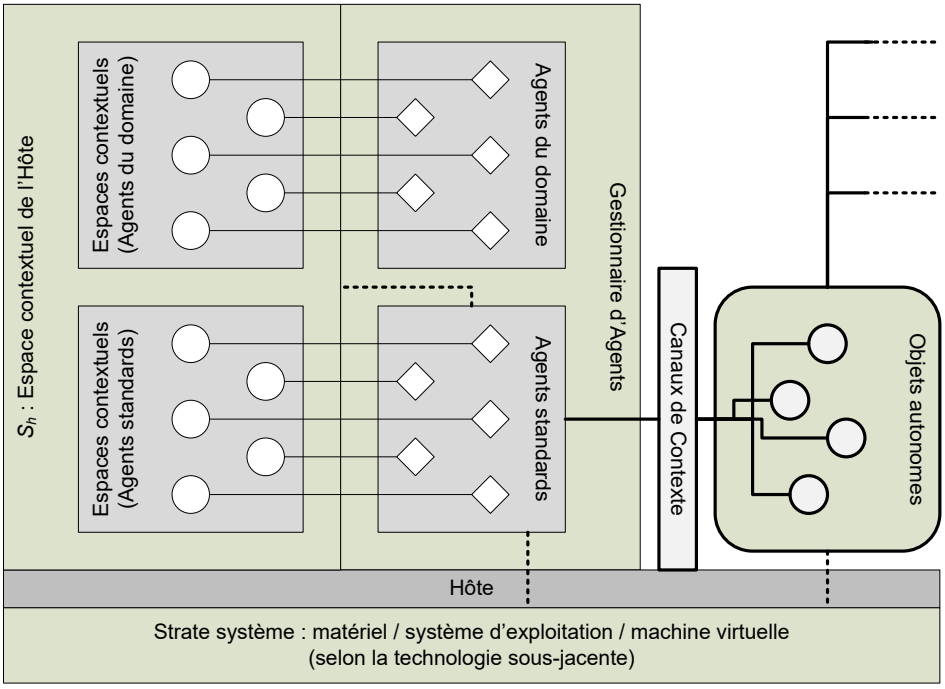


Figure 6 - Vue par strates d'un composant Hôte

3.5.5 Objet autonome

Un objet autonome est une entité dynamique qui réalise des opérations ne pouvant raisonnablement pas être faites par des Agents standards, typiquement parce que cela briserait le rythme attendu du cycle de vie des Agents. Ces opérations incluent en particulier les entrées/ sorties, surtout si celles-ci sont bloquantes, de même que des traitements susceptibles de prendre un temps arbitrairement grand à se compléter une fois entrepris.

Un objet autonome est représenté comme une entité satellite de son Hôte, les deux s'exécutant en parallèle. La forme que prend l'unité de traitement sous-jacente à un objet autonome d'un Hôte h dépend de la plateforme sur laquelle h est déployé et de la technologie avec laquelle il a été conçu.

L'interaction entre un Hôte et chacun de ses objets autonomes se limite à des moments précis (démarrage, arrêt) de leur existence, de même qu'à des échanges de Contexte entre un objet autonome et un Agent standard par la voie de canaux de Contexte (§3.5.6). De par leur forme modulaire, les objets autonomes jouent aussi un rôle d'interface avec des technologies tierces, communiquant dans le respect des métaphores de cette technologie d'une part, et par voie de Contexte avec son Hôte d'autre part.

3.5.6 Canal de Contexte

Pour que les Agents soient isolés des préoccupations associées à la programmation concurrente, ContextAA fait en sorte que les communications entre un Agent et un objet autonome ne soient possibles que si l'Agent est standard, pas du domaine, et si ces communications transitent par un canal synchronisé prévu à cet effet.

Nous nommons canal de Contexte un tel canal, et seul du Contexte y circule. Un canal de Contexte est un média unidirectionnel, au sens où il n'admet qu'un seul producteur et un seul consommateur, permettant l'ajout et la consommation concurrente de Contextes.

Restreindre les communications entre objets autonomes et Agents standards à des échanges de Contexte permet à un Hôte de n'opérer que sur un seul type de données et sur un seul format de représentation. Le fait que les canaux de Contexte soient synchronisés participe aussi à la simplification de la programmation d'Agents, éliminant pour eux les problèmes associés à la concurrence. Une conséquence de ce choix architectural est que lors d'interactions avec des technologies tierces, la transformation en Contexte ou du Contexte qui permet à ContextAA d'échanger avec ces tiers-partis doit se faire dans des objets autonomes.

ContextAA préconise que les Agents opèrent à toutes fins pratiques dans un contexte monoprogrammé et n'opèrent que sur du Contexte. Un canal de Contexte est, pour un Hôte donné, le lieu d'où proviennent les données externes, transformées au besoin en Contexte, et où est aiguillé le Contexte destiné à être envoyé tel quel ou transformé par un objet autonome en un tiers format.

3.5.7 Coordination et collaboration entre Agents

De manière générale, du fait qu'il n'est pas possible d'y garantir en tout temps l'accès à un nœud spécifique, l'espace ouvert se prête mal aux stratégies de coordination inter-Agents reposant sur un orchestrateur, à moins que celui-ci ne soit déterminé dynamiquement et soit remplaçable au besoin. Il existe des exceptions à ce constat : lorsqu'un Agent peut prendre des décisions sans consensus, par exemple, ou lorsque les Agents à coordonner sont tous dans un lieu offrant une richesse de service telle que celle offerte dans un espace intelligent traditionnel.

Pour ces raisons, ContextAA n'offre pas de soutien spécifique à un mode de coordination particulier, mais assure les mécanismes de communication permettant aux Agents d'utiliser un mode de coordination ou l'autre. En dissociant les Agents de la communication directe avec le voisinage réseau, la métaphore que propose ContextAA repose sur une modalité de questions et de réponses exprimées par du Contexte; si un groupe d'Agents implémente un processus décisionnel orchestré ou chorégraphique sur cette base, cela lui appartient et n'est pas du ressort direct de ContextAA.

3.6 Dynamique de contextualisation

ContextAA propose un modèle massivement décentralisé d'Agents contextualisés accomplissant chacun une mission, et n'interagissant que sur la base de Contexte. Ce Contexte est toujours le fruit d'une transformation effectuée par un Agent, qu'il s'agisse d'un composant logiciel simple sur un capteur, d'un composant complexe sur un serveur muni d'importantes ressources matérielles, ou d'une entité mobile migrant d'un appareil à l'autre en fonction du contexte.

L'Agent, en tant que lieu de génération du Contexte et en tant qu'acteur privilégié de cette génération, introduit une « subjectivité » dans le processus de production et de consommation de Contexte; c'est ce que nous nommons le Contexte-selon : ce que fait chaque Agent à partir du Contexte avec lequel il est mis en contact dépend de l'Agent, de son espace contextuel, de sa mission, de son cadre ontique, etc. et varie donc d'un Agent à l'autre. Rien n'est tenu pour acquis quant au comportement *a priori* d'un Agent du domaine dans notre modèle, outre le fait qu'il soit dépendant du Contexte.

Un Agent agit de prime abord sur ce qu'il tient pour Contexte à partir de l'information à sa disposition, et produit du Contexte sur cette base. Conséquemment, le Contexte consommé par un Agent est considéré comme une proposition parmi plusieurs, et chaque Agent est responsable de faire des choix quant à la valeur relative de Contextes distincts décrivant des réalités connexes, d'en évaluer « subjectivement » la qualité en fonction des critères décrits par son cadre ontique.

3.6.1 Gestion du Contexte

La gestion du Contexte se décline en deux catégories, soit la gestion de l'accès au Contexte et la gestion de sa durée de vie, qui est aussi associée directement à la gestion de la croissance de la charge sur les ressources d'un Hôte donné.

L'accès au Contexte par un Agent se fait sur la base de publication de Contexte, de consommation de Contexte et de mise en correspondance entre les deux. Tout Agent peut publier ou consommer du Contexte, et ce que décrit chaque Contexte dépend de l'acception qu'en font respectivement le producteur et le consommateur. Pour contrôler l'exposition du Contexte aux autres Agents, ContextAA utilise les qualifications d'accès au Contexte. Par qualifications d'accès au Contexte, nous entendons le recours à un mot contextuellement réservé pour indiquer qui a droit d'en explorer la valeur.

Comme mentionné précédemment, la visibilité du Contexte (public, local ou privé) influence le processus de résolution automatisé des requêtes pour du Contexte. Pour ce qui est de la gestion de la durée de vie du Contexte, il se trouve que les Agents dans ContextAA consomment et produisent du Contexte sous plusieurs formes, et ce continuellement. Ainsi, si aucun mécanisme n'était mis en place pour contrôler la durée de vie du Contexte, l'espace requis pour l'entreposer croîtrait sans cesse.

Cette gestion se fait à l'aide de ce que nous nommons des méta-Contextes. Les méta-Contextes sont des Contextes destinés à informer les Agents standards appropriés de règles quant à la gestion du Contexte; il existe par exemple des méta-Contextes pour expliquer quels Contextes oublier, ce qui permet par exemple à un Agent qui publie une nouvelle version d'une requête pour du Contexte de demander la suppression des versions antérieures de cette requête.

3.6.2 « Contexte-selon », ou évaluation subjective de la qualité du Contexte

La mobilité potentielle des Agents dans l'espace intelligent ouvert amène chaque Agent à faire des choix, à discriminer entre plusieurs Contextes susceptibles de répondre à ses besoins.

Un Hôte offre aux Agents qu'il prend en charge un service de mise en correspondance des requêtes et du Contexte disponible. Rappelons que cette mise en correspondance est automatisée par l'action d'un Agent standard, et que son résultat est placé dans l'espace contextuel du demandeur. Une conséquence de cette dynamique de contextualisation est qu'il se peut que le demandeur n'obtienne aucun Contexte correspondant au modèle demandé, tout comme il se peut qu'il en obtienne plusieurs.

Placé devant deux Contextes susceptibles de répondre à ses besoins, un Agent peut discriminer sur la base de plusieurs critères. Par exemple :

- préférer le Contexte le plus récemment produit selon lui;
- préférer le Contexte dont le producteur prétend avoir réalisé la lecture de plus haute précision;
- préférer le Contexte dont le producteur est le plus proche du point où se trouve l'Agent;
- préférer le Contexte dont le producteur a meilleure réputation;
- préférer une mesure synthétique construite à partir d'un amalgame des Contextes reçus, comme par exemple la moyenne des températures décrites; etc.

Cette capacité de discriminer, que nous nommons évaluation subjective de qualité du Contexte, est au cœur du Contexte-selon, et les critères permettant de préférer un Contexte ou l'autre s'expriment, dans ContextAA, par du Contexte. L'évaluation subjective de la qualité du Contexte dépend d'autres facteurs, plus contextuels : la crédibilité apparente de l'émetteur, par exemple, qui peut être basé sur l'historique des interactions avec cet Agent (par le demandeur ou par d'autres Agents), ou encore la qualité de l'algorithme utilisé pour produire le Contexte, quand celui-ci est connu. Les critères effectifs de qualité d'un Contexte donné dépendent de l'observateur, donc de l'Agent demandeur. Les mécanismes qu'offre ContextAA pour évaluer la distance entre deux Contextes constituent un exemple d'outil permettant de réaliser certaines de ces évaluations en fonction des règles décrites dans le cadre ontique de l'Agent.

Pour un Agent sur un nœud muni d'actuateurs, il est probable que les requêtes décrivent des commandes qui lui sont destinées et que l'Agent transforme en actions. L'Agent placé face à un ensemble de requêtes discrimine parmi les Contextes obtenus; il peut trancher pour l'un ou l'autre des demandeurs, tout comme il peut soumettre des demandes supplémentaires aux fournisseurs de commandes pour l'actuateur, dans le but de trouver une action convenant à tous les demandeurs sans contrevenir aux contraintes décrites dans le Profil des usagers impliqués. ContextAA offre un cadre pour faciliter ces échanges, mais le processus décisionnel de l'Agent s'inscrit dans le Contexte-selon, et dépend aussi des règles décrites dans son cadre ontique.

Chapitre 4
ContextAA – Modèle théorique

« Ce n'est pas le moindre charme d'une théorie que d'être réfutable. » (Friedrich Nietzsche / Par-delà le Bien et le Mal)

Alors que le précédent chapitre présente la conception générale de ContextAA, celui-ci décrit le modèle théorique qui sous-tend notre démarche. Nous y présentons notre formalisme, les règles qui en guident l'écriture, et les aspects conceptuels sur lesquels repose l'implémentation de notre architecture de manière générale.

Ce chapitre débute par un bref survol de vocabulaire pour des termes « généraux », suite à quoi y sont couverts :

- les conventions de notation adoptées ici et le sens qui leur est donné;
- le formalisme et les opérations propres au Contexte de même qu'aux noms de Contexte dans ContextAA;
- les opérations possibles sur le Contexte, qu'on parle d'opérations au sens large, de transformations, de critères, d'algorithmes, etc. Nous verrons au passage comment il est possible de réaliser une composition d'opérations;
- les questions de similitude entre Contextes, de mise en correspondance de Contextes (§5.4.2) et de distance entre Contextes; et
- le formalisme propre aux Agents, avec un regard particulier sur le cadre ontique.

Ce chapitre ne couvre pas en détail les caractéristiques architecturales de ContextAA, celles-ci étant plutôt décrites et expliquées au chapitre 5. De même, la justification des choix architecturaux faits pour ContextAA se retrouve principalement au chapitre 3.

4.1 Vocabulaire de base

Le design de ContextAA repose en partie sur quelques éléments généraux de vocabulaire, décrits pour l'essentiel dans les sections suivantes. Cette section est plus courte que §4.2, qui discute de conventions de notations plus spécifiques, et ne vise qu'à éclaircir le propos dans son ensemble.

4.1.1 Invariants, préconditions et postconditions

Les types clés de ContextAA documentent leurs opérations en termes d'invariants, de préconditions et de postconditions. Ces termes ont une acception technique répandue sans être universelle, ce qui explique que nous ayons fait le choix de les définir ici. Ainsi, dans ce qui suit :

- un invariant est une condition qui prévaut avant et après chaque opération sur une instance d'un type donné. Par exemple, un invariant du Contexte est qu'il ne peut avoir à la fois un nom vide et une valeur non-vide. Ces invariants sont garantis en pratique par encapsulation dans ContextAA;

- une précondition d'une opération est une condition qui doit s'avérer avant d'amorcer cette opération, et qui doit typiquement être respectée par son client (l'appelant d'une fonction ou d'une méthode), sans quoi le programme se trouve dans un état incorrect. ContextAA ne dicte pas *a priori* le comportement d'un programme dans lequel les préconditions ne seraient pas respectées, comme par exemple des cas d'erreurs ou des états indéfinis;
- une postcondition décrit les effets observables suite à l'exécution d'une opération. Ces effets peuvent être simples, dans le cas d'une opération sans effet de bord, ou complexes si des états systémiques en sont affectés.

Sur le plan strict, le non-respect d'une précondition ne mène pas tant à un cas d'erreur qu'à un comportement indéfini. La position que nous adoptons ici est d'exprimer le non-respect d'une précondition comme une erreur lorsqu'il est possible de valider cette précondition en pratique, le tout dans une optique de clarté.

4.1.2 Opération primitive

Une opération est considérée primitive si elle peut être tenue pour acquise sans devoir être décrite ici dans le détail. Par exemple, la concaténation de chaînes de caractères est primitive au sens entendu ici, tout comme l'est l'addition d'entiers. En pratique, bien sûr, ces opérations ne sont pas nécessairement simples ou évidentes, mais il est présumé que les tenir pour acquises ne posera pas obstacle à la compréhension du texte.

4.2 Conventions et notation

Un certain nombre de conventions sont appliquées dans ce chapitre. Ce qui suit en explique les conventions et les *a priori* qu'elles impliquent.

4.2.1 Considérations générales

Les règles lexicales et grammaticales du Contexte tel que le définit ContextAA sont présentées à l'annexe A.

Nous utilisons le terme « standard », décrivant par exemple des Agents standards, des critères standards ou des algorithmes standards. Ce faisant, nous ne faisons pas référence à un standard dont serait responsable un organisme tel qu'OMG ou ISO mais bien à des outils standards à l'intérieur de ContextAA, donc qui y sont considérés comme des *a priori* systémiques.

Nous écrivons $(a, b, \ldots, c)$ pour lister les symboles $a, b, \ldots, c$. Cette notation servira aussi pour lister les paramètres formels d'une fonction (§4.2.8).

Nous écrirons $[a..b]$ pour décrire un intervalle continu de valeurs allant de a à b inclusivement. Les crochets [et] interviendront aussi dans les notations de tableaux et de listes (§4.2.7); le sens à leur attribuer devrait être clair à partir de leur contexte d'utilisation.

Pour un ensemble S, l'écriture $|S|$ signifie « cardinalité de S ».

Pour un nombre x, l'écrire $|x|$ signifie « valeur absolue de x ».

Nous écrivons Contexte, Agent et Hôte lorsque nous mettons de l'avant leur acception technique, distinguant cette écriture de celle que nous utilisons pour ces termes dans leur acception générale, soit contexte, agent et hôte.

Nous utilisons les néologismes « contextifier » (verbe) ou « contextification » pour exprimer la transformation d'une donnée d'un certain type en son équivalent sous forme de Contexte. Une entité qui peut être représentée sous forme de Contexte, donc contextifiée, est dite « contextifiable ». Un Agent du domaine (§**Erreur ! Source du renvoi introuvable.**) est c ontextifiable, et sa contrepartie exprimée sous forme de Contexte est contextifiée.

À l'inverse, nous utilisons le verbe « réifier » pour décrire la reconstitution d'un objet à partir de sa forme contextifiée. Reconstruire un Agent à partir de sa forme contextifiée est une réification de ces Agents.

Le passage du temps se représente par étapes discrètes dans ContextAA. Ainsi, lorsque nous exprimons une valeur de temps t telle que l'indice d'une étape dans l'exécution d'un Hôte, nous considérons $t \in \mathbb{Z}$.

4.2.2 Appartenance à un type

Nous écrivons $x{:}\,T$ pour exprimer que x est une instance du type T.

Nous considérons certains types comme primitifs, par exemple $string$ pour des chaînes de caractères, int pour des entiers ou $bool$ pour des booléens, sans définir systématiquement les bornes minimale et maximale qui leur sont associées.

Nous acceptons à titre de type un intervalle tel que $[0..1]$, signifiant « toutes les valeurs entre 0 et 1 inclusivement » ou $(0{,}1)$, signifiant les seules valeurs 0 et 1. Nous acceptons aussi les notations mathématiques usitées telles que $\mathbb{N}$ pour les entiers naturels ou $\mathbb{Q}$ pour les rationnels, sans bien sûr négliger la différence entre l'idéal mathématique d'un ensemble de cardinalité infinie et notre implémentation, plus concrète. Une écriture comme $[0..1] \in \mathbb{Q}$ signifiera un rationnel situé inclusivement entre 0 et 1, et il en sera de même pour l'écriture $r \in \mathbb{Q}, 0 \leq r \leq 1$.

Certains types sont particulièrement importants pour ContextAA. Entre autres, parmi ceux-ci, on trouve les noms (type $Name$), les Contextes (type C), les Agents (type A), les Hôtes (type H) et les critères (type $Crit$). D'autres types clés de notre modèle sont introduits ultérieurement.

Lorsqu'un type est générique, nous utilisons une notation entre chevrons. Nous écrivons par exemple $Set\langle T\rangle$ pour un ensemble générique d'éléments de type T ou $List\langle bool\rangle$ pour une liste de booléens.

4.2.3 Types de fonctions

Le type $Func$ est utilisé pour identifier une fonction, sans égard à son arité.

Nous exprimons $decltype\big(F(T)\big)$ le type du résultat de l'application d'une fonction de type F sur une donnée de type T, tout comme nous exprimons $decltype\big(F(T_0, T_1, \dots, T_{n-1})\big)$ le type de l'application d'une fonction de type F sur des paramètres de type $T_0, T_1, \dots, T_{n-1}$.

4.2.4 Foncteurs et expressions λ

Nous utilisons le vocable foncteur au sens d'un objet qui se comporte comme une fonction. En ceci, nous faisons un choix terminologique semblable à celui de C++, avec les *Function Objects*, et cousin de celui de certains langages fonctionnels où un foncteur est une fonction qui se comporte comme un objet. Les expressions λ sont pour nous une forme abrégée de foncteur anonyme.

Nous avons recours à un foncteur lorsqu'une opération bénéficie d'être porteuse d'états persistant d'un appel à l'autre, ce qui permet par exemple de construire une fermeture sur une valeur.

4.2.5 Chaînes de caractères

Nous considérons primitives les opérations telles que la concaténation ou la comparaison pour fins d'égalité de contenu ou d'ordonnancement lexicographique sur des chaînes de caractères. Nous considérons aussi, pour alléger le discours, que la conversion d'un nombre en chaîne de caractères est implicite lorsque le contexte le demande.

Par souci de simplicité, nous escamotons en grande partie la question des encodages de caractères pour fins d'internationalisation, et nous ne nous préoccuperons pas de majuscules et de minuscules. Évidemment, sur le plan technique, ContextAA traite ces considérations.

Nous écrivons une chaîne de caractères vide sous la forme "".

La fonction $length$ de signature $length :: string \to \mathbb{N}$ exprime le nombre de symboles d'une chaîne de caractères; ainsi, $length("abc") = 3$ et $length("") = 0$. Nous considérons cette fonction primitive.

La concaténation de chaînes de caractères s'exprime comme suit :

$$concat :: string \to string \to string$$

$$concat("", "") \triangleq ""$$

$$concat("", s) \triangleq s$$

$$concat(s, "") \triangleq s$$

$$concat("a \dots b", "c \dots d") \triangleq "a \dots d"$$

Nous présumons que la gestion de caractères spéciaux comme le caractère "\"" respecte les usages et les attentes de plusieurs langages de programmation, par exemple C, C++, Java et C#. En pratique, concaténer "J'aime mon \"" et "prof\"!" résultera en "J'aime mon \"prof\"!".

4.2.6 Ensembles

Nous écrivons $\{a, b, \dots, i, \dots\}$ l'ensemble contenant les éléments $a, b, \dots, i, \dots$

Nous exprimons l'ensemble vide par $\emptyset$ ou { } selon les circonstances.

Dans ContextAA, un ensemble est typé : on parlera généralement de $Set\langle T\rangle$ plutôt que de Set pour un ensemble dont les éléments sont de type T.

Étant donné un ensemble s dans ContextAA, chaque élément de s est unique dans s :

$$s: Set\langle T\rangle, e: T \left(e \in s \Leftrightarrow (s \cup \{e\} = s) \right)$$

Soit $s: Set\langle T\rangle$, nous exprimons la cardinalité de s par $|s|$.

L'appartenance d'un élément e à un ensemble S s'exprime $e \in S$ mais nous utilisons parfois $e: S$, à la manière de l'appartenance à un type, lorsque cela s'y prête et n'entraîne pas de confusion.

4.2.7 Listes

Nous exprimons une liste de n éléments de même type sous la forme $[e_0, e_1, \dots, e_{n-1}]$ lorsque nous identifions les éléments un à un.

Lorsque nous souhaitons exprimer une liste par construction d'une tête t et d'une queue q, nous exprimons par $[t|q]$ la liste concaténée dont t est la tête et q est la queue; dans ce cas, q est la liste des éléments à la suite de t. Pour exprimer une liste d'au moins un élément, nous écrivons $[t|\ \dots]$.

La liste vide est représentée par [ ]. Il arrive que nous considérions comme interchangeables liste vide [ ] et ensemble vide $\emptyset$ lorsque le contexte s'y prête et lorsque cela permet d'alléger l'écriture.

4.2.8 Fonctions

Nous exprimons une fonction selon deux notations distinctes, soit celle de son type, qui exprime sa signature et explique comment l'appeler, et celle exprimant sa définition, son effet. À titre d'exemple, une fonction $carré$ acceptant un entier en paramètre et ayant pour résultat un entier qui soit le carré dudit paramètre se présenterait comme suit :

$$carré :: int \to int$$

$$carré(n: int) \triangleq n \times n$$

ou, alternativement :

$$carré :: int \to int$$

$$carré(n: int) \triangleq n^2$$

La première expression est la signature de la fonction, et indique que $carré$ consomme un int et a pour résultat un int. Nous admettons les fonctions génériques sur la base de types $T, U, \dots$ avec l'écriture $f\langle T, U\rangle :: T \to U$ ou simplement $f :: T \to U$ si le contexte s'y prête. À titre d'exemple :

$$carré\langle T\rangle :: T \to T$$

$$carré(n: T) \triangleq n \times n$$

De manière alternative, si cela n'obscurcit pas le propos, nous irons même jusqu'à simplifier l'écriture de la description de l'effet d'une fonction en omettant les types de ses paramètres :

$$carré(n) \triangleq n \times n$$

Dans le cas où une fonction a des implémentations variant selon la forme de ses paramètres, nous offrons plus d'une définition pour son effet. Par exemple, pour exprimer un prédicat $empty$ applicable à une liste, nous pourrions avoir :

$$empty :: List\langle T\rangle \to bool$$

$$empty([h|\ \dots]) \triangleq false$$

$$empty([\ ]) \triangleq true$$

Nous acceptons un même nom pour plusieurs signatures de types distinctes. Par exemple, $empty$ applicable à une $List\langle T\rangle$ peut coexister avec $empty$ applicable à un $Set\langle T\rangle$ sans que cela ne cause d'ambiguïté en pratique dans ContextAA.

Nous supposons, dans notre notation, le *Currying*, qui permet de traiter une fonction d'arité $n: \mathbb{N}, n > 0$ comme une fonction acceptant un paramètre et retournant une fonction d'arité $n - 1$. Pour cette raison, une fonction comme (notation C) :

```
int somme(int x, int y) { return x + y }
```

s'exprimera, dans notre formalisme, sous la forme :

$$somme :: int \to int \to int$$

$$somme(x, y) \triangleq x + y$$

où la signature de types suppose visiblement le *Currying*.

Dans le cas de foncteurs ou d'expressions λ, qui permettent la fermeture (§4.2.9) sur des états capturés du contexte lors de l'appel, nous utilisons la notation suivante :

$$memeValeurQue :: int \to \Big(int \to bool\Big) \to bool$$

$$memeValeurQue(val: int)(n: int) \triangleq val = n$$

Dans cette notation, val est capturé à la construction de $memeValeurQue$, alors que n est utilisé par la suite chaque fois que le $memeValeurQue$ est utilisé comme une fonction. Cela est similaire à ce qui est fait dans le langage C++, où $memeValeurQue$ serait un foncteur ou une λ (val serait capturé à la construction), alors que dans un langage fonctionnel, ce serait une étape intermédiaire d'une séquence de *Currying*.

Pour simplifier l'expression de certains algorithmes, nous utilisons :

$$boolToInt :: bool \to (0,1)$$

$$boolToInt(true) \triangleq 1$$

$$boolToInt(false) \triangleq 0$$

Dans le cas d'une surcharge d'un opérateur op, nous exprimons sous la forme (op) la signature de la fonction résultante :

$$(<) :: Set \to Set \to bool$$

$$(<)(s, s') \triangleq |s| < |s'|$$

4.2.9 Fermetures

À titre utilitaire, nous utilisons parfois la capacité de construire une fermeture sur un paramètre, exprimée de manière générique ci-dessous :

$$bind :: Func \to Arg \to \Big(Arg' \to Result\Big) \to Result$$

$$bind(f, a) \triangleq \big(Func(x: Arg') \triangleq f(a, x)\big)$$

Dans cette écriture, $Func$ est une fonction au sens générique du terme, et $Result$ est le type du résultat de son application sur un paramètre donné. Exprimé en mots, $bind(f, x)$ retourne une opération g telle que $g(y) \equiv f(x, y)$. Sur le plan technique, la résolution du type de $g(y)$, donc de $f(x, y)$, est faite de manière statique à partir de la signature des types impliqués.

De même, pour alléger certaines écritures, nous utilisons aussi des outils typiques de la programmation fonctionnelle tels que :

$$map \rightarrow Func \rightarrow List\langle T\rangle \rightarrow List\langle decltype\big(Func(T)\big)\rangle$$

$$map(f, [\quad]) \triangleq f([\quad])$$

$$map(f, [h|t]) \triangleq [f(h)|map(f, t)]$$

La notation $decltype(expr)$ prend le sens de type de l'expression $expr$. Opérant sur la base des types seuls, elle n'évalue pas la valeur de l'expression $expr$.

4.2.10 Transformer un ensemble en liste

Construire $lst: List\langle T\rangle$ à partir de $s: Set\langle T\rangle$ pour un type T donné peut se faire par construction, c'est-à-dire en prenant un élément de s, en considérant cet élément comme la tête de lst, et en construisant la queue de lst à partir de ce qu'il reste dans s. La liste vide [ ] est construite à partir de l'ensemble vide $\emptyset$.

$$makeList :: Set\langle T\rangle \rightarrow List\langle T\rangle$$

$$makeList(\emptyset) \triangleq [\quad]$$

$$makeList(\{e_0, e_1, \dots, e_{n-1}\}) \triangleq [e_0|makeList(\{e_1, \dots, e_{n-1}\})]$$

Dans la liste résultant immédiatement d'une telle construction, l'ordre des éléments ne doit pas être considéré comme important, la source étant un ensemble, donc n'étant pas ordonnancée de prime abord.

4.2.11 Gestion des erreurs

En pratique, certaines expressions sont des erreurs, et peuvent mener, selon les langages, à des bris, des erreurs, des levées d'exceptions ou des comportements indéfinis.

Bien que l'implémentation de ContextAA prenne soin de réaliser un traitement adéquat des erreurs et des situations exceptionnelles, nous avons escamoté cette question pour l'exposition de notre formalisme. La raison pour ce choix est que le traitement des erreurs est techniquement complexe, dépend parfois du contexte, et pourrait obscurcir le cœur du propos que notre formalisme vise à mettre en lumière. Pour cette raison, la description du formalisme réduit le signalement de cas problématiques au seul mot $error$.

4.2.12 Fonctions sur des listes

Pour exprimer une signature de type, nous utilisons $List\langle T\rangle$ pour décrire une séquence d'éléments de type T. Sauf indication contraire, dans ce document, cette notation décrit non pas une structure de données spécifique, par exemple une liste simplement ou doublement chaînée, mais bien une séquence d'éléments du même type. Lorsque les éléments sont de types distincts, nous utiliserons plutôt le terme n-uplet, ou simplement uplet.

Les fonctions suivantes sont utilisées. Pour établir qu'une liste l soit vide ou non, un prédicat $empty(l)$ est décrit à §4.2.8. Pour obtenir l'élément en tête d'une liste (précondition : la liste n'est pas vide) :

$$head :: List\langle T\rangle \to T$$

$$head([h|\ldots]) \triangleq h$$

$$head([\quad]) \triangleq error$$

De même, pour obtenir la queue d'une liste :

$$tail :: List\langle T\rangle \to List\langle T\rangle$$

$$tail([h|t]) \triangleq t$$

$$tail([\quad]) \triangleq [\quad]$$

Contrairement à $head$, qui opère sur un élément de la liste, $tail$ a un comportement clairement défini en présence d'une liste vide.

La longueur d'une liste peut s'exprimer comme :

$$length :: List\langle T\rangle \to int$$

$$length([t|q]) \triangleq 1 + length(q)$$

$$length([\quad]) \triangleq 0$$

La concaténation de deux listes s'exprime comme suit :

$$concat :: List\langle T\rangle \to List\langle T\rangle \to List\langle T\rangle$$

$$concat([\quad],[\quad]) \triangleq [\quad]$$

$$concat([\quad], lst) \triangleq lst$$

$$concat(lst, [\quad]) \triangleq lst$$

$$concat([h|t], [h'|t']) \triangleq [h|concat(t, [h'|t'])]$$

4.2.13 Fonctions et notations étendues

Dans certaines circonstances, nous introduisons dans une fonction des alias à l'aide de let pour en arriver à une écriture plus concise. Par exemple :

$$sommeTailles :: Set\langle T\rangle \to Set\langle T\rangle \to \mathbb{N}$$

$$sommeTailles(s_0, s_1) \triangleq \begin{cases} let\ x = |s_0|, y = |s_1| \\ x + y \end{cases}$$

Dans certaines circonstances aussi, nous utilisons une structure alternative pour choisir entre deux options de calcul sur la base de critères qui, autrement, seraient pénibles à exprimer. Par exemple, la fonction $sommeVals$ ci-dessous qui calcule la comme des valeurs entières entre deux entiers $bMin$ et $bMax$ tels que $bMin \leq bMax$:

$$sommeVals :: \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$$

$$sommeVals(n, n) \triangleq n$$

$$sommeVals(bMin, bMax) \triangleq \begin{cases} error \text{ if } bMax < bMin \\ bMin + sommeVals(bMin + 1, bMax) \text{ otherwise} \end{cases}$$

Les alternatives sont des allègements syntaxiques. Il est possible de les remplacer par des structures fonctionnelles, par exemple :

$$select :: bool \to Func \to Func \to Args \ldots \to Result$$

$$select(true, f, f', x \ldots) \triangleq f(x \ldots)$$

$$select(false, f, f', x \ldots) \triangleq f'(x \ldots)$$

Dans cette notation, $Args$ ... est une séquence de types et x ... est une séquence de paramètres dont les types correspondent, dans l'ordre, à ceux décrits par $Args$ ... ce qui permet d'écrire :

$$sommeVals(n, n) \triangleq n$$

$$sommeVals(a, b) \triangleq select\big(b < a, \big((a, b) \triangleq error\big), \big((a, b) \triangleq a + sommeVals(a + 1, b)\big), a, b\big)$$

Dans cette dernière expression, des fonctions anonymes (des expressions λ) sont construites dans la définition de $sommeVals(a, b)$ et sont passées à select pour que celle-ci évalue l'une ou l'autre en fonction du fruit de l'évaluation de la condition $b < a$.

4.2.14 Opérateurs

Dans certains cas, les fonctions de ContextAA sont en fait des opérateurs. Puisque les opérateurs relationnels dans notre formalisme utilisent des notations se rapprochant de celles connues en mathématiques plutôt que de celles rencontrées dans certains langages de programmation (p. ex. : nous exprimons que a et b sont égaux par $a = b$ plutôt que par `a==b`), l'expression sous forme de fonction de ces opérateurs dans notre formalisme pose un problème de notation.

Devant choisir une notation, nous avons utilisé des noms tels que ceux rencontrés dans certains langages de programmation contemporains, ces noms ayant le mérite d'être compacts et usités. Ainsi, à titre d'exemple, la fonction $(==)$ définit l'opérateur relationnel « est égal à ».

4.2.15 Notation en extension

Nous écrivons $[x \in S \parallel pred(x)]$ pour exprimer la liste des éléments x de l'ensemble S tels que $pred(x)$. Nous utilisons aussi parfois cette écriture pour les chaînes de caractères.

4.2.16 Paires et uplets

Lorsque cela s'avère pertinent, nous aurons recours à la notation $Pair\langle T, U\rangle$ pour décrire un type représentant une paire de deux éléments, de type T et U respectivement. Les fonctions suivantes seront utilisées pour opérer sur ce type :

$$makePair :: T \to U \to Pair\langle T, U\rangle$$

$$makePair(k{:}\,T, v{:}\,U) \triangleq [k, v]$$

Nous utilisons ici une notation de liste pour simplifier l'écriture, ce qui ne présume pas d'une implémentation sur la base du même substrat en pratique. De même, dans les fonctions qui suivent, nous considérons une instance de $Pair\langle T, U\rangle$ où la partie de type T se nomme k et la partie de type U se nomme v comme équivalente à la liste $[k, v]$ outre le fait que nous levons alors la restriction selon laquelle les éléments d'une liste sont tous du même type.

$$first :: Pair\langle T, U\rangle \to T$$

$$first(p: Pair\langle T, U\rangle) \triangleq head(p)$$

$$second :: Pair\langle T, U\rangle \to U$$

$$second(p: Pair\langle T, U\rangle) \triangleq head\big(tail(p)\big)$$

Dans le cas où nous représentons un n-uplet pour $n: \mathbb{N}, n > 2$, nous généralisons le cas particulier $Pair\langle T, U\rangle$ au cas général $Tuple\langle T_0, T_1, \dots, T_{n-1}\rangle$. Ainsi, nous avons :

$$makeTuple :: T_0 \to T_1 \to \dots \to T_{n-1} \to Tuple\langle T_0, T_1, \dots, T_{n-1}\rangle$$

$$makeTuple(e_0: T_0, e_1: T_1, e_2: T_2, \dots, e_{n-1}: T_{n-1}) \triangleq [e_0, makeTuple(e_1, e_2, \dots, e_{n-1})]$$

$$makeTuple(e: T) \triangleq [e]$$

$$tupleElement :: \mathbb{N} \to Tuple\langle T_0, T_1, \dots, T_{n-1}\rangle \to T$$

$$tupleElement(0, t) \triangleq head(t)$$

$$tupleElement(m, t: Tuple\langle T_0, T_1, \dots, T_{n-1}\rangle) \triangleq \begin{cases} error \text{ if } m \geq n \\ tupleElement\big(m - 1, tail(t)\big) \text{ otherwise} \end{cases}$$

4.2.17 Contexte

Dans notre notation, un Contexte se nomme habituellement c, c', c'', ... et l'ensemble des Contextes est identifié par C. Ainsi, l'écriture $\forall c: C$ signifie « pour tout Contexte c ».

4.2.18 Agent

Dans notre notation, les Agents se nomment habituellement a, a', a'', ... et l'ensemble des Agents est noté A. Nous écrivons $\forall a: A; a \in h$ pour signifier « tout Agent a sur l'Hôte h »; nous abusons ainsi quelque peu de la notation ensembliste mais supposons que cela ne nuira pas à la compréhension du propos. Nous abrégeons cette écriture à $\forall a \in h$ quand le contexte est tel que $a: A$ nous semble évident.

4.2.19 Hôte

Dans notre notation, les Hôtes se nomment habituellement h, h', h'', ... et l'ensemble des Hôtes est noté H.

Le voisinage d'un Hôte h est constitué des Hôtes avec lesquels h peut communiquer sans passer par un Hôte intermédiaire, et est noté N_h.

4.2.20 Espace contextuel

L'espace contextuel de $a: A$ s'exprime S_a. L'espace contextuel de $h: H$ est noté S_h.

Il va de soi que $\forall a \in h: S_a \subset S_h$. Nous n'utilisons pas $\forall a \in h: S_a \subseteq S_h$ du fait qu'il est entendu que h aura toujours des Contextes propres à son fonctionnement, et disjoints de ceux des Agents qui y logent.

4.2.21 Nom de Contexte

Nous notons $Name$ l'ensemble des noms de Contextes. Les règles de validité d'un nom $n: Name$ sont données à l'annexe A.

4.3 Contexte

Nous définissons le Contexte de manière simple et récursive sous la forme d'une paire nom, valeur :

$$c = n\{v\}$$

Le nom n respecte certaines règles syntaxiques (§4.4.1), alors que la valeur v est un ensemble de Contextes représentant ses valeurs. Notez que c lui-même n'est pas un ensemble mais bien une paire; l'écriture $c = n\{v\}$ exprime les éléments constitutifs de c mais pas sa structure.

Étant donné $c = n\{v\}$, nous avons $v = \{v_0\ v_1 \ldots v_{m-1}\}$ avec $v_i: C; 0 \leq i < m$.

Dans un souci de concision, nous écrivons parfois $name(c)$ pour le nom de c, et $value(c)$ pour la valeur de c.

Le nom d'un Contexte est un identifiant unique dans son contexte : pour la plupart des opérations, deux Contextes sont considérés un seul et même Contexte s'ils ont le même nom (§4.3.4).

Nous considérons équivalentes les écritures $v = \{\ \ \}$ et $v = \emptyset$.

Un exemple simple de Contexte c serait :

$$c = 34\left\{public\left\{temperature\{unite\{Fahrenheit\}valeur\{32.5\}\}\right\}\right\}$$

…où $\boldsymbol{name(c) = 34}$ et où $\boldsymbol{|value(c)| = 1}$. Une exploration des sous-Contextes dans cet exemple mène éventuellement au Contexte de nom $\boldsymbol{temperature}$ dont la valeur est un ensemble de Contextes de cardinalité $\mathbf{2}$. Notez que la forme utilisée dans cet exemple est la forme canonique du Contexte, ce dernier pouvant être représenté aussi sous une forme compacte (nous y reviendrons à la section §4.3.5).

4.3.1 Contextes bruts et synthétiques

Nous nommons Contexte brut le fruit de la transformation initiale d'une donnée en réel enrichi. Cette transformation T est de la forme :

$$\mathrm{T} :: T \rightarrow C$$

et transforme donc un T, type de la donnée saisie, en C (donc en Contexte).

Nous nommons Contexte synthétique le fruit d'une transformation T de la forme :

$$\mathrm{T} :: C \to C$$

qui transforme un Contexte en un Contexte. Le Contexte synthétique combine fréquemment plusieurs Contextes sources pour générer un Contexte destination, ce qui peut être représenté par applications successives d'une transformation générant un Contexte synthétique.

$$\mathrm{T} :: C \to C \to \ldots \to C$$

En pratique, nous utilisons le terme Contexte sans le qualifier de brut ou de synthétique, sauf si la distinction peut faciliter la compréhension.

4.3.2 Construction d'un Contexte

Au besoin, nous exprimons la construction d'un Contexte à partir d'un nom n et d'une valeur v comme :

$$Contexte :: Name \to Set\langle C\rangle \to C$$

$$Contexte(n{:}\, Name, v{:}\, Set\langle C\rangle) \triangleq n\{v\}$$

4.3.3 Nom et valeur d'un Contexte

Pour alléger l'écriture, nous définissons les fonctions $name$ et $value$ applicables à un Contexte :

$$name :: C \to Name$$

$$name(n\{v\}) \triangleq n$$

$$value :: C \to Set\langle C\rangle$$

$$value(n\{v\}) \triangleq v$$

Étant donné $c{:}\, C$, nous écrivons $x \in c$ pour exprimer de manière concise $x \in value(c)$. S'il importe d'inclure c lui-même dans l'expression, alors nous écrivons $x \in \{c \cup value(c)\}$.

4.3.4 Équivalence entre Contextes

L'équivalence entre deux Contextes dans ContextAA est un comparatif de surface, pas un comparatif en profondeur. Cette opération repose sur l'égalité de leurs noms, et présume donc de l'unicité du nom dans son contexte :

$$\forall \mathrm{c}, \mathrm{c}' {:}\, C \left(\mathrm{c} = \mathrm{c}' \Leftrightarrow name(\mathrm{c}) = name(\mathrm{c}')\right)$$

Ainsi, nous avons :

$$(==) :: C \to C \to bool$$

$$(==)(n\{v\}, n'\{v'\}) \triangleq n = n'$$

Exprimé autrement, dans un contexte donné, $name(c)$ identifie c sans équivoque. Cette définition de l'équivalence en termes de comparaison en surface est complétée par notre approche au nommage du Contexte, et détermine une approche pragmatique plutôt que puriste à la comparaison de Contextes. Cette approche n'est pas universelle, mais constitue un choix facilitant l'usage du Contexte dans la plupart des situations.

4.3.5 Forme canonique et forme compacte

Il existe plus d'une écriture équivalente pour un même Contexte. L'une d'elles est la forme canonique, où chaque nom composite est remplacé par un équivalent hiérarchique.

En plus de la forme canonique, il est aussi possible d'exprimer un Contexte sous forme compacte, où chaque Contexte hiérarchique $c: C; size(c) = 1$ est remplacé par le nom composite (§4.4.2) équivalent.

Par exemple :

- la forme canonique du Contexte $h: a: c\{x\ y\}$ est $h\left\{a\left\{c\{x\ y\}\right\}\right\}$;
- la forme canonique du Contexte $h: a\{x\ y: z\ w\}$ est $h\{a\{x\ y\{z\}\ w\}\}$;
- la forme compacte du Contexte $h\left\{a\left\{c\{x\ y\}\right\}\right\}$ est $h: a: c\{x\ y\}$.
- la forme compacte du Contexte $h\{a\{x\ y\{z\}\ w\}\}$ est $h: a\{x\ y: z\ w\}$.

4.3.6 Sous-Contextes

Si $c, c': C$, alors c' est un sous-Contexte de c si c' est l'une des valeurs de c ou si c' est un sous-Contexte d'un de ces valeurs. Exprimé autrement :

$$isSubCtx :: C \to C \to bool$$

$$isSubCtx(c', c) \triangleq \left(c' \in value(c)\right) \vee \left(\exists c'' \in value(c): isSubCtx(c', c'')\right)$$

4.3.7 Taille d'un Contexte

La fonction $size$ exprime la cardinalité de la valeur d'un Contexte :

$$size :: C \to \mathbb{N}$$

$$size(c: C) \triangleq |value(c)|$$

Quelques exemples suivent :

Table 1 - Exemples de tailles de Contextes

$c : C$	$size(c)$
$c = n\{\ \ \}$	$size(c) = 0$
$c = n\{a\}$	$size(c) = 1$
$c = n\{a\ b\}$	$size(c) = 2$
$c = n\{a\{b\}\ c\}$	$size(c) = 2$
$c = n\{a\{b\}\ c\ d\{e\ f\ g\ h\}\}$	$size(c) = 3$

4.3.8 Contexte terminal

Soit $c: C$, alors c est terminal s'il a un nom mais pas de valeur :

$$\forall c \in C \left(terminal(c) \Leftrightarrow value(c) = \emptyset\right)$$

Exprimé sous forme de fonction :

$$terminal :: C \rightarrow bool$$

$$terminal(c) \triangleq size(c) = 0$$

Exprimé autrement, un Contexte terminal n'a pas de sous-Contextes.

4.3.9 Autres identités

Pour $c: C$, les identités suivantes s'avèrent :

$$terminal(c) \wedge empty(n) \Rightarrow c \equiv \emptyset$$

$$terminal(c) \wedge \neg empty(n) \Rightarrow c \equiv n$$

Nous considérons donc équivalents l'un à l'autre un Contexte terminal dont le nom est vide et $\emptyset$, tout comme nous considérons équivalents l'un à l'autre $c = n\{\emptyset\}$ et n. En pratique, ceci permet d'alléger certaines écritures.

4.3.10 Contexte bien formé et malformé

Qu'un Contexte soit bien formé est un invariant. Un « Contexte » qui serait non-terminal dont le nom est vide serait malformé :

$$\nexists c: C(\neg terminal(c) \wedge empty(name(c)))$$

Puisqu'un « Contexte malformé » est un bris d'invariant pour le Contexte, un tel Contexte ne peut être rencontré en pratique que lorsque le Contexte est en cours de construction, de finalisation ou de mutation, ou encore lorsque le Contexte doit être construit à partir d'un autre format. Techniquement, un « Contexte malformé » ne peut être intégré à ContextAA, et n'est donc pas vraiment un Contexte.

À titre d'exemple :

- le Contexte { } est bien formé et équivaut à ""{ } ou à ∅;
- le Contexte 3{ } est bien formé et peut aussi s'écrire 3{ } ou 3;
- le Contexte $unit\{Celsius\}$ est bien formé;
- le Contexte $température\{unité\{Celsius\}value\{10\}\}$ est bien formé;
- le Contexte $temperature\{unit\{Fahrenheit\}value\{50\}\}$ est bien formé;
- $\{bad\ \{\}$ est malformé et ne peut être utilisé pour construire un Contexte.

4.3.11 Représentations alternatives

Le Contexte tel que nous le définissons représente de l'information pouvant être structurée sous forme arborescente, avec la restriction qu'un nœud c ne peut pas avoir deux enfants c', c'' tels que $equivalence\big(name(c'), name(c'')\big) = 1$ (§4.4.6.4).

Bien que nous utilisions une représentation reposant sur des accolades équilibrées dans notre notation, il est possible d'exprimer le Contexte autrement mais de manière équivalente, comme l'illustre la figure 7 :

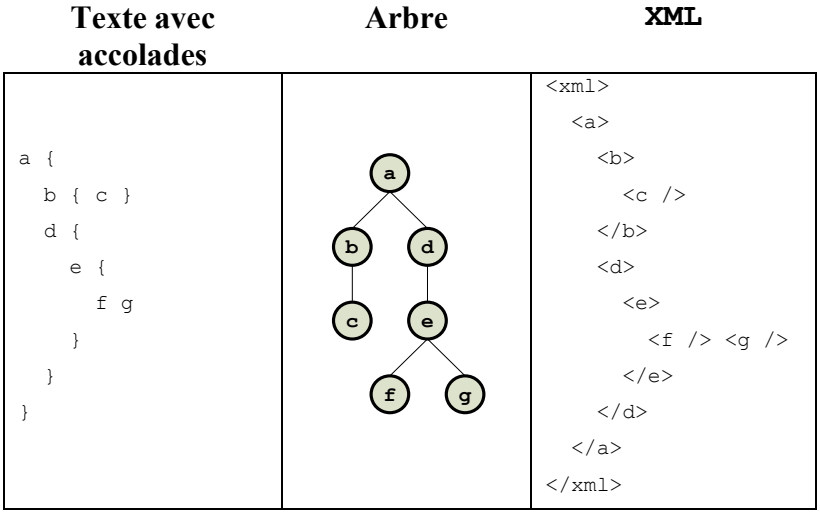


Figure 7 - Contexte et forme arborescente

Pour cette raison, nous décrivons parfois le Contexte avec un vocabulaire utilisé pour exprimer des concepts propres aux arbres, parlant par exemple de nœud racine ou de profondeur.

4.3.12 Profondeur d'un Contexte

Dans l'exemple de la figure 7 ci-dessus, le Contexte présenté a pour racine le nœud a et est de profondeur 4. En pratique, nous définissons la profondeur comme suit :

$$depth :: C \to \mathbb{N}$$

$$depth(\emptyset) \triangleq 0$$

$$depth(n\{v\}) \triangleq 1 + max_{e \in v}\big(depth(e)\big)$$

4.3.13 Fusion de Contextes

La valeur d'un Contexte c est de type $Set\langle C\rangle$, et l'équivalence de deux Contextes dépend de leurs noms; c'est d'ailleurs la raison pour laquelle un nom de Contexte est unique dans son contexte.

Une conséquence de ces choix est qu'ajouter $c' = a\{b\}$ à $value(c)$ si $c = x\{a\ b\ c\}$ résulte en $c'' = x\{a\ b\ c\}$ plutôt qu'en $c'' = x\{a\{b\}\ b\ c\}$, car a est considéré égal à $a\{b\}$. De façon générale, cette propriété du Contexte est avantageuse et permet de définir des opérations relativement peu coûteuses sur ce type, en particulier à la racine des Contextes : les comparaisons de Contextes peuvent dans bien des cas être superficielles, ce qui est une économie appréciable sur les systèmes à faibles ressources que ContextAA vise à occuper. Cependant, cette propriété peut surprendre celles et ceux qui se seraient attendu à une comparaison en profondeur dans de telles circonstances.

Lorsque ce comportement n'est pas celui désiré, il est possible de réaliser une fusion de deux Contextes :

$$merge :: C \to C \to C$$

$$merge(n\{v\}, n'\{v\}) \triangleq \emptyset$$

$$merge(n\{v\}, n'\{v'\}) \triangleq \emptyset$$

$$merge(n\{v\}, n\{v'\}) \triangleq n\left\{\bigcup_{\substack{e\in v,\\ e'\in v'}} merge(e, e')\right\}$$

Une fusion de Contextes est beaucoup plus coûteuse que ne l'est l'insertion sur la base du nom seul des Contextes, ce qui explique que ce ne soit pas le mode d'opération par défaut dans ContextAA.

4.4 Noms de Contextes

Un nom de Contexte, ou $Name$, est assujetti à un certain nombre de règles lexicales et syntaxiques, listées à l'annexe A. Nous utilisons $Name$ ('N' majuscule) pour indiquer le type, et $name$ ('n' minuscule) pour la fonction décrite à §4.3.3.

4.4.1 Vocabulaire

En résumé, un nom de Contexte peut être :

- vide;
- fait d'un seul élément; ou
- fait d'une séquence d'éléments séparés l'un de l'autre par un « : ». On dira alors du nom qu'il est composite (§4.4.2), ou de forme composée. Le recours au symbole « : » pour séparer des éléments d'un même nom est similaire à l'une des syntaxes utilisées pour les URN [111].

Certains éléments ne sont applicables qu'à un nom qui n'est pas composite. À titre d'exemple, le symbole $*$; le symbole $**$; le symbole $?=$ qui peut être suivi par un jeton, de même qu'un numéro de séquence préfixé d'un # ne peuvent être composites.

Un élément d'un nom, qu'il soit composite ou non, peut être une séquence de caractères placée entre guillemets, par exemple "abc" ou "$ab\ c$"; ou encore une séquence d'au moins un symbole acceptable (ni blanc, ni l'un d'un petit groupe de symboles spéciaux). Il peut aussi se limiter à un caractère de soulignement, _, constituant un symbole de non-correspondance pour les fins de certaines opérations de comparaison de noms, ou au symbole ?.

Un support particulier est offert à un élément de la forme $?=s$ où s est un élément d'un nom. Cette forme est utilisée dans certaines expressions plus complexes pour extraire un élément et l'entreposer dans une dictionnaire de modèles permettant de lui référer ultérieurement par le « nom » s dans une expression $eval$, comme dans :

$$a\left\{**\left\{battery\left\{**\left\{remaining\left\{value\left\{?=batt\right\}\right\}\right\}\right\}\right\}\right\}$$

$$a\left\{eval\{"batt"\}\right\}$$

qui évaluerait la valeur résiduelle d'une pile exprimée sous la forme d'un Contexte contenant $battery$, et dont un sous-Contexte $remaining$ contiendrait le sous-Contexte $value$ dans lequel une valeur serait nommée $batt$ pour les besoins du calcul, puis évaluée.

Certains éléments sont en fait des noms réservés pour des opérations prises en charge de manière primitive par ContextAA. Les noms actuellement réservés sont listés à l'annexe A; cette liste est susceptible de croître avec l'évolution de ContextAA.

Quelques exemples de noms correctement formés : `15.2`, `_`, `hello_there`, `"hello there"`, `*`, `"me*too"`, `host:?:ctx:1`, `?:?:3`, `?:?`, `**`, `-3`, `Ohm`, `"int f(int n){return n;}"`, `#5`, `a:"b c":d`, etc.

Quelques exemples de noms malformés : `am bad`, `{f}`, `not*me`, `#hello`, `bye#`, `"hum`, `_oops`.

4.4.2 Noms composites

Un nom de Contexte est composite s'il est de la forme $s[:s]*$ où chaque s est un élément, et constitue alors un n-uplet dont chacun des n éléments est une chaîne de caractères respectant les règles en vigueur pour un nom d'élément. Un nom de Contexte sans séparateur : peut être considéré à la fois comme composite et comme canonique sans que cela ne pose problème.

L'utilisation type de ce format est le triplet $h:a:c$ où $h:H$, $a:A$ et $c:C$. Un tel triplet identifie un Contexte de manière unique. Pour utilisation sur un même Hôte, l'élément h peut être omis; pour utilisation par un seul et même Agent, les éléments h et a peuvent être omis.

La notation composite équivaut à l'écriture compacte d'un Contexte, et peut s'exprimer de manière essentiellement équivalente (en tenant compte des considérations signalées dans §4.3.11) sous forme canonique, dans la mesure où chaque élément respecte les règles de validité pour un nom.

La figure 8 illustre les formes composite et canonique pour des Contextes équivalents :

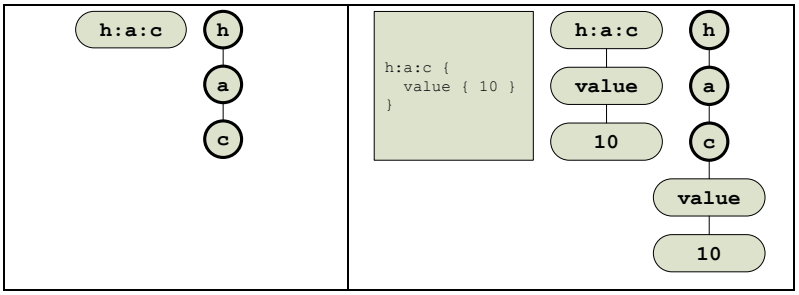


Figure 8 - Forme composite ou canonique

4.4.3 Éléments d'un nom

Parmi les opérations élémentaires définies sur les noms, quelques-unes des plus importantes sont décrites ci-dessous.

4.4.3.1 Nombre d'éléments

Le nombre d'éléments $nelems$ dans un nom peut s'exprimer comme suit :

$$nelems ::= Name \to \mathbb{N}$$

$$nelems("") \triangleq 0$$

$$nelems(e_0 : e_1 : \ldots : e_{m-1}) \triangleq m$$

4.4.3.2 Nom vide

Un nom n est vide si son nombre d'éléments est nul.

$$empty ::= Name \to bool$$

$$empty(n) \triangleq nelems(n) = 0$$

4.4.3.3 Accès à un élément

Un indice entier dans la plage $(0..nelems(n) - 1)$ peut exprimer la position d'un élément dans un nom n. L'accès à un élément peut s'exprimer comme suit :

$$elem :: Name \to int \to string$$

$$elem(e_0 : e_1 : \ldots : e_{m-1}, i) \triangleq \begin{cases} e_i \text{ if } 0 \leq i < m \\ error \text{ otherwise} \end{cases}$$

4.4.3.4 Équivalence entre éléments

Deux éléments sont équivalents l'un à l'autre s'ils sont identiques sur le plan lexicographique, tenant compte de la casse. L'équivalence entre deux éléments est une opération réflexive, symétrique et transitive. Ainsi, à titre d'exemple, soit les éléments suivants :

$$e_0 = abc; e_1 = abc; e_2 = aBc; e_3 = abcd$$

Alors $e_0 = e_1$, $e_1 = e_0$ mais $e_0 \neq e_2$ et $e_0 \neq e_3$.

4.4.3.5 Subsomption

Opérer sur le Contexte demande souvent de réaliser des opérations de mise en correspondance de modèles. Ces opérations sont détaillées à §5.4.2. Lors de telles opérations, la simple équivalence entre éléments est enrichie par des règles de subsomption :

- le symbole ? subsume un élément. Ainsi, si $n_0 = a:?:c, n_1 = a:b:c;\ n_2 =?:d:?$, alors n_0 est équivalent à n_1, n_0 est équivalent à n_2 mais n_1 n'est pas équivalent à n_2;
- le symbole $?= var$ pour un symbole var donné subsume un élément et associe sa valeur à la clé var dans le dictionnaire de modèles associé à l'opération en cours;
- le symbole $*$ subsume un nom entier;
- le symbole $**$ subsume un nombre arbitrairement grand de noms dans une branche d'un Contexte donné. Il est habituellement suivi d'un autre nom, comme par exemple $** \{celsius\}$ qui signifie « tout jusqu'à ce que le nom $celsius$ soit rencontré » et mène à une correspondance avec $celsius$, $a\{celsius\}$, $a\{b\{celsius\}\}$ mais pas $a\{b\}$.

Dans ce document, nous écrirons $subsumes(e, e')$ pour exprimer que e subsume e'.

La subsomption n'est pas symétrique, du fait que $subsumes(e, e')$ ne signifie pas en général que $subsumes(e', e)$.

La subsomption est réflexive : $subsumes(e, e)$.

La subsomption n'est toutefois pas transitive. À titre d'exemple, en effet, $subsumes(a, ?) \wedge subsumes(?, b)$ mais $\neg subsumes(a, b)$.

4.4.4 Noms en tant que texte

Il est possible d'opérer sur un nom n comme si l'on opérait sur du texte :

$$texte :: Name \rightarrow string$$

Ainsi :

$$texte(n) \triangleq "n"$$

$$texte("n") \triangleq "n"$$

$$texte(e_0 : e_1 : \cdots e_n) \triangleq concat\big(texte(e_0), ":", texte(e_1), ":", \cdots, ": ", texte(e_n)\big)$$

4.4.5 Concaténation

La concaténation de deux noms s'exprime comme suit :

$$concat :: Name \rightarrow Name \rightarrow Name$$

$$concat("", n) \triangleq n$$

$$concat(n, "") \triangleq n$$

$$concat(e_0 : e_1 : \ldots : e_{m-1}, e'_0 : e'_1 : \ldots : e'_{n-1}) \triangleq e_0 : e_1 : \ldots : e_{m-1} : e'_0 : e'_1 : \ldots : e'_{n-1}$$

4.4.6 Comparaison entre deux noms

Comparer deux noms peut se faire sur une base d'égalité, de reconnaissance de modèle, d'équivalence, de différence et de distance.

La comparaison de deux noms n_0, n_1 dans certaines fonctions telles que $equivalence(n_0, n_1)$ ou $distance(n_0, n_1)$ requiert que $nelems(n_0) = nelems(n_1)$. Pour ces opérations, des symboles de non-correspondance sont concaténés en nombre suffisant au nom ayant le plus petit nombre d'éléments. Par exemple, lors d'un appel à $equivalence(a\!:b, a\!:b\!:c)$ le nom $a\!:b$ est considéré équivalent à $a\!:b\!:\_$ pour les fins du calcul.

4.4.6.1 Égalité de deux noms

Deux noms n_0, n_1 sont égaux au sens de l'opérateur == si leur éléments sont équivalents lorsque pris un à un. On peut exprimer cette fonction comme suit :

$$(==) :: Name \to Name \to bool$$

$$(==)(n_0, n_1) \triangleq nelems(n_0) = nelems(n_1) \wedge \bigwedge_{i=0}^{nelems(n_0)-1} elem(n_0, i) = elem(n_1, i)$$

4.4.6.2 Différence entre deux noms

La différence entre deux noms n_0, n_1 s'exprime sous la forme suivante, qui rappelle pour l'essentiel la différence ensembliste :

$$difference :: Name \to Name \to Name$$

Le fruit de ce calcul est le nom n' tel que les éléments correspondants à chaque position entre n_0 et n_1 sont conservés à la même position dans n'alors que ceux pour lesquels il n'y a pas de correspondance sont remplacés dans n' par des symboles de non-correspondance. Quand un élément en subsume un autre, l'élément subsumé est celui conservé dans n'. Ainsi :

$$n_0 = a\!:?:b;\ n_1 = d\!:c\!:b \Rightarrow difference(\ n_0, n_1\ ) = \_\!:c\!:b$$

$$n_0 = ?:\mathrm{a};\ n_1 = b\!:?\Rightarrow difference(\ n_0, n_1\ ) = b\!:a$$

$$n_0 = ?:\mathrm{a};\ n_1 = ?:?\Rightarrow difference(\ n_0, n_1\ ) = ?:a$$

4.4.6.3 Équivalence entre éléments de noms

L'équivalence entre deux éléments de noms s'exprime comme suit :

$$equivalence :: string \to string \to (0,1)$$

$$equivalence(s, \_) \triangleq 0$$

$$equivalence(\_, s) \triangleq 0$$

$$equivalence(e, e') \triangleq boolToInt\big(e = e' \vee subsumes(e, e') \vee subsumes(e', e)\big)$$

ou plus simplement :

$$equivalence(e, e'\!: string) \triangleq boolToInt\big(subsumes(e, e')\big)$$

car la subsomption est symétrique et subsume effectivement l'égalité lexicographique.

4.4.6.4 Équivalence entre noms

L'équivalence entre deux noms n_0, n_1 s'exprime sous la forme suivante :

$$equivalence :: Name \to Name \to [0..1] \in \mathbb{Q}$$

$$equivalence(n_0, n_1) \triangleq \begin{cases} let\ N = \max\big(nelems(n_0), nelems(n_1)\big) \\ \dfrac{\sum_{i=1}^{N} equivalence\big(elem(n_0, i), elem(n_1, i)\big)}{N} \end{cases}$$

Le fruit de ce calcul est le rapport entre le nombre d'éléments correspondants entre n_0, n_1 et $\max\big(nelems(n_0), nelems(n_1)\big)$. Ainsi, $equivalence(n_0, n_1) = 1$ seulement si n_0 et n_1 correspondent en tout point l'un à l'autre (§4.4.6.3).

Quelques exemples de la fonction $equivalence(n_0, n_1)$ pour des noms n_0 et n_1 donnés sont proposés à la figure 9 :

$\boldsymbol{n_0}$	$\boldsymbol{n_1}$	$\boldsymbol{equivalence(n_0, n_1)}$
$a:b:c$	$a:b:c$	1
$a:d:c$	$a:b:c$	$\frac{2}{3}$
$a:d:c$	$?:b:c$	$\frac{2}{3}$
$a:d:c$	$?:b:e$	$\frac{1}{3}$
$a:b:c$	$d:e:f$	0

Figure 9 - Exemples d'équivalence entre noms

Selon notre définition, la fonction $equivalence$ admet des équivalences partielles.

4.4.6.5 Disparité opératoire entre formes compacte et canonique

Supposons $n, n' : Name$ tels que :

$$n = e_0 : e_1 : \ldots : e_{m-1} ; n' = e'_0 : e'_1 : \ldots : e'_{m-1}$$

Les règles de transformation de la forme compacte à la forme canonique suggèrent que ces deux noms puissent être considérés équivalents à $c, c' : C$ ci-dessous :

$$c = e_0 \left\{ e_1 \{ \ldots \{ e_{m-1} \} \} \right\} ; c' = e'_0 \left\{ e'_1 \{ \ldots \{ e'_{m-1} \} \} \right\}$$

où $depth(c) = depth(c') = m$.

Cependant, il faut alors remarquer que, étant donné la définition de la fonction $equivalence$, nous avons :

$$c = c' \Leftrightarrow name(c) = name(c') \Leftrightarrow e_0 = e'_0 \Leftrightarrow (equivalence(e_0, e'_0) = 1)$$

Conséquemment, si $equivalence(e_i, e'_i) \neq 1, i > 0$, alors $equivalence(n, n') < 1$, ce qui peut se produire par exemple en comparant $c = a{:}\, b$ avec $c' = a\{b\}$. Ceci implique que les formes compactes et canoniques ne soient pas pleinement équivalentes sur le plan opératoire.

Cependant, rapporter deux Contextes sur le même mode de représentation, qu'il s'agisse du mode compact ou du mode canonique, permet de résoudre cette disparité au besoin. C'est ce que fait ContextAA en pratique, opérant sur des formes canoniques.

4.4.6.6 Distance entre deux noms

La distance entre deux noms s'exprime à partir de l'équivalence entre deux noms (§4.4.6.4) :

$$distance :: Name \rightarrow Name \rightarrow [0..1] \in \mathbb{Q}$$

$$distance(n_0, n_1) \triangleq 1 - equivalence(n_0, n_1)$$

Intuitivement, si $n_0, n_1 : Name$ correspondent en tout point, donc si $equivalence(n_0, n_1) = 1$, alors $distance(n_0, n_1) = 0$.

4.4.7 Reconnaissance de modèles

Le Contexte permet de décrire des modèles pour fins de reconnaissance de Contextes. À cette fin, certains jetons jouent un rôle spécialisé. Pour comprendre ce rôle, il est préférable d'avoir au préalable une familiarité avec les règles d'équivalence de noms et de subsomption.

Le format de Contexte appliqué dans ContextAA permet de représenter des affirmations telles que « le niveau sonore ambiant est de 80 décibels dans la chambre à coucher » ou « la précision du capteur A est meilleure que celle du capteur B », et ce de multiples façons. Le langage interne d'évaluation d'expressions soutient l'expression de certaines de ces relations.

Un modèle de Contexte est une requête exprimée sous forme de Contexte, et dans laquelle certains éléments peuvent être décrits de manière générique. Informellement, dans « le niveau sonore ambiant est de $? = x$ décibels », la valeur mesurée pour x est laissée ouverte. ContextAA offre une syntaxe pour poser des questions à éléments manquants, ce qui permet à $a \in h$ de faire un inventaire des Contexte publiés dans N_h pour déterminer, à partir des valeurs rapportées par ces différents Contextes, quelle est la valeur qui lui semble la plus à propos.

De même, dans « le niveau sonore ambiant est de $? = val$ décibels dans $? = lieu$ », le lieu est associé à un nom ($lieu$) et il en va de même pour la valeur mesurée (val).

ContextAA n'exprime pas le Contexte sous forme de langage naturel mais bien dans un formalisme qui lui est propre, décrit dans §4.3. Ce langage combine des affirmations telles que :

$$a{:}\, 3\left\{"bruit\ ambiant"\{unité\{décibel\}valeur\{80\}lieu\{ici\}\}\right\}$$

qui représente le Contexte 3 produit par l'Agent a, et des requêtes telles que :

$$a'{:}\, 7\left\{"bruit\ ambiant"\{unité\{décibel\}valeur\{*\}\}\right\}$$

ce qui équivaut à :

$$a'{:}\, 7\{"bruit\ ambiant"\{unité\{décibel\}valeur\}\}$$

où $lieu$ n'est pas pris en considération et où la présence du sous-Contexte $valeur$ est importante, sans égard à sa valeur, ou encore :

$$a'' : 35\left\{"bruit\ ambiant"\left\{unité\{décibel\}valeur\{?=val\}lieu\{?=lieu\}\right\}\right\}$$

Représenté sous forme fonctionnelle, la mise en correspondance d'un modèle de Contexte avec une séquence de Contextes a la signature suivante :

$$Dictio :: Pair\langle List\langle C\rangle, List\langle Pair\langle string, List\langle string\rangle\rangle\rangle\rangle$$

$$corresp :: C \to List\langle C\rangle \to Dictio$$

où le type $Dictio$ représente un dictionnaire de modèles (plus de détails plus bas) . Le dictionnaire accepte plusieurs valeurs pour un même nom, ce qui permet à un modèle comme $**\{valeur\{?=val\}\}$ de faire correspondre val à plusieurs noms.

La définition de $corresp(c, lst)$ a été omise par souci de simplicité, celle-ci étant complexe. Soit $corresp(m, lst) = p$, alors $first(p)$ est une liste de Contextes $c \in lst$ pour lesquels la mise en correspondance de m avec c résulte en $c' : C\left(\neg empty(c')\right)$, alors que $second(p)$ est un dictionnaire.

Ces deux exemples de requêtes mettent en relief qu'une requête pour du Contexte dans ContextAA résulte en la production de deux résultats, soit un Contexte, fruit de la correspondance entre le modèle de Contexte décrit par la requête, et un dictionnaire des noms de Contexte correspondant aux noms préfixés par $?=$ dans un modèle lors de la mise en correspondance avec d'autres Contextes.

4.4.7.1 Jeton $*$ (*match any*)

Le jeton $*$ subsume un nom de Contexte. Par exemple :

- l'expression $*$ subsume tout Contexte;
- l'expression $*\{a\}$ correspond à un Contexte, peu importe son nom, dont l'une des valeurs est a, comme par exemple $b\{a\}$ ou $b\{c\ a\{b\}\ d\}$;
- l'expression $a\left\{*\{b\}\right\}$ correspond à un Contexte nommé a dont l'une des valeurs contient dans ses valeurs un Contexte b, comme par exemple $a\left\{c\{b\}\right\}$ ou $a\left\{d\ c\{b\}\right\}$ mais pas $a\{b\}$.

4.4.7.2 Jeton $**$ (*multilevel match any*)

Le jeton $**$ subsume à un nombre arbitrairement grand de Contextes hiérarchisés dans une même branche. Ce jeton apparaît habituellement dans une structure hiérarchique avec un sous-Contexte indiquant où il faut cesser de réaliser une correspondance implicite. Par exemple :

- l'expression $**$ subsume tout Contexte;
- l'expression $**\{val\}$ correspond à toute séquence de Contextes dans une même branche jusqu'à correspondance du nom val, comme par exemple $a\left\{b\{val\}\right\}$, $a\left\{b\left\{c\left\{val\{10\}\right\}\right\}\right\}$, val ou $val\{a\}$, mais pas $a\{b\}$;
- l'expression $a\left\{**\left\{b\left\{**\{c\}\right\}\right\}\right\}$ correspond à un Contexte a ayant un sous-Contexte b, ce dernier ayant un sous-Contexte c;
- évidemment, $**$ subsume $*$.

4.4.7.3 Jeton ? (*match auto*)

Le jeton ? subsume un seul élément de nom. Par exemple :

- l'expression ? correspond à a ou à $b\{c\}$ mais pas à $a{:}\, b\{c\}$;
- l'expression $?:a\{b\}$ correspond à $a{:}\, a\{b\}$ ou $c{:}\, a\{b\}$ mais ne correspond ni à b, ni à $a{:}\, c\{b\}$.

Pour un nom non-composite, ? équivaut à $*$.

4.4.7.4 Jeton ? = (*named element*)

Le jeton ? = a un impact semblable à celui du jeton ? mais associe un nom au modèle correspondant dans un dictionnaire de modèles.

4.4.7.5 Jeton *all*

Le jeton all a pour impact d'obliger que tous les modèles d'un même groupe aient une correspondance, puis de fusionner les résultats. Il sert par exemple à exprimer une requête pour un Contexte contenant à la fois une unité exprimée en degrés Celsius et une valeur numérique.

Pour distinguer les unes des autres les règles décrivant les modèles qui doivent être identifiés pour remplir les exigences d'un all, on aura recours à des clauses d'échappement. Par exemple :

$$all\left\{\#1\{**\{temperature\}\}\#2\{**\{precision\}\}\right\}$$

pour tout Contexte ayant à la fois un sous-Contexte $temperature$ et un sous-Contexte $precision$.

Notez la distinction entre $rq = **\{a\ b\}$ et $rq' = all\left\{\#1\{**\{a\}\}\#2\{**\{b\}\}\right\}$: le modèle de Contexte rq requiert qu'un sous-Contexte ait comme valeurs les Contextes a et b, alors que le modèle de Contexte rq' requiert qu'un Contexte ait un sous-Contexte a et un autre sous-Contexte b, sans que ces deux Contextes fassent partie de la valeur d'un même Contexte. Concrètement, rq accepte $c\{a\ b\}$ mais pas $c\{x\{a\}y\{b\}\}$ ou $c\{a\{b\}\}$ alors que rq' accepte les trois.

4.4.7.6 Jeton # (échappement)

Une clause d'échappement est un élément de nom débutant par le symbole #. Les clauses d'échappement sont pour l'essentiel escamotées lors de la construction d'une opération sur le Contexte. Une clause d'échappement permet d'exprimer un modèle jouant un rôle structurel mais ne devant pas intervenir dans le processus de mise en correspondance.

Un exemple est donné dans §4.4.7.5 avec l'expression $all\left\{\#1\{**\{temperature\}\}\#2\{**\{precision\}\}\right\}$ où les expressions $**\{temperature\}$ et $**\{precision\}$ sont identiques au sens de la comparaison des Contextes (une comparaison de surface). L'insertion de clauses d'échappement comme #1 et #2, constituant des noms distincts, permet d'intégrer ces deux expressions sous un même nom. Dans le cas d'une composition d'opérations sur le Contexte, il est possible de voir les noms précédés par un # comme des étapes du calcul.

4.4.7.7 Échappement et ordonnancement des valeurs

Dans le cas où $c:C$ décrit une composition d'opérations, il se peut que celles-ci doivent être exécutées dans un ordre particulier, or le fait que $value(c)$ soit un ensemble a pour effet que l'ordre dans lequel se trouvent ses éléments soit *a priori* indéfini, et peut par conséquent ne pas être celui souhaité.

Les clauses d'échappement permettent de résoudre ce problème. Par exemple, il est possible d'utiliser l'écriture $\#n$, où $n \in \mathbb{Z}$ et où $\#i < \#j \Leftrightarrow i < j$, pour imposer un ordre de traitement des Contextes c' tels que $c' \in value(c)$. Cette mise en ordre des Contextes est contextuelle, et n'est faite par ContextAA que dans les cas où un tel ordonnancement est pertinent.

4.4.8 Qualifications d'accès au Contexte

Il est d'usage dans ContextAA d'insérer les noms $public$, $local$ ou $private$ dans le Contexte pour indiquer l'accessibilité de la valeur du Contexte ainsi décrit.

$$Q::Set\langle Name\rangle$$

$$Q \triangleq \{private, local, public\}$$

Nous disons que le passage par les noms $public$, $local$ et $private$ sont des usages du fait qu'ils n'ont pas de rôle intrinsèque particulier, hormis leur rôle d'identifiants reconnaissables participant à la mécanique normale de mise en correspondance des Contextes.

Cela dit, les Agents standards, qui participent à la mécanique de prise en charge des requêtes pour du Contexte, respectent cet usage : par exemple, l'Agent standard $a' \in h$ servant d'intermédiaire pour appliquer une requête sur h à la faveur d'un Agent requérant $a \in h' : h \neq h'$ utilisera $local$ à titre de discriminant pour exprimer ladite requête de manière à limiter sa portée au Contexte sur h.

4.4.9 Dictionnaire de modèles

Les opérations de mise en correspondance de Contextes telles que celles décrites à §4.4.7 s'accompagnent d'une possible cueillette de noms de modèles, et ce à travers un dictionnaire de modèles. Cette cueillette constitue une collecte de valeurs pour des variables nommées dans le respect des règles lexicales pour un nom, et est guidée plus spécifiquement par des demandes de nommage.

Par dictionnaire, nous entendons ici une liste dont chaque élément e est un $Pair\langle string, List\langle Name\rangle\rangle$ tel que $first(e)$ est un mot pour lequel un ajout au dictionnaire a été demandé, et $second(e)$ est la liste des noms de Contexte y correspondant.

Le rôle du dictionnaire de modèles est de rapporter l'ensemble des correspondances de noms correspondant à un jeton $?=$ pour une requête de Contexte donnée. Un Agent exprimant une question sous forme de Contexte dans laquelle certains noms de Contextes doivent être répertoriés sous un même nom symbolique, par exemple val dans

$$c\left\{**\left\{battery\left\{**\left\{remaining\left\{**\left\{value\{?=val\}\right\}\right\}\right\}\right\}\right\}\right\}$$

est susceptible de constater plusieurs valeurs (plusieurs noms) correspondant à val en fonction des Agents dans son voisinage. Ces diverses valeurs (ces divers noms) seront logées dans un dictionnaire pour faciliter leur consultation et leur utilisation.

4.4.9.1 Jeton *commit*

Lors d'une requête de Contexte, il est possible de cibler des Contextes d'un modèle pour leur associer des alias transitoires. Ces alias peuvent être réutilisés ailleurs dans la requête si celle-ci est une composition d'opérations, et les Contextes qui leur correspondent peuvent être intégrés sur demande à l'espace Contextuel de l'Agent requérant à l'aide du jeton $commit$.

Par exemple, soit la requête c suivante soumise par l'Agent a (notez que les opérations sur le Contexte telles que $compose$ sont décrites à la section §4.5) :

$$c = compose\left\{\#0\left\{**\left\{memory\left\{**\left\{value\{?=mv\}\right\}\right\}\right\}\right\}\#1\{commit\{mv\}\}\right\}$$

La résolution de c identifie tous les Contextes accessibles au requérant dans la mesure où ceux-ci contiennent un sous-Contexte $memory$ contenant un sous-Contexte $value$ non-terminal, et associe au nom mv dans le dictionnaire de l'étape $\#0$ de cette composition tous les sous-Contextes ainsi identifiés. L'étape #1, quant à elle, mémorise dans S_a le Contexte $a\{dictionary\{mv\}\}$ dont la valeur est la liste des valeurs associées à mv dans le dictionnaire de l'étape $\#0$.

Les Contextes inscrits au dictionnaire de l'Agent a deviennent partie intégrante de S_a et peuvent être utilisés dans la construction de Contextes subséquents.

4.4.9.2 Fonction *tryMatch*

Une fonction de mise en correspondance type est :

$$tryMatch :: C \rightarrow C \rightarrow Dict \rightarrow Pair\langle C, Dict\rangle$$

Cette fonction met en correspondance un Contexte c et un Contexte m, ce dernier décrivant un modèle de Contexte, et peuple un dictionnaire de modèles avec les jetons identifiés par des demandes de nommage. Elle a pour résultat une paire contenant le dictionnaire de modèles peuplé en conséquence, et un Contexte c' correspondant à c mais élagué des branches qui ne sont pas décrites par m.

À titre d'exemple, étant donné le Contexte c et le modèle de Contexte m ci-dessous :

c

```
h:a:c {
  temperature {
    value{10}
    unite{Celsius}
    precision{0.001}
  }
}
```

m

```
h:?=agent:?=ctxid {
  temperature {
    value{ ?=val }
    unite{ ?=unit }
    precision{ ?=prec }
  }
}
```

Le dictionnaire de modèles résultant de la mise en correspondance de m et de c contiendra les associations suivantes :

Clé	Valeur
agent	a
ctxid	c
val	10
unit	Celsius
prec	0.001

4.5 Opérations sur le Contexte

Le Contexte est pour ContextAA à la fois une structure de données et un langage. À ce titre, le Contexte admet un certain nombre d'opérations. Parmi les principales opérations sur le Contexte, on trouve :

- les critères, qui sont des prédicats exprimés sous forme de Contexte;
- les transformations, qui acceptent un Contexte et produisent un Contexte en retour; et
- les algorithmes, qui appliquent des opérations complexes sur un Contexte, souvent à partir d'un critère, et dont le type de résultat peut varier mais est souvent une séquence de Contextes.

4.5.1 Prédicats sous forme de Contexte

Plusieurs opérations, en particulier les critères, produisent un résultat booléen. Lorsque de telles opérations sont faites sur du Contexte et ont pour résultat du Contexte, elles produisent l'un de deux Contextes connus au préalable, soit $true$ (pour « vrai ») et $false$ (pour « faux »). Ces deux Contextes sont des constantes globales de ContextAA, nommées respectivement $true\_context$ et $false\_context$.

4.5.2 Critères standards

Une catégorie importante d'opérations sur le Contexte est le critère, identifié ici par le type $Crit$. Un critère $\varphi: Crit$ est un prédicat unaire contextualisable, applicable sur un Contexte.

$$Crit :: C \rightarrow bool$$

Un critère est typiquement un foncteur ou une fonction. Un critère exprimé sous forme d'un foncteur peut conserver à l'interne des états distincts du Contexte auquel il est appliqué, bien que la plupart soient immuables ou tout simplement sans états pour maximiser leur utilité. Dans ContextAA, les critères jouent le rôle d'opérations primitives pour raisonner sur le Contexte.

Un exemple de critère sans états est $terminal$.

Un exemple de critère immuable avec états est :

$$matchName :: Name \rightarrow C \rightarrow bool$$

$$matchName(patt: Name)(\ c: C) \triangleq equivalence(name(c), patt) = 1$$

Un critère tel que $terminal$ peut s'exprimer au choix sous la forme d'une fonction ou d'un foncteur. Un critère tel que $matchName$ doit s'exprimer sous la forme d'un foncteur ou d'une fonction de plus haut niveau (retournant un critère : $Func \to Name \to \Big(C \to bool\Big)$) pour être en mesure de capturer $patt$ en raison de l'arité (unaire) de l'opération.

Il est possible d'exprimer les tests de visibilité d'un Contexte sous forme de prédicats :

$$public :: C \to bool$$

$$local :: C \to bool$$

$$privé :: C \to bool$$

4.5.3 Transformations standards

Une transformation τ: T est une opération unaire contextualisable, applicable à un Contexte et dont le résultat est une séquence de Contextes :

$$\mathrm{T} :: C \to List\langle C\rangle$$

Dans plusieurs cas, la séquence résultante ne contiendra qu'un seul élément. Règle générale, pour $c: C$, $\tau(c) = \emptyset$ si τ n'est pas applicable à c, alors que dans la majorité des cas $\tau(c) = [c']$ (§4.2.7).

L'une des transformations les plus simples est l'identité, exprimée ici par la fonction $identity$:

$$identity :: C \to List\langle C\rangle$$

$$identity(c) \triangleq [c]$$

Techniquement, cette fonction n'est pas une identité au sens strict, mais retourner une liste de Contextes d'un seul élément permet à cette fonction de s'intégrer de manière plus harmonieuse au reste du système.

Un autre exemple de transformation simple est $abstractName$, qui reçoit un Contexte et produit un Contexte identique mais avec un nom plus abstrait :

$$abstractName :: C \to List\langle C\rangle$$

$$abstractName(n\{v\}) \triangleq [*\,\{v\}]$$

La seule restriction imposée à τ: T est que les Contextes résultants soient bien formés.

4.5.3.1 Apprentissage et oubli

En tout temps, h: H possède un ensemble potentiellement vide $\mathrm{T}_h \subset S_h$ de transformations pouvant être appliquées à du Contexte dans S_h. Certaines de ces transformations se limitent aux noms de Contextes, alors que d'autres constituent des changements de référentiels (p. ex. : une transformation réalisant une conversion de °F à °C).

Le contenu de l'ensemble T_h varie au fil du temps, du fait que les transformations pertinentes pour h peuvent varier selon le contexte; h acquiert et oublie $\tau \in \mathrm{T}_h$, et plus généralement $c \in S_h$, en fonction des besoins et des ressources disponibles. Nous utilisons la notation $\mathrm{T}_{h,t}$ lorsqu'il s'avère pertinent de considérer le moment t auquel T_h est considéré.

4.5.3.2 Équivalence par transformation de Contexte (τ-équivalence)

Le support intrinsèque à la mobilité des agents et des individus que préconise ContextAA a pour conséquence de mettre chaque Agent en contact avec une vaste gamme de formats distincts de Contexte. Pour cette raison, identifier les similitudes entre les Contextes est une approche souvent plus pragmatique que ne le serait la recherche de la correspondance exacte entre deux Contextes.

Il arrive que deux Contextes correspondent l'un à l'autre à une transformation près, donc que :

$$\exists c, c' : C \left(c \neq c' \wedge \left(\exists \tau : \mathrm{T} \left(\left(c = head\left(\tau(c')\right) \right) \vee \left(head\left(\tau(c)\right) = c' \right) \right) \right) \right)$$

Pour rapprocher l'un de l'autre deux Contextes et établir la similitude entre eux, ContextAA automatise certaines transformations de Contextes. Étant donné $c, c' \in C$, c est τ-équivalent à c', exprimé sous la forme suivante :

$$équiv_\tau :: C \to C \to [0..1] \in \mathbb{Q}$$

pour $\tau : \mathrm{T}_h$ avec $\tau = max_{\tau' \in \mathrm{T}_h} \left(equivalence\left(head\left(\tau'(c)\right), c'\right) \right)$, ce qui permet d'exprimer que c est τ-égal à c' sous la forme suivante :

$$égal_\tau :: C \to C \to bool$$

$$égal_\tau(c, c') \triangleq équiv_\tau(c, c') = 1$$

Quelques remarques :

- la fonction $equiv_\tau$ peut ne pas être symétrique;
- puisque T_h varie selon les Hôtes, il est possible d'exprimer $equiv_{\tau,h}$ pour clarifier le propos.

4.5.4 Algorithmes standards

Pour ContextAA, un algorithme $\alpha : \mathrm{A}$ décrit une succession d'opérations sur du Contexte; ces opérations sont souvent paramétrées par un critère. Quelques exemples d'algorithmes suivent :

- étant donné $c : C, \varphi : Crit$, trouver le premier Contexte $x \in \{c \cup value(c)\}$ pour lequel φ s'avère, donc le premier x pour lequel $\varphi(x)$. Notez l'union ensembliste, qui permet de considérer à la fois $name(c)$ et le nom des Contextes de $value(c)$, ce qui est important du fait que, tel que mentionné précédemment, plusieurs opérations sur le Contexte opèrent d'abord sur le nom;
- étant donné $c : C, \varphi : Crit$, trouver $\forall x \in \{c \cup value(c)\} : \varphi(x)$;
- étant donné $c, e : C$, vérifier si $isSubCtx(e, c)$;
- étant donné $c, e : C, \varphi : Crit$, compter $x \in \{c \cup value(c)\} : \varphi(x)$;
- étant donné $c : C, \tau : \mathrm{T}$, appliquer τ à $\forall c' (c' \in \{c \cup value(c)\})$, récursivement, etc.

À titre d'exemple, l'algorithme $contains$ décrit ci-dessous :

$$contains :: Crit \to C \to bool$$

$$contains(\varphi, c) \triangleq \varphi(c) \vee \exists x \left(x \in value(c) \wedge contains(\varphi, x) \right)$$

Exprimé autrement, l'algorithme $contains(\varphi, c)$ est un prédicat binaire qui s'avère seulement si $\varphi(c)$ ou si $contains(\varphi, x)$ pour $x \in value(c)$.

Un autre exemple est l'algorithme $count$ décrit ci-dessous :

$$count :: Crit \to C \to \mathbb{N}$$

$$count(\varphi, c) \triangleq boolToInt\big(\varphi(c)\big) + \sum_{x \epsilon value(c)} count(\varphi, x)$$

L'algorithme $count(\varphi, c)$ compte tous les éléments $x \in \{c \cup value(c)\}$ pour lesquels $\varphi(x)$ s'avère.

Notez que sur le plan de la nécessité, $contains(\varphi, c)$ est rendu redondant par $count(\varphi, c)$ du fait qu'il est possible d'exprimer le premier à partir du second :

$$contains(\varphi, c) \triangleq count(\varphi, c) > 0$$

Cependant, en pratique, la complexité algorithmique moyenne de $contains(\varphi, c)$ sera meilleure que celle de $count(\varphi, c)$ puisque $count(\varphi, c)$ doit consommer c exhaustivement alors que $contains(\varphi, c)$ peut s'interrompre dès que son critère est satisfait, donc aussi tôt que l'algorithme aura déterminé que $\exists x \in \{c \cup value(c)\} : \varphi(x)$.

Évidemment, des algorithmes de recherche sont offerts, tels que $findRoot$:

$$findRoot :: Crit \to C \to List\langle C \rangle$$

$$findRoot(\varphi, c) \triangleq \begin{cases} [c] \text{ if } \varphi(c) \\ map\left(bind(findRoot, \varphi), makeList\big(value(c)\big)\right) \text{ otherwise} \end{cases}$$

qui génère la liste des Contextes $c' \in \{c \cup value(c)\} : \varphi(c')$ sans explorer les sous-Contextes de c', et $findAll$:

$$findAll :: Crit \to C \to List\langle C \rangle$$

$$findAll(\varphi, c) \triangleq \begin{cases} \left[c | map\left(bind(findAll, \varphi), makeList\big(value(c)\big)\right)\right] \text{ if } \varphi(c) \\ map\left(bind(findAll, \varphi), makeList\big(value(c)\big)\right) \text{ otherwise} \end{cases}$$

qui génère une liste exhaustive des Contextes $c' \in \{c \cup value(c)\} : \varphi(c')$ incluant les sous-Contextes de c'. Notez que si φ est une tautologie, alors $findAll$ génère une liste exhaustive de $\forall c' \in \{c \cup value(c)\}$.

Un autre exemple est celui de l'algorithme $apply$ ci-dessous :

$$apply :: \mathrm{A} \to \mathrm{T} \to List\langle C \rangle \to List\langle C \rangle$$

$$apply(\alpha, \tau, [\quad]) \triangleq \alpha(\tau, [\quad])$$

$$apply(\alpha, \tau, [head | tail]) \triangleq concat\big(\alpha(\tau, head), apply(\alpha, \tau, tail)\big)$$

Certains algorithmes accumulent les Contextes pour lesquels un critère s'avère. Supposons l'algorithme $firstOf$ opérant sur une liste de Contextes comme le serait $makeList\big(value(c)\big), c : C$:

$$firstOf :: Crit \to List\langle C \rangle \to C$$

$$firstOf(\varphi, [\quad]) \triangleq \emptyset$$

$$firstOf(\varphi, l) \triangleq \begin{cases} head(l) \text{if } \varphi\big(head(l)\big) \\ firstOf\big(\varphi, tail(l)\big) \text{ otherwise} \end{cases}$$

4.5.5 Relation entre algorithmes et critères

Dans ContextAA, un algorithme α: A exprime une solution générale à un problème de raisonnement sur le Contexte, et procède généralement sur la base d'un critère suppléé à l'utilisation. Combiner $\varphi: Crit$ avec α spécialise le comportement de α.

Ainsi, étant donné $\alpha_i, i = 1..n, \varphi_j, j = 1..m$ où $|\mathrm{A}| = n, |Crit| = m$, le nombre de combinaisons possibles entre α_i, φ_j est $\cong m \times n$.

À titre d'exemple, pour un algorithme tel que $contains(\varphi, c)$, pouvoir spécifier φ indépendamment de $contains$ permet de construire des combinaisons comme :

- est-ce que le nom de $c: C$ ou l'un de ses sous-Contextes décrit une valeur numérique?
- est-ce que le nom de $c: C$ ou l'un de ses sous-Contextes décrit une unité de mesure?
- est-ce que le nom de $c: C$ ou l'un de ses sous-Contextes décrit une température avec une valeur qui, une fois convertie en °C, se situe inclusivement entre le seuil où l'eau bout et celui où l'eau gèle?

4.5.6 Composition d'opérations

Il est possible de combiner des opérations pour construire des opérations plus complexes, qu'il s'agisse de critères ou de transformations. Par exemple :

$$f\_and\_g :: Crit \to Crit \to \big(C \to bool\big)$$

$$f\_and\_g(\varphi, \varphi')(c) \triangleq \varphi(c) \wedge \varphi'(c)$$

$$f\_or\_g :: Crit \to Crit \to \big(C \to bool\big)$$

$$f\_or\_g(\varphi, \varphi')(c) \triangleq \varphi(c) \vee \varphi'(c)$$

$$f\_g :: \mathrm{T} \to Crit \to \big(C \to bool\big)$$

$$f\_g(\tau, \varphi)(c) \triangleq \varphi\left(head\big(\tau(c)\big)\right)$$

Des opérations telles que f_and_g, f_or_g ou f_g sont des combinateurs. La structure du Contexte tel que nous le définissons fait de son parcours une opération coûteuse, et pouvoir combiner en une seule entité plusieurs opérations permet d'accélérer ces parcours.

À titre d'exemple, supposons la séquence d'opérations suivante :

$$\text{Let } c, c', c'': C; \varphi, \varphi': Crit; \alpha: \mathrm{A}$$

$$c' = \alpha(\varphi, c)$$

$$c'' = \alpha(\varphi', c')$$

Supposons que α soit un algorithme tel que $count$ pour lequel un parcours exhaustif du Contexte auquel il est appliqué doit être fait. Pour simplifier l'exemple, supposons que α ne filtre pas les éléments du Contexte auquel il est appliqué, de sorte que $size(c') = size(c)$. Dans un tel cas, α doit réaliser deux parcours complets d'un Contexte de la taille de c.

Par contre, en utilisant un combinateur tel que f_g, nous obtenons :

$$\text{Let } c, c'' : C; \varphi, \varphi' : Crit; \alpha : \text{A}$$

$$c'' = \alpha(f\_g(\varphi, \varphi'), c)$$

où c' « disparaît », étant relégué au rôle d'état transitoire du calcul. Ceci permet de réaliser la totalité des opérations en un seul parcours, une économie substantielle.

4.5.7 Exemples d'applications

Supposons $matchName$ présenté dans §4.5.2 :

$$\text{Let } c = température\{unite\{Celsius\}\ valeur\{10\}\}$$

$$contains(matchName("Celsius"), c) = true$$

$$contains(matchName("unite"), c) = true$$

$$contains(matchName("Fahrenheit"), c) = false$$

$$contains\big(f\_or\_g\big(matchName("Fahrenheit"), matchName("Celsius")\big), c\big) = true$$

Un algorithme peut bien sûr être utilisé dans un autre algorithme. Par exemple :

$$reapply :: \text{A} \to Crit \to \Big(C \to bool\Big)$$

$$reapply(\alpha, \varphi)(c) \triangleq \alpha(\varphi, c)$$

$$findFirst :: Crit \to C \to C$$

$$findFirst(\varphi, c) \triangleq \begin{cases} c \text{ if } \varphi(c) \\ \emptyset \text{ if } terminal(c) \wedge \neg\varphi(c) \\ firstOf\left(reapply(findFirst, \varphi), makeList\big(value(c)\big)\right) \text{ otherwise} \end{cases}$$

Ici, $reapply$ est un foncteur ou une fonction de haut niveau dont le fruit de l'application est un critère.

Comprendre la mécanique des opérations sur le Contexte dans le cadre de ContextAA est nécessaire pour en profiter pleinement. Supposons par exemple un Hôte sur lequel certains Agents seulement possèdent un Contexte nommé $Celsius$. Alors, la requête suivante :

$$has\{** \{Celsius\}\}$$

mènera à la réponse duale $true$ et $false$ du fait que (a) pour au moins un Agent, la réponse est $true$, (b) pour au moins un Agent, la réponse est $false$, et (c) les noms de Contexte sont uniques dans un ensemble donné, ce qui élimine *de facto* la redondance.

De même, en supposant que l'agent $tempProvider$ soit l'un de ceux possédant le Contexte $Celsius$, la requête suivante :

$$has\left\{tempProvider\{**\{Celsius\}\}\right\}$$

mènera aussi à la réponse duale $true$ et $false$ du fait que le résultat de son traitement sera $true$ pour $tempProvider$ (par hypothèse) et $false$ pour tous les autres (les Agents qui ne portent pas le nom racine $tempProvider$ ne « possèdent » pas le Contexte demandé, mais cette non-possession fait partie du résultat de la requête). Si le souhait est d'exprimer une requête s'avérant seulement pour $tempProvider$, la forme composite suivante :

$$compose\left\{\#0\{tempProvider\}\ \#1\ \left\{has\{**\{Celsius\}\}\right\}\right\}$$

est celle qui donnera les résultats souhaités, filtrant d'abord les Agents non-désirés puis appliquant le prédicat has sur le Contexte résultant.

4.5.8 Évaluation d'expressions

Il est possible pour un Contexte d'exprimer une évaluation d'expressions arithmétiques, relationnelles ou logiques de manière à interagir avec le dictionnaire de modèles. Le nom de Contexte $eval$ est réservé à cet effet, et définit un sous-langage sur le Contexte.

Ce sous-langage s'exprime à l'aide d'un Contexte $eval\{exprs\}$, où $exprs$ est une séquence d'expressions dont la forme est décrite àl'annexe B. Les guillemets peuvent être insérés autour de $exprs$ pour lever une ambiguïté : par exemple, les évaluations suivantes résultent toutes en un Contexte de nom 8 :

$$eval\{8\}$$

$$eval\{3 + 5\}\ (sans\ espaces)$$

$$eval\{"\ \ 3\ \ +\ \ 5\ \ "\}\ (avec\ espaces)$$

$$eval\ \{"\backslash"a\ b\backslash"\ =\ 3;\backslash"a\ b\backslash"\ +\ 5"\}\ (nom\ de\ Contexte\ avec\ espaces)$$

Le fruit de l'évaluation d'un Contexte $eval\{exprs\}$ est la valeur du dernier Contexte évalué, autre qu'un $commit$. La valeur d'une expression $eval$ se limitant à un $commit$ est $\emptyset$.

Un Contexte c dont le nom est $eval$ doit être tel que $nelems\big(value(c)\big) = 1$. La valeur de c dans ce cas est le résultat de l'évaluation de $name\big(value(c)\big)$.

Par exemple :

Table 2 - Exemples d'évaluation d'expressions par ***eval***

Contexte	**Résultat de l'évaluation**
$eval\{"2 + 3"\}$	La représentation de 5 sous forme de Contexte
$eval\{"x = 2; y = 3; x + y"\}$	La représentation de 5 sous forme de Contexte
$eval\{"x = 2; x = x + 1; commit\ x"\}$	La représentation de 3 sous forme de Contexte
$eval\{"1 > 3"\}$	La représentation de $false$ sous forme de Contexte
$eval\{"c = a == b? 3: -8.5"\}$	Attribue à une variable c la valeur 3 si la valeur de a est égale à celle de b, et lui attribue la valeur -8.5 dans le cas contraire. Ici, a et b seraient des noms pris du dictionnaire de modèle. La valeur de c suite à l'affectation sera le résultat de l'évaluation de l'expression entière
$eval\{"a = 3; b = -2; c = a * b; commit"\}$	Ici, les valeurs des variables a, b et c sont intégrées au dictionnaire de modèles et remplacent toute valeur du même nom qui s'y trouverait déjà. La valeur du Contexte résultant est celle de la dernière évaluation autre qu'un $commit$, ce qui permet d'enchaîner le Contexte résultant dans une séquence d'opérations (Contexte $compose$)
$eval\{"a = 3; b = -2; c = a * b; commit\{a\ c\}"\}$	Ici, les valeurs des variables a et c sont intégrées au dictionnaire de modèles et remplacent toute valeur du même nom qui s'y trouverait déjà. La valeur du Contexte résultant est celle de la dernière évaluation autre qu'un $commit$

Le sous-langage d'un Contexte $eval$ supporte, avec une syntaxe analogue à celles de C, C++, Java ou C#, les opérateurs arithmétiques ($+$ et $-$ à la fois unaires et binaires, de même que $*, /, \%$), logiques ($\&\&, ||, !$), relationnels ($<, <=, >, >=, ==, !=$), l'affectation ($=$), le groupement à l'aide de parenthèses, et permet d'utiliser le ; pour séparer une expression en sous-expressions (format : $expr_0; expr_1; \cdots; expr_n$). La valeur de l'expression entière est la valeur de la dernière sous-expression évaluée : $eval\{"x = 3; y = 4; "\}$ équivaut au Contexte de nom 4, alors que $eval\{""\}$ correspond à un Contexte vide.

Les expressions $eval$ peuvent être combinées à d'autres expressions de Contexte pour former des expressions plus complexes. Par exemple, $compose\left\{\#0\{eval\{"2 + 3"\}\}\#1\{keep\_if\{is\_number\}\}\right\}$ résultera en un Contexte de nom 5, alors que $compose\left\{\#0\{eval\{"2 + 3"\}\}\#1\{keep\_if\{is\_date\}\}\right\}$ résultera en un Contexte vide.

Enfin, une expression telle que

$$compose\left\{\#0\ \left\{*\right.\right.$$
$$\left.\left.*\left\{battery\left\{**\left\{remaining\left\{value\ \{?=val\}\right\}\right\}\right\}\right\}\ \#1\left\{eval\left\{"val\ >\ 50"\right\}\right\}\right\}$$

permet de savoir si le niveau de la pile d'un appareil est supérieur à 50%.

4.5.8.1 Expressions *commit*

L'ajout du nom x et de sa valeur au dictionnaire de modèles dans un Contexte $eval$ se fait par une expression de la forme $commit\{x\}$ ou $eval\{"x = 3;\ commit\ x"\}$, ce qui équivaut à $eval\{"x = 3"\}$ ou à $eval\{"3"\}$ à ceci près que x est un nom de Contexte conservé dans le premier cas alors que 3 est le nom du Contexte conservé dans les deux autres cas. L'ajout au dictionnaire de modèles de tous les noms créés dans un Contexte $eval$ se fait par $commit$ tout simplement. Une expression $commit$ peut insérer plusieurs Contextes x,y,z avec $eval\{" \ldots ; commit\ \{x\ y\ z\}"\}$.

Des expressions telles que $eval\{"x = x + 1; commit\ x"\}$ sont permises, par exemple pour incrémenter la valeur associée à un nom x dans le dictionnaire d'un Agent.

4.6 Similitudes entre Contextes

Le modèle mis de l'avant par ContextAA préconise que chaque Agent soit porteur de sa représentation propre du Contexte, mais que les Agents partagent une syntaxe commune pour permettre aux Hôtes de réaliser une mise en correspondance des Contextes produits et requis sur cette base.

Il est possible d'exprimer l'équivalence pleine et entière (booléenne, contrairement à $equivalence$ qui est rationnelle) d'un Contexte à un autre comme suit :

$$fullyEquivalent :: C \to C \to bool$$

$$fullyEquivalent(n\{v\}, n'\{v'\}) \triangleq \begin{pmatrix} equivalence(n, n') = 1 \wedge \\ size(v) = size(v') \wedge \\ \forall x \in v\left(\exists x' \in v'\left(fullyEquivalent(x, x')\right)\right) \end{pmatrix}$$

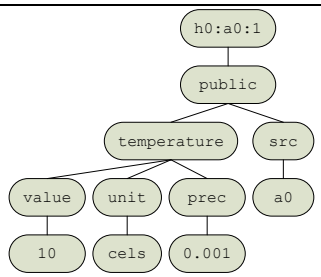
L'équivalence pleine et entière ainsi définie ne couvre toutefois pas tous les cas dignes d'intérêt. En pratique, un Agent doit être en mesure de traiter des équivalences partielles. Dans des environnements riches et diversifiés tels les systèmes ubiquistes urbains, la capacité qu'a un Agent de s'adapter à son milieu accroît les probabilités qu'il puisse accomplir sa mission, et cette adaptation passe entre autres par une flexibilité quant au format des Contextes rencontrés.

À moins de travailler à partir d'une ontologie fixée *a priori*, comprendre le Contexte implique comprendre les relations qui s'y dessinent.

Un exemple simple mais illustratif de la situation serait :

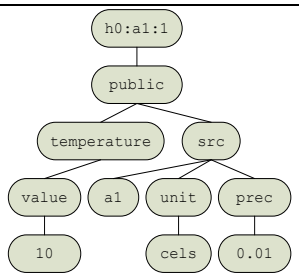
c_0

```
h0:a0:1 {
  public {
    temperature {
      value { 10 }
      unit  { celsius }
      precision { 0.001 }
    }
    source { a0 }
  }
}
```



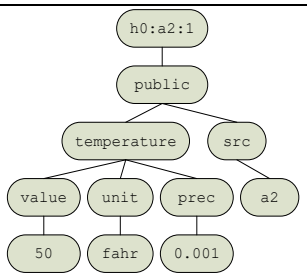
c_1

```
h0:a1:1 {
  public {
    temperature {
      value { 10 }
    }
    source {
      a1
      unit { celsius }
      precision { 0.01 }
    }
  }
}
```



c_2

```
h0:a2:1 {
  public {
    temperature {
      value { 50 }
      unit { fahrenheit }
      precision { 0.001 }
    }
    source { a2 }
  }
}
```



Pour un être humain, il est sans doute évident qu'outre leurs précisions respectives, c_0 et c_1 décrivent la même information à travers des structures distinctes, alors que c_2 décrit la même information que c_0 mais avec une valeur différente, dû à un autre choix d'unité de mesure.

À partir de la définition que fait ContextAA de l'équivalence entre Contextes, ces trois Contextes sont différents l'un de l'autre, leurs noms étant différents. Cette différence ne signifie pas qu'ils décrivent des réalités radicalement différentes. Tout est dans la question : s'agit-il chaque fois de la même chose? Non. Décrivent-ils la même réalité de différentes manières? Oui et non; ici, tout dépend des attentes de l'Agent posant la question, et de la manière qu'aura l'Agent d'utiliser les Contextes recueillis.

En fait, les différences entre c_0 et c_1 sont que c_0 représente l'unité de mesure et la précision comme faisant partie de la description de la température, alors que c_1 représente ces deux Contextes comme faisant partie de la description de l'Agent producteur du Contexte. Chacune de ces expressions peut être préférable aux autres dans certains contextes : par exemple, si un Agent publie deux Contextes sémantiquement équivalents sous des formes syntaxiquement distinctes, les formes c_0 et c_2 peuvent être appropriées; si un autre Agent collige des Contextes consommés de diverses sources, alors il se peut que c_1 soit préférable.

Il est possible que ces différences soient porteuses de sens, comme il est possible qu'elles ne le soient pas :

- si un Agent consommateur est en mesure de traduire automatiquement de °F à °C, alors les options c_0 et c_2 se valent pour lui, bien qu'un Agent puisse privilégier l'une ou l'autre des formes dans l'optique de réduire sa charge de calcul;

- un Agent privilégiant une capture de la plus haute précision possible et placé devant c_0 et c_1 choisirait probablement c_0, mais placé devant c_1 et c_2 la décision pourrait tenir à d'autres facteurs, dont la capacité de transformer le référentiel dans au moins l'un de ces cas;
- un Agent peut s'attendre à un Contexte associant la précision à l'Agent producteur et ne pas avoir prévu le cas où la précision aurait été associée à la mesure. Dans ce cas, placé devant ces options, seul c_1 serait adéquat; etc.

Nous acceptons que les Agents producteurs expriment le Contexte selon leur cadre ontique, leurs choix de vocabulaire et de structure, tout comme nous acceptons que les consommateurs demandent le Contexte selon leur cadre ontique, leurs choix de vocabulaire et de structure. Le Contexte est pour nous syntaxique, et la sémantique associée est subjective, au sens où elle varie selon l'Agent qui l'observe. Ce sont les Hôtes, par la médiation de leurs Agents standards, qui sont responsables de la mise correspondance de l'un avec l'autre.

ContextAA n'impose pas d'*a priori* quant à l'importance des différences entre les Contextes, et ne définit pas de prime abord le sens à donner à ces différences. L'optique du *Contexte-selon* signifie que ce qui importe pour un Agent peut être secondaire pour un autre.

Plusieurs facteurs participent à l'évaluation de la qualité d'un Contexte pour un Agent. ContextAA n'est pas responsable des choix quant à la qualité relative des Contextes pour un Agent; c'est l'Agent qui est responsable d'évaluer la qualité relative des Contextes à sa disposition. Dans ContextAA, l'évaluation de la qualité relative des Contextes est subjective.

ContextAA offre par contre des outils pour quantifier la distance relative entre deux Contextes. Ces outils sont décrits dans les sections suivantes. Des exemples de l'impact de chacune suivront, et des tests plus détaillés sont livrés aux annexes G et H.

4.6.1 Distance sémantique – discussion informelle

Les mécanismes de mise en correspondance décrits à §5.4 portent fruit lorsque l'Agent consommateur connaît quelque peu la structure du Contexte souhaité, et quand cette structure rejoint celle par laquelle s'exprime un Agent fournisseur. Dans notre approche de Contexte-selon, chaque Agent est responsable de ses propres choix de représentation, ce qui réduit les probabilités d'une correspondance structurelle immédiate entre Contexte publié et Contexte demandé.

Pour déterminer une mesure quantitative de la distance entre ce qu'expriment deux Contextes, nous quantifions leur distance sémantique. Informellement, l'objectif est de définir la distance sémantique ρ ayant la signature suivante :

$$\rho :: C \to C \to \mathbb{Q} \in [0..1]$$

L'idée est que la distance sémantique entre $c, c' : C$ se veut une mesure de la distance entre ce que c et c' représentent. Dans cette section, nous examinons d'abord les caractéristiques attendues d'une bonne fonction ρ, puis nous examinons ce que propose ContextAA en détail.

Quelques critères s'appliquent d'office pour ρ. Le premier est que si c et c' sont pleinement équivalents l'un à l'autre, alors la distance sémantique entre eux doit être nulle. Pour illustrer cette réalité avec un cas simple, avec $c : C$, il faut que :

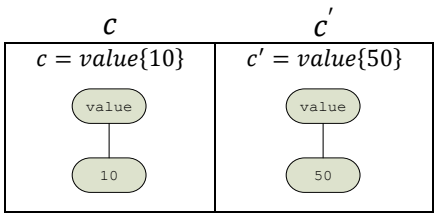
$$\rho(c, c) = 0$$

Lorsque c et c' sont partiellement équivalents, nous souhaitons que $\rho(c, c')$ le reflète : plus c et c' sont semblables et plus petite devrait être leur distance sémantique.

Dans le cas simple de Contextes terminaux, une définition intuitive serait :

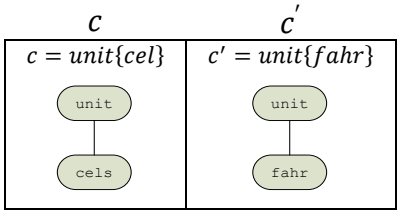
$$\rho(n\{\emptyset\}, n'\{\emptyset\}) = 1 - equivalence(n, n') = distance(n, n')$$

Examinons de plus près le comportement attendu de $\rho(c, c')$ dans le cas où c et c' sont non-terminaux. Un cas simple serait :

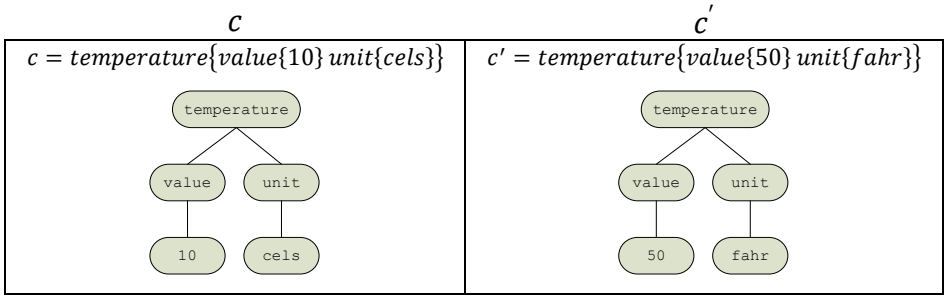


Il existe ici une relation sémantique apparente entre c et c', les deux portant un nom équivalent : en surface, c et c' sont égaux. Cependant, il est difficile d'évaluer quelle devrait être la valeur de $\rho(c, c')$ sur la base de ce seul exemple, outre qu'elle devrait être non-nulle.

La situation est semblable avec ce qui suit. Il existe une relation sémantique entre c et c' mais celle-ci est difficile à quantifier (sont-ils proches ou loin l'un de l'autre?) :



Visiblement, la valeur que nous parvenons à donner à une bonne définition de ρ gagne en pertinence si ses paramètres sont riches en contenu; il est difficile de déterminer la distance sémantique entre Contextes *a priori* pauvres ou de petite taille. Intégrer les deux exemples ci-dessus dans une même structure nous donne toutefois plus d'information et permet d'en arriver à un jugement plus éclairé :



À l'œil humain, il semble clair que c et c' décrivent une même situation, du moins pour cette parcelle à laquelle nous sommes confrontés (pris dans un contexte plus large, il est possible que les réalités décrites soient bien différentes). Ceci met toutefois en valeur une caractéristique du Contexte-selon : si les Contextes exprimés ici répondent aux questions posées par un Agent, alors il est légitime qu'ils soient considérés équivalents par cet Agent.

Un résultat tel que $\rho(c, c') = 0$ dans ce cas est approprié dans la mesure où les transformations requises pour passer de c à c' peuvent être appliquées par $a \in h$ sont telles que pour a, c et c' sont τ-équivalents (§4.5.3.2) pour un τ tenant compte de l'entièreté de c et c'.

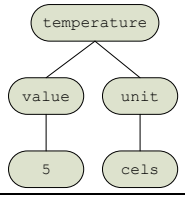
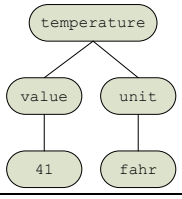
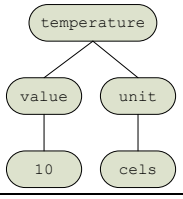
De ce survol informel du problème émergent quelques informations :

- le calcul de la distance sémantique $\rho(c, c')$ bénéficie d'un effet Gestalt, au sens où des Contextes qui semblent *a priori* distincts (sémantiquement distants) peuvent, si examinés dans leur contexte, s'avérer sémantiquement proches l'un de l'autre, peut-être même sémantiquement identiques;
- la valeur de $\rho(c, c')$ dépend de la qualité relative des Contextes qui lui sont fournis. Le terme « qualité » est délibérément laissé vague ici, étant directement associé au contexte ayant mené à l'acquisition du Contexte : description de la requête, Contexte disponible lors de l'acquisition, besoins de l'Agent demandeur, richesse relative du Contexte décrivant la requête, etc. Par exemple, la requête $** \{valeur\}$ peut mener à un nombre élevé de réponses, la seule contrainte étant que le nom $valeur$ apparaisse dans chacun des Contextes acquis, alors qu'il est difficile de tirer un constat clair de Contextes tels que 10 et 50 s'ils sont pris hors de leur contexte;
- la valeur calculée par la fonction $\rho(c, c')$ variera selon le contexte, en particulier selon le moment et les circonstances. Nous écrirons $\rho_h(c, c')$ la fonction $\rho(c, c')$ telle qu'elle s'évalue sur $h: H$. Par exemple, si $\exists \tau: \mathrm{T}\big(c' \cong \tau(c)\big)$ sur $h: H$ mais pas sur $h': H$, alors $\rho_h(c, c') < \rho_{h'}(c, c')$.

Une application directe de $\rho(c, c')$ est le cas où un Agent a compare $c, c': C$ pour voir si les deux sont en accord quant à la description d'un état, par exemple la température ambiante, ou encore pour évaluer, peut-être à partir d'autres Contextes, lequel des deux est le plus probable selon lui.

Une autre application typique est le cas où a connaît *a priori* $c: C$, a obtenu $c', c'': C$, et cherche à identifier lequel d'entre eux est le plus semblable à c.

Examinons ce second cas. Supposons les Contextes suivants :

c	c'	$c'' = \tau(c') \cong c$
$c = temperature$ $\{value\{5\}\, unit\{cels\}\}$	$c' = temperature$ $\{value\{41\}\, unit\{fahr\}\}$	$c'' = temperature$ $\{value\{10\}\, unit\{cels\}\}$
		

Dans cet exemple, les Contextes c et c' sont τ-équivalents pour un τ décrivant une conversion $°F \rightarrow °C$. Les Contextes c' et c'' décrivent une réalité semblable à un changement de référentiel près.

Les Contextes c et c'' sont structurellement identiques mais diffèrent sur le plan de la valeur exprimée. Ce cas d'utilisation correspond à une situation où deux capteurs auraient lu des valeurs connexes et où l'Agent consommateur devrait faire un choix à savoir laquelle est « la meilleure » :

- si a est en mesure de traduire du référentiel °F au référentiel °C, alors le Contexte le plus proche de c pour a est c';

- si a n'est pas en mesure de traduire du référentiel °F au référentiel °C, alors le Contexte le plus proche de c pour a est c''.

Cet exemple met en relief la différence entre équivalence structurelle et équivalence de contenu. Ici, c est structurellement équivalent à c'' car les deux ont les mêmes noms jouant les mêmes rôles pour tout Contexte dans c et c'' à une profondeur donnée, mais ils ne sont pas équivalents en terme de contenu car ils ne sont pas τ-équivalents.

L'application d'une transformation τ sur c ou c' peut mettre en relief que c'est entre eux que la similitude sémantique est la plus forte. Pour qui se limite à la similitude structurelle, le lien est plus fort entre c et c''. Les deux relations ont leur importance; ici, appliquer $\tau(c')$ peut être vu comme une manière de réduire la distance sémantique entre c et c'.

4.6.2 Distance structurelle

Soit les fonctions auxiliaires $smallest$ et $largest$ ci-dessous, qui permettent toutes deux de définir un ordonnancement entre deux Contextes :

$$smallest :: C \to C \to Set\langle C \rangle$$

$$smallest(n\{v\}, n'\{v'\}) \triangleq \begin{cases} v \text{ if } size(v) < size(v') \vee size(v) = size(v') \wedge n < n' \\ v' \text{ otherwise} \end{cases}$$

$$largest :: C \to C \to Set\langle C \rangle$$

$$largest(n\{v\}, n'\{v'\}) \triangleq \begin{cases} v \text{ if } size(v) > size(v') \vee size(v) = size(v') \wedge n > n \\ v' \text{ otherwise} \end{cases}$$

Nous définissons la distance structurelle $\sigma(c, c')$ entre $c, c' : C$ comme suit :

$$\sigma :: C \to C \to \mathbb{Q} \in [0..1]$$

$$\sigma(n\{\emptyset\}, n'\{v'\}) \triangleq 1$$

$$\sigma(n\{v\}, n'\{\emptyset\}) \triangleq 1$$

$$\sigma(n\{\emptyset\}, n'\{\emptyset\}) \triangleq distance(n, n')$$

$$\sigma(n\{v\}, n'\{v'\}) \triangleq \left\{ \begin{array}{c} \text{Let } small = smallest(v, v'), large = largest(v, v'), \\ szsmall = size(small), szlarge = size(large) \\ \left(\dfrac{distance(n, n') + \left(1 - \dfrac{szsmall}{szlarge}\right) + \dfrac{\sum_{e \in small} \left(min_{e' \in large}(\sigma(e, e')) \right)}{szsmall}}{3} \right) \end{array} \right.$$

La distance structurelle σ est une mesure composite qui tient compte de l'équivalence des noms des Contextes comparés, de la cardinalité de leurs valeurs respectives et de la distance structurelle des éléments des valeurs de ces Contextes.

La commutativité de σ est préservée par l'ordonnancement lexicographique des noms de Contexte dans le cas où $size\big(value(c)\big) = size\big(value(c')\big)$ dans les fonctions $smallest$ et $largest$.

Tel que nous la définissons, la distance structurelle est sensible aux variations de structure et ne tient pas compte de mises en correspondance de modèles. Ainsi, bien que si $c = a\{*\}, c' = a\{b\{c\}\}$ $\Rightarrow c = c'$, il se trouve que $\sigma(c, c') > 0$ de par les distinctions structurelles entre les Contextes comparés.

4.6.2.1 Équivalence structurelle – discussion informelle

Définissons informellement l'équivalence structurelle, sur la base de la distance structurelle $\sigma(c, c')$ entre c et c', telle que

$$\sigma :: C \to C \to \mathbb{Q} \in [0..1]$$

Nous définissons l'équivalence structurelle comme équivalente à une distance structurelle nulle, donc au cas $\sigma(c, c') = 0$, de manière telle que $c, c' : C$ soient structurellement équivalents si :

- $depth(c) = depth(c')$;
- $equivalence\big(name(c), name(c')\big) = 1$;
- $size\big(value(c)\big) = size\big(value(c')\big)$; et
- $\forall e \in value(c) : terminal(e) \vee \exists e' \in value(c')(\sigma(e, e') = 0)$.

Tout comme la distance structurelle, l'équivalence structurelle est une fonction commutative.

4.6.3 Distance de contenu

Soit la fonction auxiliaire $bestEquiv$ ci-dessous, qui repère l'élément dans un ensemble de Contextes donné qui est le plus proche en termes de contenu d'un Contexte c :

$$bestEquiv :: C \to \mathrm{T} \to \mathrm{T} \to Set\langle C\rangle \to \mathbb{Q} \in [0..1]$$

$$bestEquiv(c, \tau, \tau', \emptyset) \triangleq 1$$

$$bestEquiv(c, \tau, \tau', v) \triangleq \min_{e \in v} \Big(\varrho \left(head\big(\tau(c)\big), head\big(\tau'(e)\big) \right) \Big)$$

Notez que $bestEquiv$ utilise ϱ définie ci-dessous; la relation entre les deux en est une d'interdépendance. Pour simplifier l'écriture, nous présumons pour ce qui suit que $\tau = \tau' = identity\langle T\rangle$ et que ces transformations sont telles que la liste résultante n'a qu'un seul élément. Ceci nous permettra d'omettre les paramètres τ, τ' qui gardent toutefois leur pertinence dans le cas général.

Nous définissons la distance de contenu entre $c, c' \in C$ de la manière suivante :

$$\varrho :: C \to C \to \mathbb{Q} \in [0..1]$$

$$\varrho(n\{v\}, n'\{\emptyset\}) \triangleq distance(n, n')$$

$$\varrho(n\{\emptyset\}, n'\{v'\}) \triangleq distance(n, n')$$

$$\varrho(n\{v\}, n'\{v'\}) \triangleq \left(\frac{distance(n, n') + \left(\frac{\sum_{e \in v} bestEquiv(e, v')}{size(v)} \right)}{2} \right)$$

Il est possible que pour certaines paires $c, c' : C$, il y ait plus d'un élément $e \in value(c)$ pour lesquels un même élément $e' \in value(c')$ soit le meilleur équivalent possible.

4.6.3.1 Équivalence de contenu – discussion informelle

Définissons informellement l'équivalence de contenu, sur la base de la distance de contenu $\varrho(c, c')$ la distance de contenu entre c et c', telle que :

$$\varrho :: C \to C \to \mathbb{Q} \in [0..1]$$

Nous définissons l'équivalence de contenu comme équivalente à une distance de contenu nulle, donc au cas $\varrho(c, c') = 0$, de manière telle que $c, c' : C$ soient équivalents sur le plan du contenu si :

- $equivalence\big(name(c), name(c')\big) = 1$; et
- $\forall e \in c (\exists e' \in c' : \varrho(e, e') = 0)$.

4.6.4 Interpréter la distance structurelle

Quelques remarques suivent quant à l'interprétation des valeurs obtenues par l'application de $\sigma(c, c')$:

- obtenir $\sigma(c, c') = 0$ équivaut à établir l'existence d'une équivalence ou d'une subsomption de chaque Contexte de c à une profondeur donnée avec un Contexte de c' à la même profondeur, et inversement. Visuellement, il serait possible de superposer c et c' pour que chaque nœud soit superposé avec un nœud équivalent ou avec lequel il existe une relation de subsomption;
- en contrepartie, $\sigma(c, c') = 1$ équivaut à constater que c ne semble pas montrer de similitude structurelle avec c' : leurs noms ne sont pas équivalents, leurs valeurs ne sont pas de même taille, et si les deux Contextes sont non-terminaux, alors aucun élément de c ne semble avoir de similitude structurelle avec un élément de c';
- tout comme ϱ , σ accorde un poids prépondérant aux noms et aux valeurs situés plus près de la racine d'un Contexte. Ainsi, que $\sigma(c, c')$ soit près de 0 suggère que c et c' soient structurellement proches dès la racine, mais il arrive que, même si la valeur de $\sigma(c, c')$ s'approche de 1, il soit possible de rapprocher c de c' sur le plan structurel en appliquant des transformations comme celles listées dans §4.6.6.

Le recours à $\sigma(c, c')$, à l'image du recours à $\varrho(c, c')$, permet à un Agent de discriminer quant à la proximité relative de la structure d'un Contexte au contenu d'un autre. La métrique résultante est sensible aux similitudes à proximité de la racine des Contextes comparés, ce qui suggère qu'il faille porter attention à la forme de ces Contextes pour en tirer maximalement profit.

4.6.5 Interpréter la distance de contenu

Quelques remarques suivent quant à l'interprétation des valeurs obtenues par l'application de $\varrho(c, c')$:

- obtenir $\varrho(c, c') = 0$ équivaut à reconnaître l'inclusion de c dans c', aux profondeurs correspondantes, mais ne signifie pas que c et c' soient équivalents du fait que ϱ n'est pas commutative. Par exemple, $\varrho(a, a\{b\}) = 0$ alors que $\varrho(a\{b\}, a) = \frac{1}{2}$;
- bien sûr, $\exists c, c' : C\big(\varrho(c, c') = \varrho(c', c)\big)$ mais il s'agit d'exceptions;

- obtenir $\varrho(c, c') = 1$ signifie que le contenu de c semble ne pas être en lien avec le contenu de c' pour qui base sa perspective sur c;
- de manière générale, étant donné $c: C(depth(c) = d)$, les noms de Contextes aux profondeurs plus proches de 1 ont un poids plus élevé dans le calcul de ϱ que ceux aux profondeurs plus proches de d. En effet, dans $\varrho(n\{v\}, n'\{v'\})$, le poids de $distance(n, n')$ compte pour la moitié de la valeur évaluée par la fonction alors que chacun des noms des Contextes dans v et v' compte pour une fraction de l'autre moitié;
- pour profiter de ϱ en pratique, il importe de comprendre la relativité des poids des noms : s'il existe une certitude *a priori* quant aux noms racines (ceux de profondeur proche de 1), comme dans le cas où il est connu au préalable que des noms seront différents, provenant par exemple d'Agents distincts sur des Hôtes distincts, conserver ces noms ou les retirer avant d'évaluer ϱ influencera le sens donné aux valeurs calculées. Dans le cas de noms racines distincts pour c et c', conserver ces noms pour le calcul de $\varrho(c, c')$ signifie que la valeur minimale obtenue sera $\frac{1}{2}$, donc qu'un résultat tel que $\varrho(c, c') = 0{,}6$ peut signifier une relativement faible différence dans les contenus de $value(c)$ et $value(c')$.

Le recours à $\varrho(c, c')$ permet à un Agent de discriminer quant à la proximité relative du contenu d'un Contexte au contenu d'un autre. Pour en tirer profit au maximum, il importe de tenir compte de l'aspect non-commutatif des paramètres passés à la fonction dans le choix de l'ordre des Contextes qui lui sont fournis, et de tenir compte du poids plus important des Contextes plus près de la racine dans l'évaluation des résultats.

4.6.6 Discussion

À la lueur de cette définition, il peut sembler tentant d'exprimer l'image de ϱ dans $[-1..1]$ pour que le signe de la fonction permette de déterminer lequel d'entre c et c' est le Contexte le plus riche, mais il existe plusieurs cas où $\varrho(c, c') > 0 \wedge \varrho(c', c) > 0$, comme par exemple le cas $c = a\{b\}, c' = a\{d\}$. Une valeur non-nulle pour $\varrho(c, c')$ peut être vue comme signifiant qu'une partie de c n'est pas aussi une partie de c'.

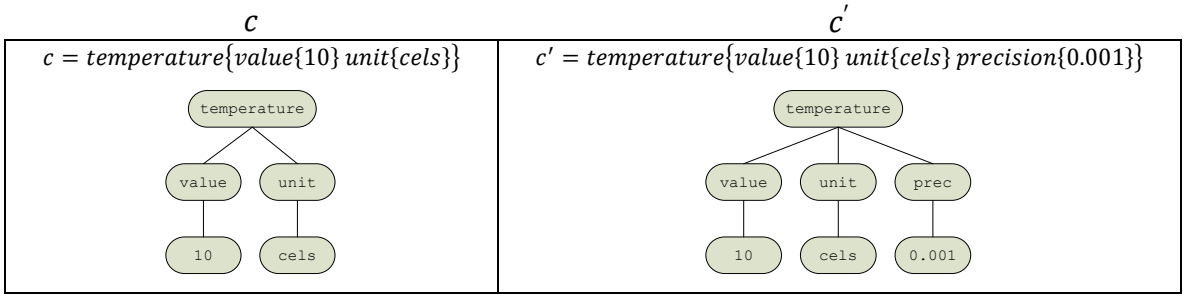
Nous n'exigeons pas que ϱ soit commutative. Dans un cas comme celui où c serait un strict sous-ensemble de c', il s'avérerait que $\varrho(c, c') = 0$ mais $\varrho(c', c) \neq 0$. De même :

$$terminal(c) \vee \forall e \in value(c)(\exists e' \in value(c'): \varrho(e, e') = 0)$$

n'implique pas que

$$terminal(c') \vee \forall e' \in value(c')(\exists e \in value(c): \varrho(e', e) = 0)$$

À titre d'illustration :



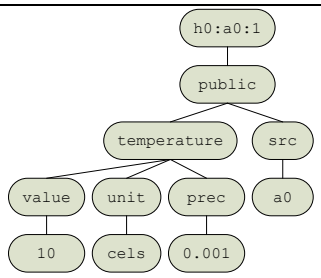
Étant donné c et c' ci-dessus, il y a lieu de s'attendre à ce que $\sigma(c, c') > 0$ dû à l'existence de différences structurelles entre ces deux Contextes (plus exactement : $size\big(value(c)\big) \neq size\big(value(c')\big)$). Il y a aussi lieu de s'attendre à ce que $\varrho(c, c') = 0 \wedge \varrho(c', c) \neq 0$ puisque tous les éléments de $value(c)$ sont aussi dans $value(c')$ alors qu'il existe un élément de $value(c')$, plus précisément le Contexte $precision$, qui n'apparaît pas dans $value(c)$.

Sans aller dans le détail pour le moment, en établissant que la distance sémantique est fonction à la fois de la distance structurelle et de la distance de contenu, donc que $\rho(c, c') = f\big(\sigma(c, c'), \varrho(c, c')\big)$, nous savons que ρ ne sera pas commutative.

Examinons à nouveau c_0 et c_1. Ces Contextes sont structurellement différents mais porteurs d'un sens similaire :

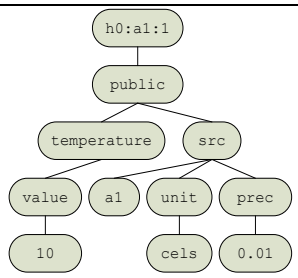
c_0

```
h0:a0:1 {
  public {
    temperature {
      value { 10 }
      unit  { cels }
      precision { 0.001 }
    }
    source { a0 }
  }
}
```



c_1

```
h0:a1:1 {
  public {
    temperature {
      value { 10 }
    }
    source {
      a1
      unit { cels }
      precision { 0.01 }
    }
  }
}
```



Les principales différences entre c_0 et c_1 sont :

- la manière par laquelle l'Agent producteur se décrit lui-même. Puisque ces deux Agents se nomment respectivement $a0$ et $a1$ et puisque chacun représente ce fait dans le Contexte qu'il publie, c_0 et c_1 sont différents en termes de contenu. L'abstraction de racine permet de réduire l'impact de cette différence si elle n'est pas significative pour l'Agent demandeur;
- la précision de la mesure telle qu'elle est exprimée dans c_0 et dans c_1 (c_0 semble plus précis que c_1); et
- la façon qu'a chacun des Agents de décrire le Contexte. Pour $a0$, la précision et l'unité de mesure font partie de $temperature$, alors que pour $a1$ elles font partie de $source$.

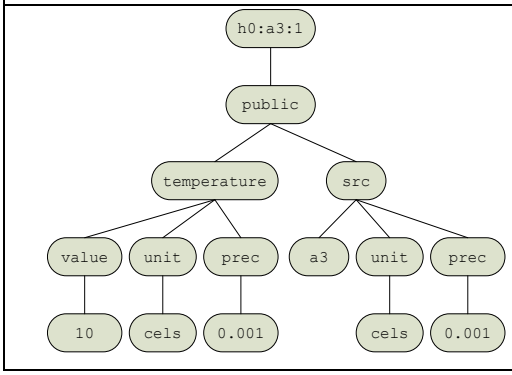
D'autres représentations sont possibles, évidemment, dont une où l'Agent producteur placerait $source$ dans $temperature$, ce qui pourrait être utile si un même producteur fournissait le Contexte décrivant plusieurs lectures sur plusieurs types de capteurs.

Notre objectif est de permettre à un Agent de déduire que c_0 et c_1 semblent décrire la même idée, mais que c_0 semble offrir une lecture de meilleure précision que c_1. Nous voulons, avec $\rho(c_0, c_1)$, évaluer la proximité des idées que ces Contextes décrivent; d'autres outils permettront d'établir que c'est sur la base de la précision qu'il sera possible de discriminer.

Nous souhaitons aussi que la fonction de distance sémantique permette à un Agent de détecter que c_2 et c_3 ci-dessous décrivent essentiellement la même réalité, mais que c_2 décrit les Contextes associés à la lecture $(precision,\ unit)$ à la fois dans la source de la lecture et dans la valeur lue, alors que c_3 exprime ces idées de manière plus concise :

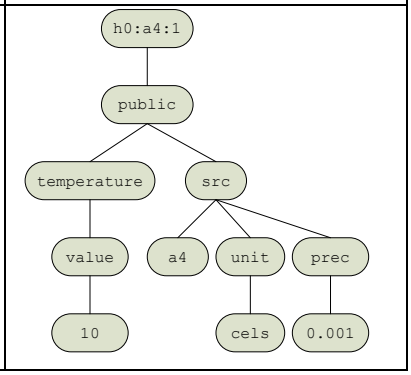
c_2

```
h0:a3:1 {
  public {
    temperature {
      value { 10 }
      unit  { celsius }
      precision { 0.001 }
    }
    source {
      a3
      unit  { celsius }
      precision { 0.001 }
    }
  }
}
```



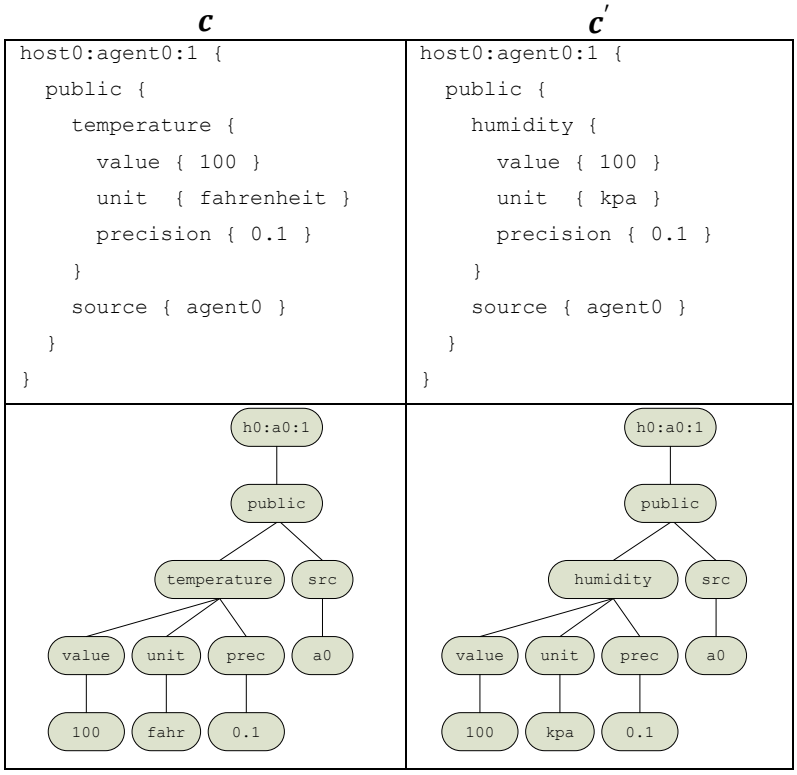
c_3

```
h0:a4:1 {
  public {
    temperature {
      value { 10 }
    }
    source {
      a4
      unit { celsius }
      precision { 0.001 }
    }
  }
}
```



Dans ce cas, $\sigma(c_2, c_3) > 0$, $\varrho(c_2, c_3) > 0$ et $\varrho(c_3, c_2) > 0$ mais nous sommes en droit de nous attendre à ce que $\varrho(c_3, c_2)$ soit relativement petite, la différence identifiée ne dépendant que du nom des Agents, alors que $\varrho(c_2, c_3)$ devrait être un peu plus élevée. Ici, un Agent consommateur qui ne serait pas préoccupé par les noms des Agents fournisseurs pourrait filtrer ceux-ci par des expressions de Contexte menant à des mises en correspondance.

Pour mieux déterminer ce que nous attendons de ρ, supposons qu'un Agent compare c et c' ci-dessous :



Ce cas montre deux Contextes pour lesquels la similitude structurelle est forte; cependant, nous souhaitons que notre fonction reflète le fait que c décrit une sorte de Contexte (la température ambiante) alors que c' décrit une toute autre sorte de Contexte (l'humidité ambiante).

4.6.7 Rapprocher les Contextes

L'évaluation de $\sigma(c, c')$ ou de $\varrho(c, c')$ avec $c, c' : C$ est faite à la demande d'un Agent a et est guidée par sa mission. Pour cette raison :

- la comparaison de Contextes survient habituellement lorsqu'un consommateur compare les Contextes produits par divers fournisseurs, ou encore lorsqu'un consommateur compare les Contextes produits par un même fournisseur à divers moments;
- comparer deux Contextes survient aussi dans un contexte de découverte, au sens où un Agent chercherait à obtenir ce qui, du Contexte disponible dans le voisinage, ressemble le plus (structurellement, en termes de contenu) à un modèle de Contexte correspondant à ce qu'il cherche à consommer.

Dans un cas comme dans l'autre, la forme par laquelle sont exprimés les Contextes évalués influence les résultats des distances calculées. Pour cette raison, adapter la forme et le contenu des requêtes de Contexte peut influencer les constats découlant des calculs réalisés par les fonctions σ et ϱ.

Notez que ContextAA offre les fonctionnalités pour exprimer les opérations de rapprochement de Contextes décrites dans les sections qui suivent, mais que leur application n'est pas automatisée et dépend présentement du cadre ontique (§4.8) des Agents.

4.6.7.1 Suppression de racine

Considérons par exemple ce qui suit :

c	c'

h0:a0:1 {
  public {
  temperature {
  value { 100 }
  unit { fahrenheit }
  precision { 0.1 }
  }
  source { a0 }
  }
}
	h1:a1:1 {
	  public {
	  temperature {
	  value { 100 }
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  source { a1 }
	  }
	}

Sur la base de c, c', nous avons :

$\varrho(c, c')$	$\varrho(c', c)$	$\sigma(c, c')$
0,395833	0,395833	0,240741

Dans ce cas, les différences de noms à la racine de c et de c' influencent le résultat. Si nous limitons notre examen aux valeurs de c et de c', ce qui est simple ici puisque $size\big(value(c)\big) = size\big(value(c')\big) = 1$, alors nous avons :

c	c'

public {
  temperature {
  value { 100 }
  unit { fahrenheit }
  precision { 0.1 }
  }
  source { a0 }
}
	public {
	  temperature {
	  value { 100 }
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  source { a1 }
	}

Avec cette nouvelle paire c, c', nous avons maintenant :

$\varrho(c, c')$	$\varrho(c', c)$	$\sigma(c, c')$
0,125	0,125	0,055556

Cette transformation structurelle, nommée suppression de racine, réduit de manière importante la distance structurelle et la distance de contenu entre les Contextes comparés.

4.6.7.2 Abstraction de racine

Une autre transformation simple applicable au Contexte est l'abstraction de racine, par laquelle la racine d'au moins un des deux Contextes est remplacée par une abstraction susceptible de subsumer la racine de l'autre Contexte, les rapprochant effectivement l'un de l'autre.

Une abstraction de racine possible pour $c = n\{v\}$ est sa transformation en $c' =* \{v\}$, de manière à provoquer une subsomption de la racine. Ceci a pour conséquence de faire en sorte que l'évaluation de distance entre c et c' repose sur v.

Notez que l'abstraction de racine ne modifie pas la structure du Contexte, alors que la suppression de racine a cet impact. Cette distinction influence les calculs de distance entre deux Contextes. Conséquemment, l'une ou l'autre de ces opérations peut être utile en fonction du cadre ontique de l'Agent qui y a recours.

Si n est composite et si sa structure importe, il est aussi possible de remplacer n par une abstraction composite telle que certains des éléments (ou tous les éléments) subsument ceux d'un autre Contexte.

L'exemple ci-dessous montre le résultat de deux abstractions de noms, soit une abstraction de racine et l'abstraction d'une feuille :

c	c'

h0:a0:1 {
  public {
  temperature {
  value { 100 }
  unit { fahrenheit }
  precision { 0.1 }
  }
  source { a0 }
  }
}
	?:?:? {
	  public {
	  temperature {
	  value { 100 }
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  source { * }
	  }
	}

Ici, la requête est exprimée par c et correspond à une demande pour du Contexte représentant une température exprimée dans un format particulier, sans égard à la provenance mais en demandant à ce que cette provenance soit tout de même indiquée selon un format particulier. Si c' est comparé avec c, alors nous avons :

$\varrho(c, c')$	$\varrho(c', c)$	$\sigma(c, c')$
0	0	0

Un exemple plus réaliste serait celui proposé ci-dessous :

c	c'	c''

?:?:? {
  public {
  temperature {
  value { * }
  unit { fahrenheit }
  precision { * }
  }
  }
}
	h0:a0:1 {
	  public {
	  temperature {
	  value { 100 }
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  source { a0 }
	  }
	}
		h0:a1:1 {
		  public {
		  temperature {
		  value { 37.78 }
		  unit { celsius }
		  precision { 0.1 }
		  }
		  source { a1 }
		  }
		}

Supposons ici que l'Agent consommateur souhaite comparer c à c' et à c''. La description que donne c du Contexte souhaité est donc un modèle sur la base duquel sont évalués des Contextes candidats. Nous avons dans ce cas :

$\varrho(c, c')$	$\varrho(c', c)$	$\sigma(c, c')$
0	0,125	0,0555556
$\varrho(c, c'')$	$\varrho(c'', c)$	$\sigma(c, c'')$
0,041667	0,145883	0,063786

Au sens de la requête décrite par c, les Contextes c' et c'' sont deux candidats valables mais sur la base des valeurs calculées avec ϱ et σ, du fait que $\varrho(c, c') < \varrho(c, c'') \wedge \sigma(c, c') < \sigma(c, c'')$, c' semble mieux correspondre aux exigences décrites par c que ne le fait c'' : la correspondance de contenu entre c et c' est complète alors qu'une légère distance structurelle entre les deux est relevée, différence due à la présence d'une description de la source de c' alors que c n'a pas décrit une requête pour un tel sous-Contexte.

Examinons un autre exemple :

c	c'	c''

?:?:? {
  public {
  temperature {
  value { * }
  unit { fahrenheit }
  precision { * }
  }
  }
}
	h0:a0:1 {
	  public {
	  temperature {
	  value { 100 }
	  }
	  source {
	  a0
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  }
	}
		h0:a1:1 {
		  public {
		  temperature {
		  value { 37.78 }
		  unit { celsius }
		  precision { 0.1 }
		  }
		  source { a1 }
		  }
		}

Cet exemple montre un cas où la requête décrite par c montre des différences structurelles plus prononcées avec c' qu'avec c'', bien que sur le plan du contenu la distance avec c' semble de prime abord moins grande qu'avec c'' du fait que c et c' utilisent la même unité de mesure pour décrire une température. Nous avons dans ce cas :

$\varrho(c, c')$	$\varrho(c', c)$	$\sigma(c, c')$
0,083333	0,083333	0,0802469
$\varrho(c, c'')$	$\varrho(c'', c)$	$\sigma(c, c'')$
0,041667	0,145883	0,063786

Déterminer lequel des candidats est le plus approprié est moins évident ici que dans les cas qui ont précédé : structurellement, les deux candidats sont proches de ce que requiert c, alors que c'' semble légèrement plus près de c que ne l'est c' sur le plan du contenu. Dans ce cas, le Contexte exprimé dans c mènerait probablement à un choix sous-optimal.

4.6.7.3 Permutation de Contexte

Parfois, la forme que prend un Contexte tel que publié se rapproche de celle du Contexte tel que demandé. Il est alors possible de réaliser une permutation de Contexte, soit déplacer un sous-Contexte d'une racine à l'autre, comme dans le cas suivant :

c	c'	c''

?:?:? {
  public {
  temperature {
  value { * }
  unit { fahrenheit }
  precision { * }
  }
  }
}
	h0:a0:1 {
	  public {
	  temperature {
	  value { 100 }
	  }
	  source {
	  a0
	  unit { fahrenheit }
	  precision { 0.1 }
	  }
	  }
	}
		h0:a1:1 {
		  public {
		  temperature {
		  value { 100 }
		  unit { fahrenheit }
		  precision { 0.1 }
		  }
		  source { a1 }
		  }
		}

Ici, c'' est le fruit de la permutation des sous-Contextes de c' nommés $unit$ et $precision$ de la racine $source$ vers la racine $temperature$. Ces permutations visent à réduire la distance structurelle entre la forme prise par le demandeur et celle exprimée dans le Contexte consommé.

Bien qu'il soit possible de réaliser toutes les permutations possibles d'un Contexte, il s'agit d'une opération de complexité élevée en pratique, ce qui explique que ContextAA ne la réalise pas de manière systématique. Le service demeure à la disposition des Agents, et ces derniers peuvent appliquer des algorithmes pour choisir des permutations appropriées à leurs besoins.

4.6.7.4 Rapprochement de Contextes par transformations

De manière générale, au moins trois possibilités s'offrent à l'Agent consommateur rencontrant des Contextes semblables mais suffisamment différents pour qu'il soit difficile de choisir le plus adéquat du lot. La première est d'abandonner, au sens où les différences entre c' et c'' du point de vue de la requête c sont trop faibles pour permettre de discriminer réellement en faveur ou en défaveur de l'un ou l'autre de ces candidats. Ici, « abandonner » signifie abandonner la tentative de choisir l'un ou l'autre, pas abandonner la mission : il est possible que les deux Contextes soient si similaires à c que l'un ou l'autre soit convenable pour le consommateur. Il est aussi possible que ni l'un, ni l'autre ne semble adéquat lorsque comparé à c, et qu'il soit préférable pour l'Agent consommateur de ne pas les utiliser. Le cadre ontique du consommateur est ce par quoi ce dernier fera un choix éclairé.

La deuxième est de transformer c' et c'' pour les rapprocher du modèle préconisé par c. Le défi est ici de ne pas dénaturer les candidats par ces transformations, donc de ne pas modifier le sens qui pourrait leur avoir été conféré au préalable. ContextAA offre des services pour permettre aux Agents de réaliser ces transformations, mais ne les automatise pas. Nous nommons déformation de Contexte les transformations successives appliquées à un Contexte d'une forme à l'autre.

Certaines transformations, par exemple abstraction de racine, sont d'abord des considérations subjectives de la part de l'Agent requérant, et comportent de faibles risques d'altérer la sémantique apparemment associée au candidat, alors que d'autres (permuter une valeur d'un Contexte à un Contexte voisin, par exemple) pourraient la modifier de façon plus important : par exemple, attribuer à un Contexte une valeur qu'il n'avait pas initialement ou une unité de mesure qui ne convient pas à ce qui a été mesuré.

Les transformations de Contexte dans le but d'obtenir un Contexte semblable, mais plus près d'être utilisable par un Agent a se font sur la base des paramètres suivants, qui font tous partie du cadre ontique de a :

- le seuil de tolérance de a envers la distance structurelle, $\mathrm{dist}_a^\sigma : \mathbb{Q} \in [0..1]$;
- le seuil de tolérance de a envers la distance de contenu, $\mathrm{dist}_a^\varrho : \mathbb{Q} \in [0..1]$;
- le seuil de tolérance de a aux déformations structurelles, $\mathrm{def}_a^\sigma : \mathbb{Q} \in [0..1]$; et
- le seuil de tolérance de a aux déformations de contenu, $\mathrm{def}_a^\varrho : \mathbb{Q} \in [0..1]$.

À titre d'exemple, supposons $c, c' : C(\sigma(c, c') > \mathrm{dist}_a^\sigma)$. L'Agent a pourra utiliser c'' si :

$$\exists \tau : \mathrm{T}, c'' : C \begin{pmatrix} Let\ c'' = \tau(c') \\ \big(\sigma(c, c') > \sigma(c, c'')\big) \wedge (\sigma(c', c'') \leq def_a^\sigma) \end{pmatrix}$$

La mécanique pour les déformations de contenu est analogue à celle présentée ici pour les déformations structurelles. Réduire la distance entre deux Contextes par déformation de l'un d'eux, dans le respect d'une tolérance aux déformations bien sûr, ne garantit pas que le Contexte résultant soit utilisable. En arriver à un Contexte utilisable par des déformations dépend du Contexte d'origine et des déformations choisies. Évidemment, le nombre de transformations à réaliser pour mener à bien une déformation de Contexte peut être élevé, du moins avec une approche naïve et qui reposerait sur une application de force brute.

La troisième option est de faire une évaluation pondérée comprenant à la fois la distance structurelle et la distance de contenu, de telle sorte que les pondérations relatives à chacune soient choisies par l'Agent requérant. Ceci nous amène à définir la distance sémantique.

4.6.8 Distance sémantique

Bien que l'état de nos travaux nous ait mené à déterminer les mesures de distance σ et ϱ décrites dans §§4.6.2-4.6.2.1, il n'est pas exclu que des travaux ultérieurs suggèrent des métriques supplémentaires. Pour cette raison, la discussion qui suit à propos de la distance sémantique traite à la fois du cas singulier auquel nous sommes présentement confrontés, et de l'éventualité d'une généralisation tenant compte de métriques supplémentaires.

Nous nommons distance sémantique, notée ρ, un cumul pondéré des diverses distances entre $c, c' : C$. Avec une pondération par défaut de $\frac{1}{2}$ pour σ comme pour ϱ, la distance sémantique s'exprime comme suit :

$$\rho :: C \to C \to \mathbb{Q} \in [0..1]$$

$$\rho(c, c') \triangleq \frac{\varrho(c, c') + \sigma(c, c')}{2}$$

Généralisant cette fonction pour couvrir un ensemble $f : F \left(F :: C \to C \to \mathbb{Q} \in [0..1]\right)$ de fonctions évaluant la distance entre deux Contextes, et supposant une pondération uniforme pour chacune d'elles, la distance sémantique s'exprimerait :

$$\rho(c, c') \triangleq \frac{\sum_{f \in F} f(c, c')}{|F|}$$

Enfin, en allouant des pondérations distinctes $\forall f \in F$ et en notant ω_i la pondération de $f_i, 0 \leq i < |F|$, dans la mesure où $\forall \omega_i : 0 \leq i < |F| : 0 \leq \omega_i \leq 1$ et dans la mesure où $\sum_{i=0}^{|F|-1}(\omega_i) = 1$, nous obtenons :

$$\rho(c, c') \triangleq \sum_{i=0}^{|F|-1} (f_i(c, c') \times \omega_i)$$

Pondérer les mesures de distance permet à un Agent d'insister sur l'importance relative pour lui d'une mesure en comparaison avec une autre. Ceci permet entre autres à un Agent de ne pas tenir compte d'une métrique qui ne conviendrait pas à ses besoins.

4.6.9 Interpréter la distance sémantique

Le rôle de ρ dans ContextAA se situe dans l'action d'un Agent a sur un Hôte h et dans son cycle d'opération. Puisqu'une requête pour du Contexte est contextifiable, une requête produite par a est en fait un Contexte q produit par a et entreposé dans S_a; q réside dans S_a jusqu'à ce que l'Agent standard chargé de résoudre les requêtes ne l'y cueille. Une fois cueilli, q est résolu, dans S_a si q est privé; dans S_h s'il est local; si q est public, alors il sera disponible pour $\forall h' \in N_h$.

La résolution de q est toujours différée, ne serait-ce que d'une itération dans les cas privé et local, du fait qu'elle s'opère en arrière-plan, à travers l'action d'un Agent standard spécialisé à cette fin.

La résolution de q génère une séquence $r: List\langle C \rangle$ de Contextes candidats, dont les éléments sont placés dans S_a pour que a y ait accès. Il est possible que a ait besoin de tous les Contextes candidats, par exemple dans le cas où a ferait l'inventaire des actuateurs avoisinants, tout comme il est possible que a détermine le meilleur candidat du lot pour lui (p. ex. : le fournisseur de Contexte sur la luminosité ambiante se situant le plus près de lui dans la même pièce, ce qui peut faire l'objet d'une requête à part entière ou se limiter à un $eval$ sur le contenu de S_a).

La mobilité des Agents et des humains, combinée à notre choix de préférer un cadre ontique selon l'Agent à ontologie commune, fait en sorte que chercher à découvrir dynamiquement le Contexte candidat le plus proche de ce que recherche a accroît les probabilités que a puisse assurer la continuité de son service et poursuivre sa mission.

Identifier le Contexte candidat le plus proche sémantiquement d'un modèle de Contexte donné, à partir de métriques comme la σ et ϱ, permet à a de déterminer s'il semble probable qu'un candidat qui ne serait pas exactement de la forme requise puisse être transformé de manière à s'en rapprocher.

4.6.10 Méta-Contextes

Certains Contextes ont pour principal rôle de décrire des règles d'entretien de Contexte; en particulier, les règles décrivant ce qui fait qu'un Contexte logé dans un espace contextuel (§3.4.4) demeure pertinent ou non alors que le temps passe sont décrites sous cette forme, et sont donc des règles subjectives.

Sur la base de ces méta-Contextes, un Agent donné peut décrire les règles qui lui permettent d'oublier les Contextes qui, parmi ceux qu'il a accumulés, sont devenus périmés. Ceci contribue à contrôler la croissance de consommation de ressources par Agent sur un Hôte donné.

La mécanique soutenue par ContextAA en lien avec les méta-Contextes va comme suit :

- une opération de gestion de Contexte est étiquetée comme étant un méta-Contexte;
- les requêtes provenant d'Agents du domaine escamotent les méta-Contextes. Ceci évite par exemple le cas où un Agent, en demandant un modèle de Contexte donné obtiendrait non seulement les Contextes correspondants, mais aussi les méta-Contextes demandant leur élimination de l'espace contextuel;
- un Agent standard spécialisé consomme les méta-Contextes et met en application les règles que ceux-ci décrivent. Ceci permet entre autres d'encadrer la croissance de l'espace requis pour entreposer l'espace contextuel S_h d'un Hôte h.

L'étiquetage d'un Contexte en tant que $n$ième méta-Contexte pour un Agent donné se fait par l'injection du nom de Contexte $meta: n$ à sa racine.

4.7 Agents

Dans ContextAA, un Agent est une entité dépendante du Contexte [71] qui s'inscrit dans un cycle d'opérations guidé par un Hôte et son gestionnaire d'Agents (§5.2). À titre de rappel, un Agent est une entité active prise en charge par un Hôte, et qui cherche à accomplir une missionen appliquant une perspective subjective sur le Contexte, perspective décrite par son cadre ontique.

Les Agents font partie de l'une de deux familles, soit les Agents standards et les Agents spécifiques au domaine d'application, que nous nommons simplement Agents du domaine. Un Agent standard offre des services pour son Hôte et est typiquement conçu pour une plateforme spécifique. Un Agent du domaine est axé vers la réalisation d'une mission et est typiquement contextifié, ce qui facilite sa migration d'un Hôte à l'autre.

4.7.1 Modèle général (survol)

Dans ContextAA, un Agent peut être représenté comme un 4-uplet de la forme $a: A; a = \{M, O, R, S\}$ où :

- la partie M représente sa mission, ce qu'il doit accomplir;
- la partie O représente ce qui tient lieu pour lui de perspective sur le Contexte, ce que nous nommons son cadre ontique;
- la partie R représente les restrictions de l'Agent, au sens de ce dont il a besoin au minimum pour fonctionner en termes de ressources. Ce Contexte permet d'éliminer les Hôtes potentiels pour l'Agent sur la base du non-respect des règles qu'il décrit, et peut être vide dans le cas où l'Agent n'a pas de restrictions particulières; et
- la partie S représente son espace contextuel, le Contexte à sa disposition. Techniquement, $\{M \cup O \cup R\} \in S$ pour un Agent donné, mais le découpage en un 4-uplet facilite la compréhension de la modélisation d'un Agent.

Pour clarifier le propos, nous écrirons parfois $a = \{M_a, O_a, R_a, S_a\}$ pour expliciter la relation des éléments à un Agent a donné. Informellement, M_a décrit l'état attendu suivant l'action de l'Agent a (ce qui doit s'avérer du point de vue de a); R_a décrit ce qui doit être offert par un Hôte h pour que a demeure sur h, sans quoi a devra chercher un nouvel Hôte respectant les exigences exprimées par R_a; enfin, O_a liste les requêtes pour du Contexte que a doit émettre dans le cadre de ses fonctions.

La forme générale d'un Agent du domaine de nom id est décrite par la figure 10 (page suivante).

La distinction entre « consomme » et « agit » dans la figure 10 sert à faciliter le travail des individus qui rédigent des Agents; la compartimentalisation entre « consommer », « agir » et « produire » correspond aux étapes du cycle de vie des Agents, sur lequel nous reviendrons, et l'intention est que « consommer » corresponde à ce qui doit être fait lorsque l'Agent a constate le contenu de S_a, « agir » corresponde à ce que a fait sur la base de ce Contexte, et « produire » corresponde à ce qui doit être inséré dans S_a suite à ces actions.

```
id {
  "human readable name" {
    // nom pour faciliter le référencement par des humains (utilisé à fins documentaires)
  }
  "ontic frame" {
    wants {
      // Contexte décrivant les actions à prendre lorsque l'Agent consomme
      // du Contexte
    }
    does {
      // Contexte décrivant les actions à prendre lorsque l'Agent agit sur
      // du Contexte
    }
    produces {
      // Contexte décrivant les actions à prendre lorsque l'Agent produit
      // du Contexte
    }
  }
  mission {
      // Contexte décrivant l'état que l'Agent souhaite constater suite à ses actions
      // (destiné à l'Hôte)
  }
  restriction {
      // Contexte décrivant les conditions minimales requises pour assurer le bon
      // fonctionnement de l'Agent (destiné à l'Hôte)
  }
  space {
    // Contexte contenant l'ensemble des Contextes connus de l'Agent
  }
}
```

Figure 10 - Agent du domaine (forme générale)

Supposant la fonction suivante :

$$queryFor :: C \to Q \to C$$

ayant pour rôle de prendre un modèle de Contexte et de produire en retour une requête pour ce Contexte, de même que la fonction suivante :

$$qualifOf :: C \to Q$$

exprimant la qualification d'accès associée à un Contexte donné, alors un Agent du domaine a a le comportement suivant (exprimé en pseudocode) :

Étape	Pseudocode
Consommation	$reqs = \bigcup_{\forall c:C(c \in wants)} queryFor(c, qualifOf(c))$
Action	$acts = \bigcup_{\forall c:C(c \in does)} queryFor(c, private)$ $workRes = \bigcup_{\forall c:C(c \in acts)} apply(c, S_a)$
Production	$yield(reqs \cup workRes \cup produces)$

Sur la base de cette description succincte de la mécanique associée à l'action d'un Agent, il est utile de noter ce qui suit :

- seule l'étape de production mène à l'ajout de nouveaux Contextes, et cet ajout se fait dans S_a. Ces Contextes incluent le fruit du travail accompli à l'étape d'action, l'ensemble des requêtes construites à l'étape Consommer, et le Contexte que a devait produire *a priori*;
- conséquemment, un Agent est muni d'états (*Stateful*) pour tenir compte à l'étape de production des ensembles de Contextes générés par l'action des étapes de consommation et d'action;
- les actions prises par a se font à partir du Contexte logé dans S_a;
- le fruit des actions prises lorsque a agit peut être destiné à d'autres Agents que lui, et il en va de même pour les requêtes décrites à l'étape de consommation;
- les modèles de Contexte générés à l'étape de consommation sont susceptibles d'être porteurs de leur propre qualification d'accès, et il en va de même pour les Contextes décrits à l'étape de production;
- les étapes ci-dessus génèrent du Contexte, ce qui signifie qu'il faut à ContextAA des mécanismes de contrôle de la croissance de l'espace occupé par S_a. Ce mécanisme en est un d'oubli sélectif de Contexte (§4.7.5.2), et passe par la publication de méta-Contextes (§4.6.10).

Le Contexte placé dans S_a est pris en charge par des Agents standards qui assurent le traitement des demandes et la mise à jour de S_a suivant ce traitement. Même l'étape de production n'accède pas directement à S_a : les Contextes retournés à cet étape sont pris en charge par un Agent standard qui assure la mise à jour de S_a pour a.

4.7.2 Agent du domaine

Un Agent du domaine $a \in$ A suit typiquement la forme décrite dans la section §4.7.1 plus haut. Un tel Agent peut être représenté comme un n-uplet :

$$\mathrm{A} = \{M, O, R, S\}$$

dont tous les éléments sont contextifiables : ils pourraient tous se loger dans S, mais sont considérés séparément ici pour clarifier la discussion. Pour un Agent a: A tel que $a = \{M_a, O_a, R_a, S_a\}$:

- la mission M_a représente l'ensemble des tâches que a doit accomplir. Assurer la continuité de service de a signifie minimiser les moments où a n'est pas en mesure de faire ce que décrit M_a;

- le cadre ontique O_a représente le filtre par lequel a évalue le Contexte : comment a compare deux Contextes, par exemple : sur quelle base a, placé devant $c, c' \in C; c \neq c'$, choisira l'un, l'autre, les deux ou aucun des deux. C'est par O_a que a parvient à réaliser une évaluation subjective de $\forall c: C$, ce qui contribue à le rapprocher de son objectif de continuité de service;
- les contraintes R_a représentent ce dont dépend a pour accomplir M_a. À titre d'exemple, si $atteinte(a, M_a)$ telle que décrite dans nécessite que a ait accès à un appareil particulier (GPS, accéléromètre, capteur d'humidité, etc.), alors a ne pourra être déployé sur $h: H$ que si h possède cet appareil; et
- l'espace contextuel S_a représente ce que connaît a. Tout Agent a écrit dans S_a et lit indirectement (par la médiation d'Agents standards) de S_a tout Contexte devant persister entre deux moments consécutifs d'action de a. Le contenu de S_a inclut entre autres les requêtes soumises par a et les réponses obtenues pour a.

Concrètement, un Agent du domaine a accès, grâce aux services offerts par les Agents standards, à la fonction suivante :

$$descr :: H \to C$$

$$descr(h) \triangleq S_h$$

telle que $descr(h)$ exprime sous forme de Contexte les caractéristiques de $h: H$. Au sens de l'appelant de $descr(h)$, une caractéristique qui n'est pas décrite par le Contexte généré équivaut à une ressource non-supportée par h. Ainsi, soit la fonction suivante :

$$support :: C \to H \to [0..1]$$

$$support(\emptyset, h) \triangleq 1$$

$$support(r: C, h) \triangleq 1 - \left(\frac{\sum_{c \in r} \varrho\big(c, descr(h)\big)}{|value(r)|}\right)$$

quantifie le support des contraintes décrites par r sur l'Hôte h, avec $support(r, h) = 1$ seulement si toutes les contraintes exprimées par r sont respectées sur h, ce qui inclut bien sûr le cas où a n'a aucune contrainte particulière, donc le cas où $R_a = \emptyset$.

La tolérance ou non envers un support partiel des contraintes exprimées par R_a, varie selon les Agents. Dans le cas où $\exists h, h': H\big(support(r, h) = support(r, h')\big)$, a peut offrir un critère d'ordonnancement permettant de privilégier l'un ou l'autre de ces Hôtes; sans un tel prédicat, la présomption sous ContextAA est que h et h' sont aussi valables l'un que l'autre en tant que destination pour la migration de a et la mécanique de migration choisira l'un ou l'autre selon des critères qui lui appartiennent.

Une utilité directe de $support(r, h)$ est de permettre l'automatisation de la construction d'un ensemble $\Xi_a \subseteq N_h$ tel que $h' \in \Xi_a$ signifie que h' est un candidat pour une éventuelle migration $h \overset{a}{\to} h'$.

4.7.3 Atteinte d'une mission

Tout Agent a est chargé d'une mission M_a où $M_a : C$. Un Agent a exprime M_a par le Contexte décrivant ce qui doit être constaté par a pour que M_a soit un succès. La complexité des missions varie selon les Agents. À chaque fois que a agit, cette action se fait avec pour objectif l'atteinte de M_a; conséquemment, le service qu'offre a lorsque a agit à un instant t est assuré si a atteint M_a.

Bien que $M_a \in C$, nous pouvons à titre illustratif noter $atteint(a, M_a)$ la mesure de l'atteinte par a de M_a, qualifiant au besoin cette notation de l'indice t pour spécifier que le prédicat s'avère à l'instant t. .

L'évaluation de $atteint(a, M_a)$ se formalise comme suit. Supposons que $résultat(a, t)$ soit un Contexte décrivant, sur la base de ce qu'exprime O_a, le fruit des actions de a à l'instant t. Alors, nous avons :

$$résultat :: A \to \mathbb{Z} \to C$$

$$atteint :: \mathrm{A} \to M \to \mathbb{Q} \in [0..1]$$

$$atteint(a, M_a)_t \triangleq 1 - \rho\big(M_a, résultat(a, t)\big)$$

Bien que présenté sous forme d'une fonction ici, $résultat(a, t)$ peut en pratique n'être qu'un état décrit sous forme de Contexte logé dans S_a et représentant le Contexte pertinent selon a. La valeur de $atteint(a, M_a)$, sur cette base, décrit le degré de succès de a dans la poursuite de sa mission. L'évaluation de $\forall a : \mathrm{A}\big(atteint(a, M_a)\big)$ peut ainsi être prise en charge formellement par son Hôte.

4.7.4 Continuité de service

Assurer la continuité de service est l'un des principaux objectifs d'un Agent. Nous définissons la continuité de service ψ comme suit :

$$\text{Soit } i, j \in \mathbb{Z}, i < j, t_i < t_j$$

$$\psi :: \mathrm{A} \to \mathbb{Z} \to \mathbb{Z} \to [0..1]$$

$$\psi\big(a, t_i, t_j\big) \triangleq \frac{\sum_{t=t_i}^{t_j} atteint(a, M_a)_t}{j - i}$$

Le dénominateur $j - i$ décrit l'intervalle de temps (d'étapes, en fait) pour lequel la continuité de service est évaluée.

Assurer la continuité de service pour a dans l'intervalle de temps défini par t_i et t_j signifie optimiser $\psi\big(a, t_i, t_j\big)$, donc faire en sorte que $\psi\big(a, t_i, t_j\big) \leq \varepsilon$ pour ε aussi petit que possible. En pratique, cette maximisation est assujettie au contexte dans lequel évolue a et aux impondérables contextuels de son environnement.

4.7.5 Espace contextuel

Tout Agent a possède un espace contextuel S_a où sont entreposés les Contextes qu'il connaît. Le Contexte dans S_a fait partie du Contexte-selon a, bien que cette dénomination englobe non seulement le Contexte connu de a, mais aussi ce que a accepte en tant que Contexte sur le plan sémantique, donc ce qui répond aux requêtes qu'il a émises et qu'il admet comme tel de par son cadre ontique.

Sur un Hôte h donné, si $a \in h$ est un Agent du domaine, alors le seul « lien » qu'entretient a avec h est S_a; si a est un Agent standard, alors a interagit à la fois avec S_a, S_h, de même que (potentiellement) avec la plateforme sous-jacente et des canaux de Contexte.

Un espace contextuel dans ContextAA peut être vu comme un dépôt de Contexte exprimé sous forme de Contexte. S_a peut donc être décrit comme un Contexte dont les valeurs sont les Contextes racines connus de a. Sur un Hôte h, $S_h \supset \bigcup_{a \in h} S_a$.

4.7.5.1 Contenu et gestion d'un espace contextuel

Lors de l'exécution d'une itération du cycle normal d'opérations d'un Agent $a \in h$, aucun autre Agent $a' \in h (a' \neq a)$ ne peut intervenir sur l'état de S_h. Tel que mentionné précédemment, si a est un Agent du domaine, alors a ne peut consulter que ce qui se loge dans S_a, tout en n'ayant accès à aucun service permettant de modifier directement S_a. La médiation entre S_a et tout autre espace contextuel se fait par le biais d'Agents standards, et il en va de même pour les écritures dans S_a.

Un espace contextuel évolue avec le passage du temps. Ainsi, pour un Agent $a \in h$ donné, S_a contient à l'instant t :

- les connaissances acquises par a, représentées en tant que Contexte. Ceci inclut le résultat de calculs ou de raisonnements antérieurs, conservés dans S_a pour usage futur, de même que l'état du monde selon a, ou du moins le sous-ensemble pertinent à son domaine d'intérêt et à l'atteinte de sa mission;
- en pratique, pendant que a est pris en charge dans le cycle normal d'opérations de h, S_a contiendra aussi à la fois M_a, O_a et R_a. Sur le plan formel, nous distinguons toutefois ces éléments de S_a pour faciliter la discussion;
- les requêtes publiées par a, exprimées sous forme de modèles de Contexte. Tout Agent peut publier du Contexte, incluant des modèles de Contexte, susceptibles d'être consommés par des Agents standards et transmis vers d'autres Agents sur le même Hôte ou sur d'autres Hôtes pour mener à une éventuelle mise en correspondance avec le Contexte publié à cet endroit. Puisque les Agents sont pris en charge séquentiellement sur un Hôte donné, les requêtes publiées par a sont prises en charge de manière asynchrone, du moins pour qui prend la perspective de a; et
- les méta-Contextes exprimant les règles pour assurer l'entretien de S_a.

4.7.5.2 Oublier le Contexte

La gestion de S_a est une chose importante : par exemple, certains Contextes deviennent périmés avec le temps, remplacés par des Contexte plus récents ou plus proches des besoins de a; de même, certains Contextes sont de nature éphémère et doivent être éliminés une fois utilisés. Ne pas élaguer S_a de ses Contextes périmés entraînerait une consommation sans cesse croissante de l'espace requis pour chaque espace contextuel.

Tout comme la pertinence d'un Contexte pour a dépend de a lui-même, il en va de même pour les règles par lesquelles un Contexte sera privilégié à un autre ou sera jugé désuet, ce qui explique que les méta-Contextes régulant l'oubli de Contextes selon a sont décrits dans O_a. Ceci n'exclut pas la présence de méta-Contextes régulant l'oubli pour un Hôte entier, en particulier lorsque ces règles s'appliquent à des Contextes utiles à l'ensemble des Agents.

La capacité qu'a un Agent de réguler lui-même la pérennité ou l'obsolescence de Contextes dans son espace contextuel lui donne un contrôle fin sur l'état de ce qu'il choisit de placer à cet endroit.

4.7.6 Agent standard

Un Agent standard implémente des services pour l'Hôte auquel il participe. C'est par les Agents standards qu'un Hôte offre des services analogues à ceux d'un intergiciel, et assure la présence d'une interface sous forme de Contexte entre les Agents du domaine, les objets autonomes qui gravitent autour d'un Hôte, et la plateforme sous-jacente. Une liste des Agents standards déjà implémentés dans ContextAA est donnée à l'annexe F, mais il est probable que cette liste s'enrichisse avec le temps.

Chaque Agent standard a une responsabilité. Les responsabilités des Agents standards sur $h: H$ varient selon l'Agent, mais se divisent en quatre catégories :

- interactions avec le matériel;
- interactions avec les objets autonomes, et à travers ceux-ci, les interactions avec le voisinage;
- gestion et entretien du Contexte dans S_h, incluant par conséquent $S_a: a \in h$; et
- services essentiels généraux.

Dans chaque cas, un Agent standard joue un rôle analogue à celui d'un intergiciel, assurant la présence de services sur lesquels peuvent compter les autres Agents, en particulier les Agents du domaine. Plus précisément :

- dans le cas des interactions avec le matériel, le rôle d'un Agent standard est d'exprimer sous forme de Contexte une description de la plateforme sous-jacente, et de consommer sous forme de Contexte des demandes de mutation d'états de la plateforme sous-jacente;
- dans le cas des interactions avec le voisinage, le rôle d'un Agent standard est d'interagir avec des objets autonomes, en périphérie d'un Hôte, pour découpler les entrées/ sorties avec le voisinage réseau du cycle normal d'opération de cet Hôte;
- dans le cas de la gestion et de l'entretien du Contexte, le rôle d'un Agent standard est de réaliser les opérations qui ne sont pas à la portée des Agents du domaine.

Dans ContextAA, un Agent standard est un Agent, au sens où il s'inscrit dans le cycle de vie d'un Agent (§5.2.5) et dans son cycle normal d'opérations (§5.3) et au sens où il expose la même interface que celle exposée par les Agents du domaine (§**Erreur ! Source du renvoi introuvable.**). T outefois, un Agent standard a plusieurs particularités qui le distinguent des Agents du domaine sur le plan technique :

- un Agent standard est ainsi nommé du fait qu'il est disponible sur tous les Hôtes, et toujours sous la même dénomination. Ainsi, le nom d'un Agent gestionnaire de Contexte sera le même, sans égard à l'Hôte sur lequel il loge;
- un Agent standard représente un service, ou une gamme de services, pour un Hôte donné;

- un Agent standard est implémenté spécifiquement pour un Hôte donné. De même, il est lié à cet Hôte et n'en migrera pas.

Les Agents standards occupent une niche essentielle dans l'écosystème mis en place par ContextAA. Ce sont les entités qui assurent l'offre de services d'un Hôte.

4.7.6.1 Interactions avec le matériel

Un Agent du domaine a est fait de Contexte, et raisonne sur la base du Contexte mis à sa disposition dans S_a. Pour cette raison, son Hôte h représente les états du nœud sous-jacent sous forme de Contexte dans S_h. La tâche de représenter ces états sous forme de Contexte repose est dévolue aux Agents standards.

Chaque Agent standard responsable d'interagir avec le matériel consomme des données de la plateforme sous-jacente et les représente sous forme de Contexte, et peut assurer le chemin inverse à la manière d'un actuateur. Les fonctions de la plateforme sous-jacente varient selon les plateformes; un Agent standard doit interagir avec ces fonctions pour accomplir sa mission, ce qui explique qu'il soit typiquement rédigé non pas sous forme de Contexte mais bien dans un langage de programmation conventionnel selon des modalités tenant compte de son contexte applicatif.

Parmi les tâches d'un Agent standard, on trouve :

- exprimer sous forme de Contexte les ressources disponibles (niveau d'énergie restant dans la pile, mémoire disponible, espace disque disponible, etc.);
- exprimer sous forme de Contexte la présence d'appareils sur la plateforme, tels qu'un accéléromètre, un GPS ou un capteur particulier;
- exprimer sous forme de Contexte les états rapportés par les appareils disponibles sur la plateforme;
- exprimer sous forme de Contexte les interfaces permettant d'appeler un service offert de façon programmatique sur la plateforme sous-jacente; etc.
- recevoir des demandes de modification d'état, et de réaliser ces modifications d'état au nom de l'Agent demandeur, ce qui est particulièrement pertinent dans le cas de nœuds munis d'actuateurs. C'est par exemple à l'aide d'un Agent standard que serait mis en place un mécanisme de fermeture d'urgence d'un appareil.

4.7.6.2 Interactions avec les objets autonomes et le voisinage

Puisque $\forall a \in h$, a s'inscrit dans un cycle de vie, certaines opérations ne peuvent être faites par a, qu'ils soit standard ou du domaine, car elles pourraient nuire au bon fonctionnement de h et, plus spécifiquement, de $\forall a' \in h: a' \neq a$. Ces opérations incluent toute entrée/sortie bloquante et tout traitement synchrone (au sens faible du terme) susceptible de se prolonger. Les tâches qui ne sont pas susceptibles de s'inscrire dans le cycle de vie d'un Agent sont dévolues à des objets autonomes.

Tenir à jour la représentation contextualisée N_h du voisinage réseau d'un Hôte h fait partie de ces tâches. En effet, N_h est susceptible de varier au fil du temps, pour plusieurs raisons :

- la potentielle mobilité des individus, considérée comme un *a priori* de ContextAA;
- la mobilité des Agents eux-mêmes; et
- la présomption de disponibilité imparfaite des Hôtes, susceptibles d'être placés sur des nœuds dont les ressources peuvent varier avec le temps ou de subir pannes, défectuosités matérielles, défectuosités logicielles et autres interruptions de service.

Comme mentionné précédemment, l'approche soutenue par ContextAA est de proposer un modèle de programmation par lequel un Agent du domaine $a \in h$ raisonne sur le présent-selon-lui, donc sur l'état courant de S_a (le Contexte-selon). L'Agent a est isolé du réseau, et interagit strictement avec S_a; la médiation entre h et N_h passe par des objets autonomes.

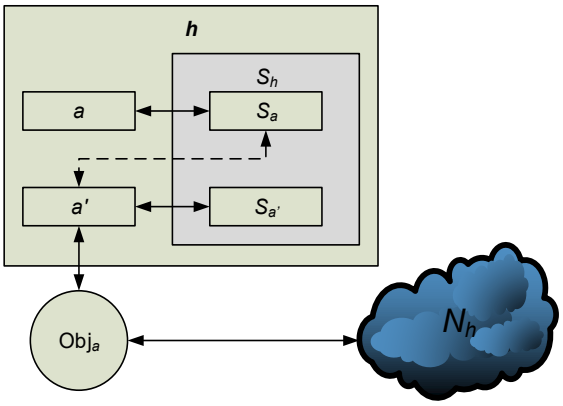


Figure 11 - Chemin de S_a à N_h

Dans la figure 11 ci-dessus, a est un Agent du domaine et n'interagit donc qu'avec S_a. L'Agent a' est quant à lui un Agent standard : en tant qu'Agent, il interagit avec $S_{a'}$ mais en tant qu'Agent standard, il a aussi le privilège d'interagir avec S_h tout entier. Pour cette raison, il est possible pour a' de consommer des requêtes dans S_a pour tenter de les résoudre, tout comme il est possible pour a' d'écrire dans S_a le résultat de la prise en charge de requêtes publiées précédemment par a.

Seuls les Agents standards transigent directement avec les objets autonomes. L'interaction entre un Agent standard et un objet autonome passe par un canal de contexte (§3.5.6), file d'attente bidirectionnelle synchronisée. Prenant pour exemple le cas de N_h, cette approche résulte en une couche isolante faite de Contexte entre $a \in h$ et N_h, couche par laquelle a peut continuer d'opérer même en totale coupure avec N_h ou lorsque $N_h = \emptyset$.

4.7.6.3 Gestion et entretien du Contexte

Un Agent standard, ayant accès au Contexte entier de son Hôte et ayant pour rôle d'interagir avec ce Contexte, complète les Agents du domaine et leur Hôte de plusieurs manières :

- prise en charge de la mise en correspondance des requêtes pour du Contexte;
- prise en charge des demandes soumises par des Agents sur d'autres Hôtes;
- prise en charge de l'entretien des espaces contextuels, sur la base de règles exprimées par chaque Agent sous forme de Contexte;
- etc.

À cette fin, ContextAA comprend des Agents standards tels qu'un Agent solutionneur, qui consomme les requêtes pour du Contexte logées dans les espaces contextuels des autres Agents sur le même Hôte; un Agent intermédiaire, qui prend à son compte les requêtes reçues d'autres Hôtes pour que l'Agent solutionneur n'ait pas à tenir compte de cette particularité; un Agent gestionnaire de Contexte, qui a pour objectif de consommer les règles de gestion du Contexte exprimées sous forme de méta-Contexte, de réaliser les tâches que ces règles décrivent, puis d'éliminer les méta-Contextes en question; etc.

4.7.6.4 Services essentiels généraux

Par services essentiels généraux, nous entendons des services tels qu'un mécanisme pour avertir l'usager de manière immédiate, par exemple dans le cas où son intégrité physique pourrait être compromise.

Un exemple concret est un Agent avertisseur, Agent standard qui ne fait que consommer les Contextes représentant des avertissements destinés à l'usager et les présente immédiatement à ce dernier, du moins dans la mesure du possible, d'une manière appropriée en fonction du nœud sur lequel son Hôte s'exécute.

4.7.6.5 Agents de soutien

Bien que les Agents standards aient pour la plupart la particularité d'être disponibles sur tous les Hôtes et de modéliser des services pour ces Hôtes, il existe une catégorie spécialisée de ces Agents que nous nommons Agents de soutien, et qui ont pour rôle d'interfacer avec un composant matériel spécifique.

La seule réelle distinction entre les Agents de soutien et les Agents standards typiques est que la présence ou l'absence d'un Agent de soutien particulier dépend du nœud sur lequel un Hôte est déployé. Pour cette raison, alors qu'il est habituellement raisonnable d'accéder au Contexte d'un Agent standard par son nom, il tend à être préférable de communiquer avec un Agent de soutien sur la base des modèles de Contexte qu'il est susceptible de générer.

Les Agents de soutien font partie de la catégorie des Agents standards au sens où ils sont attachés à un nœud particulier, sont susceptibles d'être codés « en dur » pour profiter au maximum du matériel sous-jacent, et qu'il est possible qu'ils interfacent avec un objet autonome dans le cadre de leurs fonctions, comme dans le cas où un Agent de soutien a besoin qu'une lecture bloquante sur un appareil soit réalisée.

4.8 Cadre ontique

Tel que mentionné précédemment, ContextAA évite la question d'une ontologie partagée par les Agents et utilise plutôt ce que nous nommons un cadre ontique, référentiel sémantique local à l'Agent et placé au cœur du Contexte-selon. Ceci s'inscrit dans la philosophie de ContextAA qui vise à assurer l'autonomie des Agents: en faisant de chaque Agent le porteur de son propre référentiel, l'Agent devient capable de traiter le Contexte indépendamment des autres Agents, ce qui facilite la poursuite de sa mission et sa continuité de service.

Le cadre ontique décrit ici se rapproche de définitions formelles d'ontologie comme celle de [56], qui la décrit comme une paire $\langle S, A \rangle$ où S est un vocabulaire commun et A des axiomes pour raisonner sur ce vocabulaire, ou de [14] qui précise le volet S et décrit l'ontologie comme un 4-uplet $\langle C, R, I, A \rangle$ où en plus du volet axiomes décrit par A, C est un ensemble de concepts, R un ensemble de relations et I un ensemble d'instances des concepts décrits dans C. Toutefois, le cadre ontique comme il est défini dans ContextAA est pensé pour se décrire sous forme de Contexte et s'intégrer harmonieusement dans l'ensemble de l'architecture.

4.8.1 Formalisme

Dans la perspective soutenue avec ContextAA, le cadre ontique O_a d'un Agent a décrit les objets d'intérêt pour a et les relations entre ces objets selon la perspective opératoire de a. Par perspective opératoire, nous entendons la description sous forme de Contexte des requêtes et opérations permettant à a d'accomplir sa mission. Ainsi :

- les requêtes pour du Contexte décrivent ce que a requiert pour raisonner sur le Contexte;
- les opérations décrivent ce raisonnement;
- parmi ces opérations, celles décrivant ce que publie a de manière publique constituent la contribution de $a \in h$ à l'évolution du Contexte disponible pour l'ensemble des Agents de N_h.

Une conséquence du recours à un cadre ontique selon l'Agent est qu'une chose en apparence aussi fondamentale que ce qu'est un contexte, pris en tant qu'élément du domaine d'intérêt, dépend de l'Agent. Le Contexte publié par $a \in h$ peut ne pas correspondre, en forme ou en contenu, au Contexte requis par $a' \in h'$ ou même par $a' \in h$ tel que décrit par $O_{a'}$. Il est d'ailleurs fondamental au design de ContextAA, que l'expression sémantique du Contexte soit en fait une construction syntaxique, traitée comme telle selon le cadre ontique de chacun des Agents [2].

Même les règles permettant de valider la qualité d'un Contexte donné sont locales à l'Agent a et décrites dans O_a. Parmi les règles décrites dans O_a se trouvent les seuils de tolérance envers les distances entre Contextes, de même que les seuils de tolérance envers les déformations apportées aux Contextes par voie de transformations.

4.8.2 Adéquation et vérité

En tout temps, S_a au moment t_j est construit à partir des contraintes de O_a et de $\forall c: C$ tel que :

- c est produit par a' et
- $a' \in h', h' = h \vee h' \in N_h$ à un moment $t_i: i < j$.

Le fait que $c \in S_a$ n'indique rien sur la validité de c, cependant, que ce soit de manière intemporelle (c a-t-il été produit par un Agent défectueux?) ou ponctuelle (c était peut-être valide au moment t_i, mais l'est-il encore au moment t_j?). Si $c \in S_a$, toutefois, alors il en découle que c correspond aux règles décrites par O_a. En ce sens, O_a joue entre autres rôles celui de filtre pour insertion d'un Contexte dans S_a.

Dans tous les cas, la conformité de $c: C$ aux règles de O_a est le critère clé pour accepter c dans S_a; la véracité de c ne peut pas nécessairement être vérifiée et ne peut par conséquent constituer un critère d'acceptation ferme. Dans ContextAA, un Contexte c n'est pas intrinsèquement utile ou pertinent; la question de l'utilité d'un Contexte c pour un Agent a tient non pas à sa véracité ou à sa validité, mais bien à sa conformité avec les critères du cadre ontique O_a.

D'office, si a, a': A; $a \neq a'$ alors il est possible que $O_a \neq O_{a'}$ donc qu'il existe un Contexte c qui soit admissible dans S_a mais pas dans $S_{a'}$. En ce sens, ContextAA se rapproche du modèle *Belief-Desire-Intention* [80], ou BDI, surtout pour qui tient compte de la difficulté de déterminer des faits quant à la réalité enrichie en présence de sources conflictuelles de Contexte et de composants potentiellement défectueux.

Cette perspective locale à l'Agent comporte au moins deux avantages marqués pour ContextAA, soit celui de favoriser l'autonomie des Agents, en le rendant indépendant d'une autorité tierce sur la qualité du Contexte, et celui de restreindre le Contexte auquel un Agent est confronté à ce que cet Agent décrit comme étant pertinent pour lui, évitant à l'Agent le fardeau de devoir examiner l'ensemble des Contextes mis à sa disposition dans N_h. Ce filtre implicitement appliqué sur la masse de Contexte disponible réduit la charge de travail de l'Agent et facilite son fonctionnement sur des nœuds à faibles ressources.

4.8.3 Avantages et inconvénients du cadre ontique

Le cadre ontique O_a permet à l'Agent a d'être porteur de ce qui joue pour lui le rôle d'une ontologie qui lui est propre. Cette partie de a fait de lui le porteur d'un regard actif et singulier sur l'espace intelligent, et lui garantit une forme d'indépendance en situation de faible connectivité. En ce sens, le cadre ontique rapproche a d'un état d'autonomie.

L'absence d'ontologie commune aux Agents impose en retour les mécanismes de mise en correspondance des Contextes et de transformation de Contexte sur lesquels repose ContextAA. Cet effort singulier, qui représente pour nous un gain de résilience et d'autonomie, constitue aussi un fardeau que des Agents limités à un environnement délimité dans l'espace pourraient ne pas avoir à traîner. Conséquemment, l'approche proposée par ContextAA et reposant sur un cadre ontique propre à l'Agent plutôt que sur une Ontologie commune à tous les Agents est une conséquence de la perspective architecturale micro [2] que ContextAA supporte de manière intrinsèque.

Le formalisme du Contexte, comme celui des Agents, dépend sur le plan opératoire du soutien de plusieurs composants architecturaux, discutés au chapitre 5 et qui font en sorte que ContextAA dans son ensemble fonctionne tel qu'espéré.

Chapitre 5
ContextAA – Considérations architecturales

« L'architecture, c'est une tournure d'esprit et non un métier. » (Le Corbusier / Lettre)

En plus de ses fondements théoriques décrits à au chapitre 4, ContextAA repose sur une architecture répartie conçue pour faire face à la question de la mobilité et de l'approche micro [2] dans l'espace intelligent ouvert. Ses principaux éléments constitutifs sont les composants Hôtes, les Agents, les objets autonomes et le Contexte. S'ajoutent à ces éléments les espaces contextuels, à la fois des Agents et des Hôtes, de même que les canaux de Contexte.

5.1 Hôte

Dans ContextAA, un Hôte est un processus déployé sur un nœud, fixe ou mobile, et sur lequel sont déployés des Agents, comme le montre la figure 12. Un Hôte joue plusieurs rôles, dont ceux :

- d'unité de déploiement sur des nœuds;
- de conteneur d'Agents;
- du gestionnaire du cycle de vie des Agents qui y sont déployés, que nous appelons souvent le gestionnaire d'Agents tout simplement;
- de strate isolante entre les Agents et le réseau, donc entre $\forall a: A, h: H, a \in h$ et $h': H, h' \in N_h$; et
- de strate isolante entre les Agents et les considérations de concurrence.

Cette courte liste de rôles met en relief que les rôles effectifs d'un Hôte et de ses Agents standards se confondent à bien des égards, ces derniers assurant plusieurs des services de leur Hôte de manière organique.

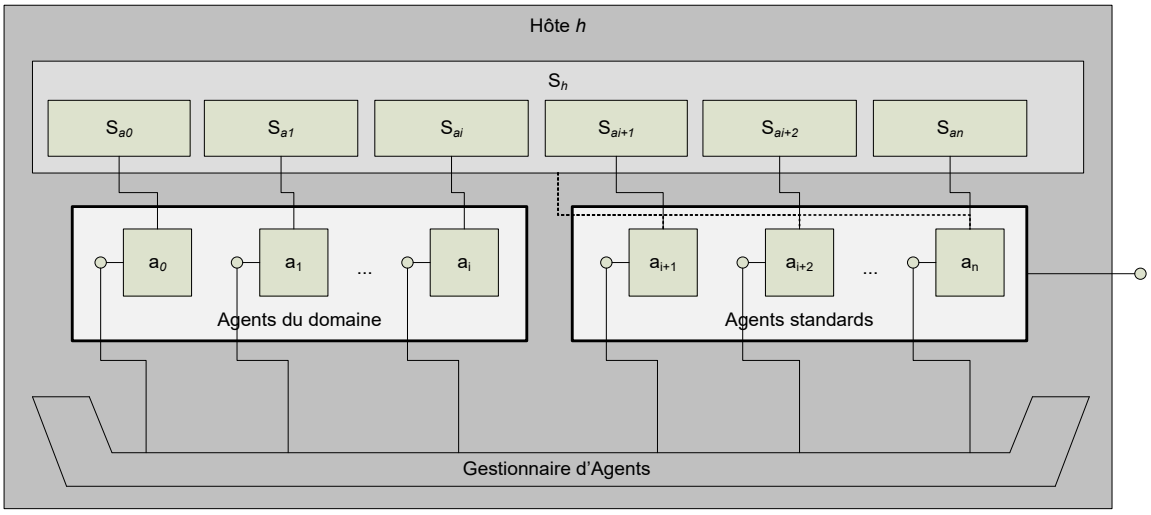


Figure 12 - Composant Hôte (vue aérienne)

Un Hôte dans ContextAA est semblable à un *Bundle* OSGi [76] ou à un composant OpenCom [26], en ce sens que chacun gère le déploiement et le cycle d'exécution d'Agents. ContextAA se distingue toutefois de ceux-ci sur le plan opératoire, du fait qu'un Hôte y est pleinement guidé par le Contexte, hormis les interactions avec les composants tierset celles avec le matériel.

Parce qu'un Hôte n'offre en soi que très peu de services, la plupart étant plutôt implémentés par les Agents standards qui y sont déployés, un Hôte est une entité très légère et peu coûteuse en termes de ressources. Dans la perspective de ContextAA, il s'agit d'une caractéristique souhaitable d'un module destiné au déploiement sur appareils mobiles ou à ressources susceptibles d'être limitées.

5.1.1 Caractéristiques d'un Hôte

La figure 12 ci-dessus met en relief certaines caractéristiques de l'architecture de ContextAA, en particulier celles en lien avec un Hôte :

- les Agents y sont pris en charge par un gestionnaire d'Agents qui les sépare en deux groupes, soit celui des Agents standards et celui des Agents du domaine. Le gestionnaire d'Agents, en pratique, est un objet autonome prenant en charge le cycle de vie des Agents déployés sur un Hôte;
- tout Agent implémente une même interface, destinée à assurer son intégration dans la mécanique du gestionnaire d'Agents. Ainsi, bien que les Agents standards et les Agents du domaine diffèrent à la fois sur le plan du rôle et des privilèges, il se trouve que pour le gestionnaire d'Agents, une fois un Agent installé dans un Hôte, que cet Agent soit standard ou du domaine n'entraîne pas de différence fondamentale sur le plan du cycle d'opérations;
- l'espace contextuel S_h d'un Hôte h est tel que $S_h \supset \bigcup_{a \in h} S_a$;
- à titre de rappel, un Agent a interagit avec son espace contextuel S_a. Pour un Agent du domaine, cette restriction est stricte, alors qu'un Agent standard a le privilège d'accéder aussi à l'espace contextuel S_h de son Hôte h, et par là aux espaces contextuels des autres Agents sur h; peut interagir avec un objet autonome à travers un canal de Contexte; et peut donc transiger indirectement avec d'autres Hôtes et des composants tiers.

La figure 12 met en relief l'isolement qu'assure un Hôte entre ses Agents et les autres Hôtes. Un Agent du domaine a n'interagit qu'avec S_a; un Agent standard sur h peut interagir avec son espace contextuel, avec S_h, avec le matériel sous-jacent (voir figure 6) et avec des canaux de Contexte. Pour sa part, un objet autonome sur h interagit avec les entités extérieures à h, mais n'interagit avec h que par le biais des canaux de Contexte auxquels il est associé.

Le design initial de ContextAA prévoyait qu'un Agent expose directement ses services à travers des interfaces conformes aux standards en vigueur sur le Web [93]. L'accent mis sur le support de substrats faibles en ressources a mis en relief les difficultés inhérentes au support plein et entier de technologies Web telles que celles envisagées à l'origine, et a mené ContextAA à déplacer les interactions avec des technologies tierces vers des objets autonomes (§**Erreur ! Source du renvoi introuvable.**), donc à ne pas les placer au cœur de ce qu'est devenu le composant Hôte mais bien en périphérie.

L'évolution de nos travaux a aussi fait en sorte que l'isolement des Agents du réseau et de la concurrence gagne en importance dans le design de ContextAA. En effet, il est devenu clair que le développement d'Agents du domaine serait susceptible d'entraîner le recours à des spécialistes d'autres domaines que l'informatique, et que se préoccuper de concurrence risquerait de rendre leur tâche plus ardue. L'isolement que nous avons mis en place permet de raisonner sur le comportement d'un Agent sur la base seule de questions posées et de connaissances acquises.

La figure 6 offre une vue stratifiée d'un Hôte et de ses principaux éléments constitutifs. Elle met entre autres en relief que :

- les Agents standards et les objets autonomes seuls ont un accès direct à la strate « système ». Cet accès privilégié permet de les représenter de manière à tirer profit au maximum de cette couche dans ce qu'elle a de plus spécifique, ce qui explique qu'une implémentation donnée d'un Agent standard soit typiquement spécifique à une plateforme;
- les objets autonomes seuls interagissent directement avec les entités extérieures à leur Hôte. Ce sont eux qui interviennent pour interfacer avec des composants tiers pour assurer l'interaction avec $h' : H, h' \in N_h$;
- les échanges entre objets autonomes et Agents standards transitent par des canaux de Contexte. Ceci participe à l'isolement des Agents de ce qui a trait à la concurrence, tout en assurant l'autonomie opératoire des objets autonomes qui peuvent s'exprimer sous la forme de fils d'exécution ou de processus distincts de leur Hôte;
- le fait que les Agents du domaine n'interagissent qu'avec leur propre espace contextuel assure entre autres choses une forme de confidentialité quant à ce qu'un Agent du domaine fait du point de vue des autres Agents du domaine.

Cette architecturemet en valeur les raisons pour lesquelles les Agents standards et les objets autonomes ont typiquement une implémentation propre à la plateforme sous-jacente, et sont implémentés sur mesure : à la manière des bibliothèques standards de plusieurs langages de programmation, ces entités de ContextAA jouent un rôle structurant pour le système dans son ensemble, et doivent être d'une grande efficacité. En retour, pouvant être strictement dépendant du Contexte, un Agent du domaine peut être « pur Contexte »; étant contextifiable, l'Agent du domaine devient mobile au même titre que le Contexte qu'il produit ou qu'il consomme, et peut migrer d'un Hôte à l'autre en fonction des besoins décrits par sa mission.

La combinaison des Agents standards et des objets autonomes constitue la gamme de services d'un Hôte. À travers ces entités, un Hôte joue le rôle d'un intergiciel pour les Agents du domaine.

5.2 Objets autonomes

Sur un Hôte h, les tâches réalisées par chaque Agent a s'inscrivent dans le cycle normal d'opérations de h. Certaines opérations sur h doivent par contre être réalisées hors de ce cadre itératif régulier : des interactions avec un usager dans une application reposant sur ContextAA, par exemple; la mise à jour de N_h lors de la disparition ou de l'apparition d'Hôtes voisins; la transmission et la réception de Contexte entre Hôtes; l'exposition ou la consommation de services en interaction avec des composants tiers; les opérations d'entrée/ sortie autres que celles requises au démarrage ou lors de l'arrêt d'un Hôte; etc.

Pour ces tâches, ContextAA utilise des objets autonomes. Un objet autonome est une entité encapsulant une unité d'exécution concurrente (fils d'exécution, *protothread*, processus, etc.) et réalisant une tâche qui ne s'intègre pas harmonieusement dans le cycle normal d'opération des Agents (§5.3). Une tâche ne s'intègre pas harmonieusement dans le cycle normal d'opérations si elle réduirait alors la réactivité de son Hôte; il s'agit donc typiquement d'une tâche dont l'exécution peut être longue ou bloquante.

Les objets autonomes peuvent être vus comme des satellites d'un Hôte, au sens où ils opèrent surtout dans sa périphérie, selon des modalités qui leur sont propres (outre la contrainte de communiquer par des canaux de Contexte, décrits à la section suivante).

5.2.1 Canaux de Contexte

Chaque objet autonome d'un Hôte h s'exécute indépendamment des autres, et à un rythme indépendant du rythme d'opération de h en tant que tel, ce qui assujettirait les interactions entre l'Hôte et ses objets autonomes à des conditions de course si ces interactions se faisaient directement et sans synchronisation. Pour cette raison, un lien de communication synchronisé mais permettant des accès non-bloquants est placé entre un objet autonome et son Hôte.

Chacun de ces liens est constitué d'un canal de Contexte, et se positionne techniquement entre un objet autonome et un Agent standard, ou (plus rarement) entre deux objets autonomes d'un même Hôte.

Un canal de Contexte est un lien unidirectionnel synchronisé sur lequel transite du Contexte, semblable en partie au système de boîtes postales du modèle d'acteurs de Hewitt et al. [44] ou aux canaux du langage Go. Lorsque cela s'avère pertinent, une communication bidirectionnelle entre un Agent standard et un objet autonome repose simplement sur une paire de canaux de Contexte.

5.2.2 Canaux de Contexte et isolement des Agents

Un effet de bord appréciable de l'isolement des questions de concurrence en périphérie d'un Hôte est qu'outre le passage par des canaux de Contexte dans le cas de certains Agents standards, un Agent ne réalise pas de lui-même de tâches concurrentes au cycle normal d'opérations de son Hôte. Conséquemment, aucune synchronisation n'est requise lors d'accès aux espaces contextuels, ce qui accélère les accès à cette importante structure de données.

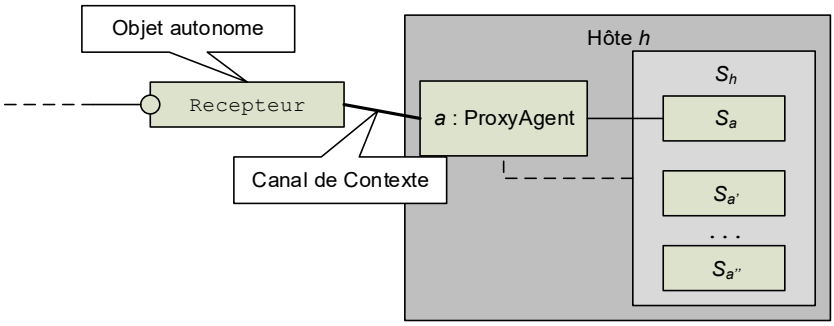


Figure 13 - Communication entre objet autonome et Agent standard (vue aérienne)

La figure 13 présente un exemple où un objet autonome *Recepteur* consomme du Contexte d'une source susceptible d'être bloquante. Le Contexte ainsi consommé est projeté sur un canal de Contexte pour être éventuellement consommé par un Agent standard a (*ProxyAgent* dans la figure). Lors de sa prochaine itération, a pourra consommer ce Contexte du canal pour l'intégrer aux espaces contextuels appropriés.

5.2.3 Gestionnaire d'Agents

Le gestionnaire d'Agents d'un Hôte est l'entité qui donne directement à chaque Agent l'opportunité de réaliser ses tâches; il sollicite directement les services des Agents et ne passe pas par des canaux de Contexte. Le gestionnaire d'Agents peut donc être vu comme le seul objet autonome qui n'est pas tant un satellite de l'Hôte qu'une entité qui participe directement à la réalisation de ses fonctions essentielles.

5.2.4 Communicateur et autres objets autonomes associés à la communication

Sur chaque Hôte h se trouve un objet autonome nommé Communicateur, dont le rôle est d'orchestrer une fédération d'objets autonomes assurant le fonctionnement des divers mécanismes de communication avec le monde extérieur à h, en particulier $\forall h' \in N_h$.

Le Communicateur d'un Hôte h partage des canaux de communication avec divers Agents standards. Là où le Communicateur et les objets autonomes sous sa gouverne prennent en charge les opérations concurrentes au cycle normal d'opérations du gestionnaire d'Agents (§5.2.3), les Agents avec lesquels le Communicateur transige du Contexte interagissent quant à eux avec S_h et accomplissent les tâches qui sont préférablement faites à l'intérieur de ce cycle.

Le Communicateur est lancé et arrêté par son Hôte, et est en soi responsable du démarrage et de l'arrêt éventuel de trois types d'objets autonomes :

- les Découvreurs, responsables de détecter les nouveaux arrivants dans N_h et de constater les Hôtes quittant N_h;
- les Émetteurs, responsables de transmettre du Contexte vers un Hôte de N_h; et
- les Récepteurs, responsables de consommer du Contexte provenant d'un Hôte de N_h.

Le Contexte destiné à être transmis par chaque Émetteur doit d'abord passer par les canaux de Contexte du Communicateur. De même, le Contexte consommé par chaque Récepteur doit transiter par le Communicateur pour être intégré à S_h.

Le Communicateur participe à deux canaux de Contexte, soit un canal entrant pour envoyer du Contexte vers un Agent standard sur son Hôte, et un canal sortant pour recevoir du Contexte d'un Agent standard sur son Hôte. La dénomination entrant / sortant adoptée ici s'exprime en fonction de la perspective de l'Hôte, pas de celle du Communicateur. Une seule paire de canaux suffit pour un Hôte donné; le Communicateur assure la répartition entrante et sortante du Contexte à travers ses Émetteurs et ses Récepteurs.

5.2.4.1 Découvreur

Un Découvreur est un objet autonome chargé de tenir à jour le voisinage réseau N_h d'un Hôte h. Il a donc pour rôle de chercher à entrer en contact avec des Hôtes susceptibles de faire partie de N_h et de détecter le départ d'Hôtes ayant participé à N_h.

En pratique, l'implémentation d'un Découvreur dépend fortement des réseaux auxquels participe h. Sur le plan général, un Découvreur typique sur h sonde périodiquement l'environnement pour découvrir des Hôtes susceptibles de faire partie de N_h, bien que d'autres modalités, par exemple réactives, puissent être implémentées sans perte de généralité puisqu'en pratique, le Découvreur informe le Communicateur d'une nouvelle connexion de manière événementielle, et ce dernier relaie cette information à son Hôte par un canal de Contexte.

Des mécanismes sont mis en place pour éviter que des Découvreurs sur h et h', s'ils s'entre-découvrent mutuellement, ne créent une double connexion entre eux.

5.2.4.2 Émetteur

Pour chaque Hôte $h' \in N_h$, suite à la découverte de h' par un Découvreur sur h et à son insertion subséquente dans N_h, le Communicateur lance un objet autonome Émetteur ayant pour rôle de faire suivre à h' le Contexte qui lui est destiné, qu'il s'agisse de requêtes pour du Contexte, produites sur h ou ayant transité par h, ou encore de réponses à des requêtes reçues antérieurement de h'.

5.2.4.3 Récepteur

Pour chaque Hôte $h' \in N_h$, suite à la découverte de h' par un Découvreur sur h et à son insertion subséquente dans N_h, le Communicateur lancera aussi un objet autonome Récepteur ayant pour rôle de consommer le Contexte reçu de h', qu'il s'agisse de requêtes pour du Contexte, ou encore de réponses à des requêtes envoyées antérieurement par h vers h'.

5.2.5 Interaction avec des composants tiers

Puisque le monde extérieur à un Hôte est fait de composants de diverses natures, soutenues par une variété de technologies, il ne serait pas raisonnable de présumer que toutes les technologies tierces adoptent, du moins à court terme, une interface basée sur du Contexte au sens où l'entend ContextAA.

Pour cette raison, il incombe à ContextAA de faire le pont avec les technologies tierces et d'offrir des composants à double vocation, exposant sous forme de Contexte les métaphores, interfaces et données de ces technologies et traduisant du Contexte vers ces autres formats.

Les objets autonomes que sont les Découvreurs, les Récepteurs et les Émetteurs sont des exemples concrets de telles interfaces pour fins protocolaires. La métaphore d'accès à un actuateur, décrite dans §5.5, peut être traduite sans peine en un appel de service. De même, l'annexe J montre comment il est possible de traduire de Contexte à XML et inversement, alors que l'annexe K montre comment il est possible de faire de même entre Contexte et JSON.

Nous avons espoir que la banque d'interfaces entre Contexte et formats tiers croîtra rapidement avec l'utilisation de ContextAA, mais soulignons qu'il existe déjà des ponts simples entre ContextAA et des technologies répandues; par exemple, Ponce et al. [82] construit un cadriciel, AmI-DEU, ajoutant une couche sémantique sur la base de ContextAA, alors que Ponce et al. [81] construit sur ce cadriciel un modèle d'activité facilitant la contribution d'experts de domaines autres que celui de l'informatique. L'implémentation proposée par cet article utilise d'ailleurs un objet autonome assurant le pont entre le protocole MQTT [48] et le milieu pur-Contexte de ContextAA.

Cycle de vie d'un Agent

L'une des responsabilités d'un Hôte est de prendre en charge le cycle de vie des Agents qui y sont déployés. Le cycle de vie d'un Agent standard est le même que celui d'un Agent du domaine : la distinction essentielle entre les deux catégories d'Agents reposant sur leurs privilèges respectifs et non pas sur les étapes par lesquelles un Agent passe dans chaque cas.

Le cycle de vie s'exprime en deux temps. Tout d'abord, la vie d'un Hôte s'inscrit dans une séquence d'étapes telle que celle présentée à la figure 14.

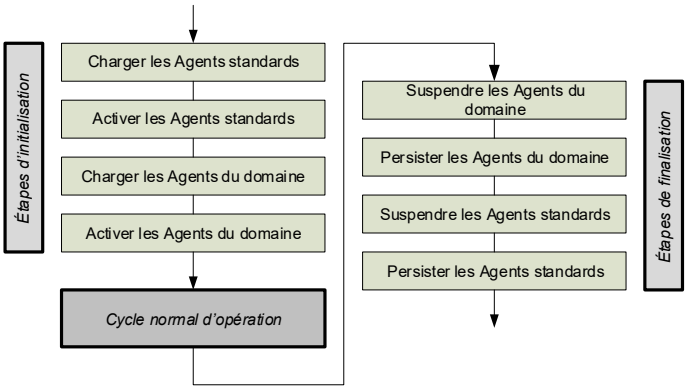


Figure 14 - Cycle de vie des Agents (vue générale)

L'initialisation des Agents standards précède celle des Agents du domaine, ces derniers étant susceptibles de compter sur les services des Agents standards dans l'exercice de leur fonction, mais cet ordonnancement n'est pas essentiel du fait que les divers Agents n'entrent vraiment en fonction que lorsque débute le cycle normal d'opération du gestionnaire d'Agents. Pour la même raison, la suspension des Agents standards suit celle des Agents du domaine. Par conséquent, du point de vue d'un Agent du domaine, les Agents standards sont en place en tout temps sur l'Hôte, quel qu'il soit.

Concrètement, le fait que les Agents standards soient initialisés avant les Agents du domaine et finalisés après les Agents du domaine n'a, dans la majorité de cas, pas d'impact direct sur l'ordre d'exécution des Agents dans le cycle normal des opérations. En effet, les Agents standards sont subdivisés en deux groupes, soit ceux qui, à chaque itération du cycle de vie, doivent s'exécuter avant les Agents du domaine et ceux qui doivent s'exécuter après. Selon la perspective des Agents du domaine, cette distinction est surtout importante lors de la première itération du cycle et vie et lors de la dernière, mais il se trouve que les Agents standards s'offrent aussi mutuellement des services, et que leur position dans le cycle normal d'opérations peut être établie selon les besoins de l'Hôte qui les accueille.

Dans quelques cas toutefois, en particulier celui du gestionnaire de Contexte, la position dans le cycle a une importance du fait qu'il importe que les autres Agents standards tenus responsables de traiter le Contexte en soutien aux Agents du domaine soient intervenus entre le moment où les Agents du domaine ont publié leur Contexte et celui où le gestionnaire de Contexte prend en charge les méta-Contextes susceptibles de faire en sorte que ces Contextes soient oubliés.

5.3 Cycle d'opération d'un Agent

Le cycle normal d'opération d'un Hôte, donc de son gestionnaire d'Agents, est d'appliquer à chaque Agent son propre cycle normal d'opération.

Le cycle d'opération d'un Agent a, mentionné dans la figure 14 ci-dessus, s'exprime quant à lui comme suit.

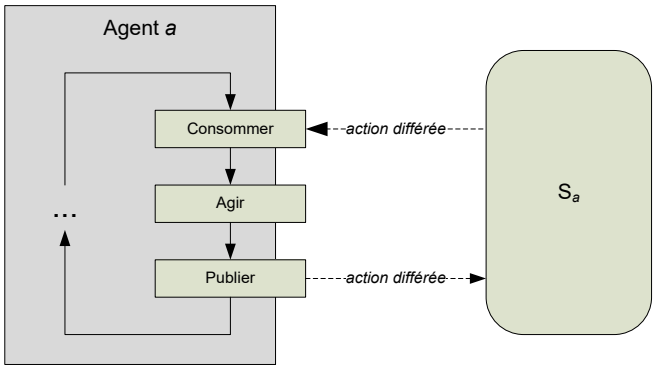


Figure 15 - Cycle d'opération d'un Agent $\boldsymbol{a}$

La mention « action différée » dans la figure 15 tient au fait que les requêtes pour du Contexte exprimées par a sont traitées par des Agents standards entre deux passages distincts du cycle d'opération de a, et apparaissent pour a comme des actions asynchrones. Du fait que certains Contextes impliquent des échanges avec d'autres Hôtes, il se peut que plusieurs passages par le cycle d'opération d'un même Agent soient nécessaires pour qu'une demande de Contexte soit desservie – en pratique, il est même possible qu'une requête pour du Contexte n'ait jamais de réponse.

Le cycle d'opération d'un Agent standardise des pratiques typiques à des entités dépendantes du Contexte, au sens où ces entités consomment, traitent et publient du Contexte. Pour un Agent du domaine a, les requêtes publiées dans S_a sont propagées vers leurs destinataires potentiels par l'action d'Agents standards, et les réponses aux requêtes de a reçues par des Agents standards sont inscrites dans S_a par ces derniers.

Les opérations *consommer*, *agir* et *publier* d'un Agent du domaine $a = \{M, O, R, S\}$ sont typiquement décrites sous forme de Contexte sur la base de M et de O principalement, alors que pour un Agent standard elles sont typiquement codées « en dur » pour tirer maximalement profit des facilités de la plateforme sous-jacente.

5.4 Concordance entre Contexte requis et Contexte offert

ContextAA offre plusieurs mécanismes pour réaliser la mise en concordance entre deux Contextes, une opération dont l'importance est mise en relief dans §4.6. La présente section examine les principaux mécanismes prévus à cet effet, soit l'équivalence entre les Contextes, la mise en correspondance avec un modèle et la distance sémantique.

5.4.1 Équivalence entre Contextes

Le mécanisme de mise en concordance de Contextes le plus souvent utilisé est l'équivalence des Contextes, sur la base de l'égalité des noms.

Ce mécanisme est névralgique au bon fonctionnement de ContextAA; sur un Hôte donné, l'implémentation de ce mécanisme doit être fortement optimisé. Rappelons que cette équivalence se calcule sur la base d'une comparaison en surface des Contextes plutôt que sur une comparaison en profondeur, et suffit souvent du fait que l'unicité des noms est un *a priori* de plusieurs contextes dans lesquels ContextAA regroupe les Contextes, soit l'espace contextuel de l'Hôte et celui de chaque Agent.

À titre d'exemple, l'équivalence entre Contextes suffit pour permettre à l'Agent a de retrouver le Contexte c dans S_a, la forme du nom de c étant sous son contrôle. Ceci fait d'une importante proportion des opérations sur le Contexte des opérations à très faible coût.

5.4.2 Mise en correspondance avec un modèle

Un mécanisme plus flexible est la mise en correspondance des modèles, par lequel un demandeur de Contexte exprime ce qu'il souhaite obtenir sous la forme d'un modèle que les Hôtes producteurs chercheront à mettre en correspondance avec le Contexte qu'offrent les Agents sous leur gouverne. Le rôle de ContextAA est d'assister la mise en correspondance de l'offre et de la demande.

Ce mécanisme permet d'exprimer des idées complexes, comme celle décrite par la figure 16. Supposons qu'à un point de l'exécution de $h: H$, il existe $a \in h$ nommé $abcd$ tel que S_a est :

```
abcd {
  public {
    388 {
      query:388 {
        local {
          **
          { celsius }
        }
      }
    }
  }
  local {
    agent-info {
      id { abcd }
      name { "tiny agent" }
    }
  }
}
```

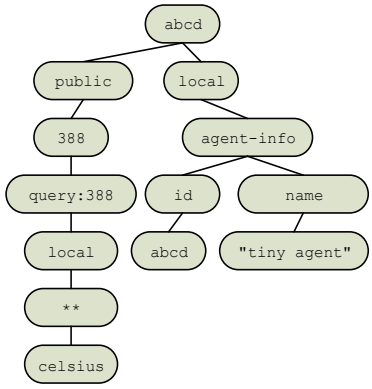


Figure 16 - Vision arborescente d'un Agent

Il serait possible à un demandeur de décrire tout Contexte ayant pour racine $abcd$ de la façon suivante :

```
abcd
```

Si cette expression est soumise en tant que requête dans h, la réponse sera l'ensemble des Contextes connus de l'Agent nommé $abcd$, car $\forall a \in h$ la racine de S_a est $name(a)$.

Un demandeur peut préciser sa pensée et exprimer ceci :

```
abcd { local }
```

Considérant l'exemple ci-dessus, la réponse à cette requête sera la branche de droite de l'arbre (ce qui apparaît dans l'encadré de la figure 17) :

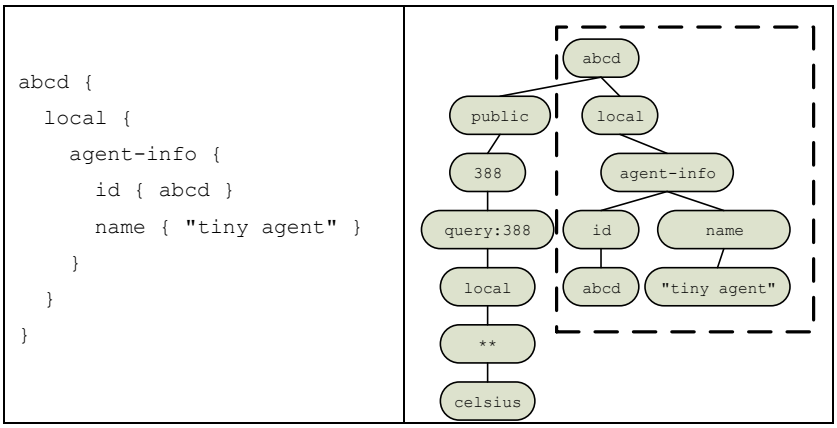


Figure 17 - Vision arborescente d'un Agent (section réponse)

Des prédicats peuvent être appliqués à ce Contexte. Par exemple, examiner s'il contient un Contexte nommé $celsius$:

$$compose\left\{\#1\{abcd\{local\}\}\#2\left\{has\{**\{celsius\}\}\right\}\right\}$$

ce à quoi la réponse sera $[false]$. De même, si la question est de savoir si ce Contexte contient le texte $"tiny\ agent"$, il est possible d'exprimer :

$$compose\left\{\#1\{abcd\{local\}\}\#2\left\{has\{**\{"tiny\ agent"\}\}\right\}\right\}$$

pour lequel la réponse sera $[true]$.

Le langage par lequel s'exprime la mise en correspondance de Contextes dans ContextAA est encore en évolution, mais comprend d'ores et déjà un certain nombre de mots et de structures grammaticales, dont une partie est décrite aux annexes A et B.

5.4.2.1 Impact des qualifications d'accès au Contexte

ContextAA utilise un mécanisme informel de qualification d'accès, inspiré de mécanismes répandus dans les langages orientés objets, soit l'apposition des noms de Contextes $public$, $local$ et $private$. Bien que ces noms de Contextes n'aient pas de sens inhérent dans ContextAA, les Agents standards chargés de la prise en charge des requêtes pour du Contexte se basent sur eux pour filtrer certaines opérations :

- lors de la résolution d'une requête pour du Contexte provenant de $h'\colon H$, un Agent standard $a \in h, h\colon H, h \neq h'$ ne considérera que les Contextes placés sous un Contexte de nom $public$;
- lors de la résolution d'une requête pour du Contexte provenant de $h\colon H$, un Agent standard $a \in h$ ne considérera que les Contextes placés sous un Contexte de nom $public$ ou $local$;

- lors de la résolution d'une requête pour du Contexte provenant de $a' \in H, h: H$, un Agent standard $a \in h$ considérera les Contextes placés sous un Contexte de nom $public$, $local$ ou $private$.

Ceci permet à un Agent $a \in h$ de « marquer » chacun des Contextes qu'il publie pour qu'ils ne soient qu'à son usage propre ($private$), à l'usage des Agents du même Hôte ($local$) ou pour usage général ($public$), ou encore de préciser qu'il émet une requête pour du Contexte dans S_a ($private$), dans S_h ($local$) ou encore dans $h': h' \in N_h$.

5.4.2.2 Critères de Contextes

Un critère de Contexte est une expression qui peut être appliquée à du Contexte et dont le fruit est l'un de deux Contextes particuliers, soit $true$ et $false$, une variante des prédicats décrits dans §4.5.1.

Lorsque ContextAA rencontre un critère de Contexte sous forme contextifiée, ce critère est réifié en un foncteur représentant l'opération décrite, et ce foncteur est par la suite appliqué comme une transformation $\tau :: C \rightarrow List\langle C \rangle$ dont l'image est restreinte à $([true], [false])$. Un critère réifié s'avère si son application sur un Contexte produit $[true]$.

Les critères de Contextes standards de ContextAA sont listés à l'annexe C, et incluent :

- le critère $has\{patt\}$, qui s'avère pour $c: C$ seulement s'il y a correspondance entre $patt$ et c. Ce critère permet de faire une recherche multi-niveaux. Un exemple d'utilisation est donné dans §5.4.2;
- le critère $is\{patt\}$, qui s'avère pour $c: C$ seulement s'il y a correspondance entre $patt$ et c. Ce critère se limite à une mise en correspondance directe. Un exemple d'utilisation serait (considérant le Contexte de la figure 16 plus haut, où on trouve un Contexte terminal de nom $abcd$), celui-ci :

  $$compose\left\{\#1\{abcd\{local\}\}\#2\{leaf\_nodes\}\#3\left\{keep\_if\{is\{abcd\}\}\right\}\right\}$$

  où $leaf\_nodes$ et $keep\_if$ sont des transformations de Contextes (section suivante);
- le critère is_date qui s'avère pour $c: C$ seulement si c est convertible en date;
- le critère is_number qui s'avère pour $c: C$ seulement si c est convertible en nombre;
- le critère not qui s'avère pour $c: C$ seulement si c est $false$; etc.

5.4.2.3 Transformations de Contextes

Une transformation de Contexte est une expression qui peut être appliquée à du Contexte et dont le fruit est une séquence de Contextes.

Lorsque ContextAA rencontre une transformation de Contexte, cette transformation est réifiée en un foncteur représentant l'opération décrite, et ce foncteur est par la suite appliqué sur du Contexte. Ceci permet à un Hôte de porter avec lui les transformations susceptibles d'être utiles aux Agents placés sous sa responsabilité, d'en acquérir de nouvelles de son environnement au besoin, et même d'évacuer celles qui ne semblent plus essentielles si les ressources disponibles semblent insuffisantes.

Les transformations standards de ContextAA sont listées à l'annexe D, et incluent :

- la transformation $exhaustive$ qui, appliquée à $c: C$, produit la liste des sous-Contextes de c. Cette opération peut être coûteuse : par exemple, appliquée à $a\left\{b\{c\}d\{e\ f\{g\}\}\right\}$, elle produira $\left[a\left\{b\{c\}d\{e\ f\{g\}\}\right\}, b\{c\}, c, d\{e\ f\{g\}\}, e, f\{g\}, g\right]$;
- la transformation or accompagnée des modèles $m_0, m_1, \dots, m_{n-1}$ qui, appliquée à $c: C$, produit la fusion de la mise en correspondance de c avec chacun de ces modèles;
- la transformation $filter\_if$ accompagnée du modèle m qui, appliquée à $c: C$, produit $\left[c': C \parallel \neg\left(isSubCtx(c', c) \wedge matches(m, c')\right)\right]$;
- la transformation $keep\_if$ accompagnée du modèle $patt$ qui, appliquée à $c: C$, produit $[c': C \parallel isSubCtx(c', c) \wedge matches(m, c')]$;
- la transformation $leaf\_nodes$ qui, appliquée à $c: C$, produit la liste des sous-Contextes terminaux de c. Par exemple, appliquée à $a\left\{b\{c\}d\{e\ f\{g\}\}\right\}$, elle produira $[c, d, e, g]$;
- la transformation cut_root qui, appliquée à $c: C$, produit $value(c)$. Elle représente ce que nous nommons une suppression de racine;
- etc.

5.4.2.4 Autres opérations de Contexte

Certaines opérations jouent un rôle qu'il est difficile de classer dans $Crit$ ou dans T. Ces opérations sont listées à l'annexe E.

En particulier, l'opération $compose$ accompagnée des opérations sur du Contexte $op_0, op_1, \dots, op_{n-1}$ fait en sorte d'exécuter les opérations $op_0, op_1, \dots, op_{n-1}$ selon un ordre déterminé par la structure suivante :

- chaque opération est de la forme $\#n\{c\}$ (§4.4.7.6) où $n \in \mathbb{Z}, c: C$. Dans cette forme, c est l'opération à appliquer et n est un indicateur guidant l'ordonnancement des opérations;
- pour deux opérations $op = \#n\{c\}, op' = \#m\{c'\}$:
  - l'opération c sera appliquée avant l'opération c' si $n < m$;
  - l'opération c' sera appliquée avant l'opération c si $m < n$; enfin,
  - l'ordre d'application de c et c' sera indéterminé si $n = m$. Ainsi, c peut précéder c' tout comme c' peut précéder c. Conséquemment, c et c' pourraient être traités concurremment, mais ne le sont pas dans l'était actuel de ContextAA pour des raisons d'isolation face aux considérations de concurrence.

Une opération $compose$ applique chaque opération sous sa gouverne en séquence, et alimente chaque opération avec le Contexte généré par la précédente.

L'opération $structured\_like$, accompagnée de $c, c': C$, exprime en pratique la fonction suivante :

$$structured\_like :: C \to C \to C$$

$$structured\_like(c: C)(c': C) \triangleq Contexte\left(concat\left(\sigma(c, c'), name(c)\right), value(c)\right)$$

L'opération $valued\_like$, accompagnée de $c, c' : C$, exprime en pratique la fonction suivante :

$$valued\_like :: C \to C \to C$$

$$valued\_like(c : C)(c' : C) \triangleq Contexte\left(concat\big(\varrho(c, c'), name(c)\big), value(c)\right)$$

5.4.3 Fournisseur d'équivalences

Certaines similitudes entre Contextes peuvent être reconnues *de facto*. Parmi les plus simples se trouvent les conversions de référentiels et les traductions littérales terme à terme, ce qui constitue une forme limitée d'internationalisation. Permettre de telles transformations accroît la mobilité des Agents leur ouvrant les portes de l'adaptabilité, mais seulement si ces Agents n'ont pas à connaître au préalable toutes les transformations possibles.

Pour y arriver, un Agent de ContextAA apprend en fonction des besoins. Un Agent a exprime ses requêtes selon le vocabulaire dans S_a. Ce vocabulaire utilise initialement les noms convenant à la mission de a telle qu'elle a été conçue. Par exemple, si la mission de a est d'évaluer le confort des lieux en fonction de certains paramètres, dont la pression atmosphérique exprimée en hPa, l'une des requêtes de Contexte émises par a pourra contenir, en tant que sous-Contexte :

$$35\left\{public\{**\ \{unite\{hPa\}valeur\}\}\right\}$$

Ici, le 35 est un identifiant de Contexte et n'a pas de sens particulier autre que celui-ci. Ce qu'il faut retenir pour notre propos est que le demandeur souhaite un Contexte comprenant à la fois le sous-Contexte $unite\{hPa\}$ et le sous-Contexte $valeur$. Cette requête peut porter fruit en laboratoire et en période de tests si les fournisseurs utilisent un vocabulaire connexe.

En pratique, dû à des changements dans son environnement ou à des déplacements, a peut trouver dans son voisinage un fournisseur a' émettant le Contexte suivant :

$$431\left\{public\left\{pressure\{unit\{hPa\}value\{1013.25\}precision\{0.001\}\}\right\}\right\}$$

Une autre possibilité pour un Contexte portant en quelque porte le même sens serait :

$$431\left\{public\left\{pression\{unite\{kPa\}valeur\{101.325\}precision\{0.001\}\}\right\}\right\}$$

Présumant que l'un de ces Contextes soit publié et l'autre soit demandé, la correspondance entre Contexte publié et Contexte requis est directe, à ceci près que l'un utilise des noms anglais et l'autre des noms français. Ici, le producteur et le consommateur sont en accord sur le vocabulaire, mais pas sur l'unité de mesure. Dans un cas comme dans l'autre, il est possible de faire des correspondances directes et simples entre la demande et l'offre :

- en constatant qu'il existe une correspondance entre les mots de la demande et ceux de l'offre; ou
- en constatant qu'il existe une transformation de référentiel menant du format utilisé pour l'offre au format utilisé pour la demande.

De telles transformations sont simples au sens où chacune d'elles implique un simple changement, que ce soit un mot pour un autre mot ou, de manière plus générale, de valeur à valeur en fonction du référentiel de la source et du référentiel de la destination.

Pour repérer de telles transformations, ContextAA utilise un fournisseur d'équivalences. Le rôle d'un fournisseur d'équivalences est double : repérer une équivalence possible pour un Contexte, et réifier cette équivalence. Concrètement, pour une requête exprimée sous la forme $patt: C$, deux options se présentent d'office, soit :

$$\exists c: C\big(match(patt, c)\big)$$

$$\nexists c: C\big(match(patt, c)\big)$$

Le fournisseur d'équivalence remplace ces options par :

$$\exists c: C\left(\neg match(patt, c) \wedge \exists \tau: \mathrm{T}\big(\exists c' \in C: c' = \tau(c) \wedge match(patt, c')\big)\right)$$

$$\nexists c: C\left(\neg match(patt, c) \wedge \exists \tau: \mathrm{T}: \big(\exists c' \in C: c' = \tau(c) \wedge match(patt, c')\big)\right)$$

au sens où, pour chaque option, la correspondance de $patt$ à c devient une non-correspondance de $patt$ à c (d'où le recours à la recherche d'une transformation adéquate) et l'identification d'une transformation τ d'un Contexte c en un Contexte c' tel qu'il y ait correspondance de $patt$ à c'.

Les transformations prises en charge par un fournisseur d'équivalences sont contextifiées et sont partagées par tous les Agents d'un même Hôte . Ceci réduit le coût d'entreposage de ces Contextes, et facilite aussi la mise en place d'une faculté d'oubli, au sens où ces Contextes, acquis dynamiquement, ne sont pas nécessairement pertinents en tout temps. Il est donc possible pour un fournisseur d'équivalences de ne pas conserver les Contextes qui ne semblent plus utilisés, libérant de l'espace sur son Hôte pour d'autres fins.

5.5 Interaction avec capteurs et actuateurs

Le système de mise en correspondance requêtes avec le Contexte disponible, décrit dans §5.4.2, fonctionne tout autant pour la consultation d'un capteur que pour l'action sur un actuateur.

En effet, lors de la consultation d'un capteur, l'interaction se fait entre un requérant, par exemple un Agent du domaine dans l'exercice de sa mission, et un Agent de soutien, capable de consulter le capteur, d'en lire une valeur et de l'exprimer sous forme de Contexte pour fins de publication. Lors de l'expression d'une requête pour un actuateur, l'interaction se fait aussi entre un demandeur, par exemple un Agent du domaine, et un Agent de soutien, capable d'agir sur l'actuateur et de lui faire influencer le monde physique.

Pour la consultation d'un capteur, l'Agent spécialisé en contact avec le capteur lira typiquement les valeurs par sondage, et publiera l'équivalent sous forme de Contexte. Les modalités de publication (à chaque changement de valeur constaté, par exemple, ou à intervalles réguliers) dépendent de l'implémentation de l'Agent spécialisé, des besoins, et des caractéristiques du capteur. Une consultation événementielle est aussi possible, mais à l'aide d'un objet autonome dû à l'aspect asynchrone de la consultation, qui ne s'harmonise pas bien avec le cycle d'opération des Agents.

Pour l'action sur un actuateur, un Agent demandeur publiera un Contexte décrivant sa requête. L'Agent spécialisé interagissant avec l'actuateur soumettra pour sa part un modèle de Contexte décrivant les requêtes qui sont du Contexte pertinent selon lui, et examinera l'ensemble des requêtes correspondantes.

La stratégie de mise en application de ces demandes dépendra de l'Agent, mais tient compte :

- des demandes constatées : est-il possible de toutes les desservir? Sinon, quelle est la stratégie à appliquer? Vise-t-on l'atteinte de la moyenne des demandes? Du mode?
- des paramètres de sécurité de l'appareil, pour éviter des bris ou des blessures;
- d'autres facteurs, comme la préservation de la pile sur un appareil mobile en vue de pouvoir poser des gestes prioritaires comme un arrêt d'urgence;
- etc.

5.6 Interaction entre Hôtes

Dans ContextAA, les Hôtes interagissent entre eux par le biais d'objets autonomes et de canaux de communication pris en charge par le Communicateur, et transigent exclusivement en termes de Contexte. Étant donné $c: C; h, h': H$, nous notons $h \xrightarrow{c} h'$ le transfert de c de h vers h' sans égard aux détails de la mécanique assurant ce transfert.

5.6.1 Déploiement d'un Agent

Le déploiement initial d'un Agent a sur un Hôte h peut se faire de plusieurs manières, incluant par une migration en provenance de $h' \in N_h$et par chargement à partir d'un flux.

ContextAA supporte aussi le déploiement d'Agents du domaine en format binaire, dans les cas où une représentation contextifiée ne s'avèrerait pas réalisable, mais l'objectif de notre démarche est de faire évoluer le langage du Contexte à un point tel que tout Agent du domaine puisse être représenté purement par du Contexte. Dans le cas du déploiement d'un Agent en format binaire, la représentation de déploiement est une bibliothèque à liens dynamiques (p. ex. : `.so`, `.dll`) et la plateforme sous-jacente requise est une contrainte inscrite dans R_a, ce qui permet d'automatiser malgré tout le déploiement et la migration de a.

5.6.1.1 Ressources minimales d'un Agent

En temps normal, R_a devrait tenir compte d'une certaine « zone de confort », et exprimer les requis de a sous forme pessimiste, dans l'optique de permettre à a de réaliser une migration lorsque $support(R_a, h)$ ne lui « suffit » plus; ceci ne constitue pas un requis formel, mais vise à permettre la migration $h \xrightarrow{a} h'$ de a vers un Hôte h'tel que $support(R_a, h')$, noté $h' \in \Xi_a$ (voir §5.6.2 pour une définition plus formelle de Ξ_a) avant que a ne subisse une discontinuité de service.

L'aspect R_a d'un Agent a joue deux rôles :

- en décrivant les ressources minimales que h doit offrir à a pour soutenir l'offre de services de ce dernier, R_a guide le démarrage du processus de migration de a, ce démarrage prenant effet lorsque $support(R_a, h)$ ne lui « suffit » plus;
- du même coup, R_a guide la détermination d'une destination pour a lors d'une migration. L'ensemble Ξ_a est en effet généré à partir des Hôtes $h' \in N_h$ susceptibles de soutenir a dans sa mission.

5.6.2 Migration d'un Agent

Un Agent du domaine a migre d'un Hôte h vers un Hôte h' par sérialisation de sa représentation contextifiée, exprimée $h \xrightarrow{a} h'$ selon la notation décrite dans §5.6. Concrètement :

- soit $\Xi_a \subseteq N_h$ l'ensemble des Hôtes tels que $h' \in \Xi_a \Leftrightarrow h' \in N_h \wedge support(R_a, h')$ « suffit » aux besoins de $a \in h$ étant donné $a = \{M_a, O_a, R_a, S_a\}$;
- si $\Xi_a = \emptyset$ alors il n'y a pas d'Hôte capable d'accueillir a dans N_h. Ceci peut être un obstacle à la continuité de service de a, mais la solution à ce problème dépend du contexte, de M_a, d'O_a, des ressources encore disponibles sur h, etc. Rappelons que bien que la continuité de service soit un objectif fondateur de ContextAA, l'atteinte de cet objectif ne peut être que favorisée, pas garantie;
- si $|\Xi_a| > 1$, alors il existe plus d'un Hôte de destination potentiel pour a. Dans ce cas, l'Hôte $h' \in \Xi_a$ pour lequel $support(R_a, h')$ sera le plus près de 1 sera privilégié. Si $\exists h', h'' \in \Xi_a\big(support(R_a, h') = support(R_a, h'')\big)$, alors a peut offrir un critère secondaire de sélection pour départager h' et h''; en l'absence d'un tel critère, ContextAA sélectionne l'un ou l'autre de ces Hôtes en fonction de critères qui lui sont propres.

Une fois un Hôte de destination h' choisi pour $h \xrightarrow{a} h'$, le gestionnaire d'Agents sur h prend en charge la partie sortante de la migration :

- a est suspendu sur h;
- a est contextifié sur h;
- a est persisté sous sa forme contextifiée sur h, en cas de problème lors de la migration;
- a est sérialisé en direction de h';
- sur réception de la forme contextifiée de a par h', le gestionnaire d'Agents sur h' prend en charge la partie entrante de la migration :
- a est persisté sous sa forme contextifiée sur h', pour fins préventives;
- un Contexte décrivant la réception de a par h' est émis par h' vers h;
- a est réifié sur h' à partir de sa forme contextifiée; puis
- a est activé sur h'.

La forme contextifiée de a persistée sur h sera écrasée par une éventuelle re-migration de a vers h. L'Hôte de destination h' est en droit de renommer a en $a'' : id(a'') \neq id(a)$ sur réception, avant que a (ou a'', s'il y a renommage) ne soit persisté puis réifié sur h'. En effet :

- ContextAA exige que $\forall h, h' : H\big(id(h) \neq id(h')\big)$;
- ContextAA exige aussi que $\forall a, a' : \mathrm{A}, h : H\left(a \in h \wedge a' \in h \Rightarrow id(a) \neq id(a')\right)$;
- par contre, ContextAA n'exige pas que $\forall a, a' : \mathrm{A}; h, h' : H\left(a \in h \wedge a' \in h' \Rightarrow id(a) \neq id(a')\right)$.

Conséquemment, il est possible que sur réception de la forme contextifiée de a par h', h' constate l'existence au préalable d'un Agent $a' \in h'\big(id(a) = id(a')\big)$. L'exigence d'un nom unique pour chaque Agent sur un Hôte donné n'est alors pas rencontrée.

5.6.3 Mécanismes d'isolation

Dans ContextAA, un Agent est isolé du réseau et des problèmes de concurrence par son Hôte et à travers la structure opératoire dans laquelle s'inscrit son cycle de vie.

Cette structure isolante a pour fonction de permettre d'en arriver éventuellement à la rédaction d'Agents du domaine par des experts du domaine. En effet, pour un Agent a, chaque opération est réalisée sur la base d'information locale, S_a, exempte de tout risque de concurrence. Si S_a comprend des requêtes pour du Contexte décrivant N_h, alors cette information est extraite de S_h, toujours sans risque de concurrence, dans la mesure où elle y a été rendue disponible au préalable.

L'isolation qu'assure un Hôte entre ses Agents et le voisinage réseau, et qui fait en sorte qu'un Agent du domaine ne peut distinguer de prime abord le cas où la connectivité de son Hôte est bonne du cas où elle est mauvaise ou inexistante.

La continuité de service de l'Agent est maintenue, dans la mesure du possible, bien que la qualité de service puisse être impactée : à titre d'exemple, puisqu'un Agent a n'a accès qu'au Contexte logé dans son espace contextuel S_a, une connectivité nulle implique que S_a ne pourra être mis à jour que par a lui-même ou par les actions d'autres Agents sur le même Hôte, ce qui peut éventuellement mener a à opérer à partir de Contextes désuets, mais sans mener à une interruption de service, du moins en général.

Les connexions et déconnexions d'Hôtes sont décrites sous forme de Contexte par N_h, qui est en pratique un sous-ensemble de S_h. Ceci découle directement de l'interaction entre objets autonomes responsables de gérer la découverte d'homologues et leurs Agents standards associés, ces derniers tenant à jour N_h dans le respect de leur cycle d'opérations. Ceci implique qu'en tout temps, $\forall a \in h$ a accès à une description *presque à jour* de N_h.

5.6.4 Actions différées

La mention « action différée » dans la figure 15 tient au fait que les requêtes soumises par a dans *publier* sont au mieux desservies lors de la prochaine étape *consommer* de a. Tout Agent dans ContextAA doit être rédigé en tenant pour acquis que ses requêtes ne seront pas desservies avant un ou plusieurs cycles d'opération supplémentaires.

La pratique préconisée pour les Agents de ContextAA est de planifier en fonction du futur mais d'agir sur le présent : un Agent a publie ses besoins sous forme de requêtes pour du Contexte dans S_a, et agit sur la base du contenu dans S_a, or il est possible qu'aucune action n'ait permis de mettre S_a à jour entre la publication des besoins de a et sa prochaine opportunité d'agir.

La mention *presque à jour* portant sur N_h à la fin de §5.6.3 est une considération au cœur des actions des Agents dans ContextAA, au sens où il est tout à fait normal (et commun) qu'un Agent $a \in h$ opère à un instant t sur du Contexte qui n'est techniquement plus valide pour h.

Un exemple d'une telle situation apparaît à la figure 18 :

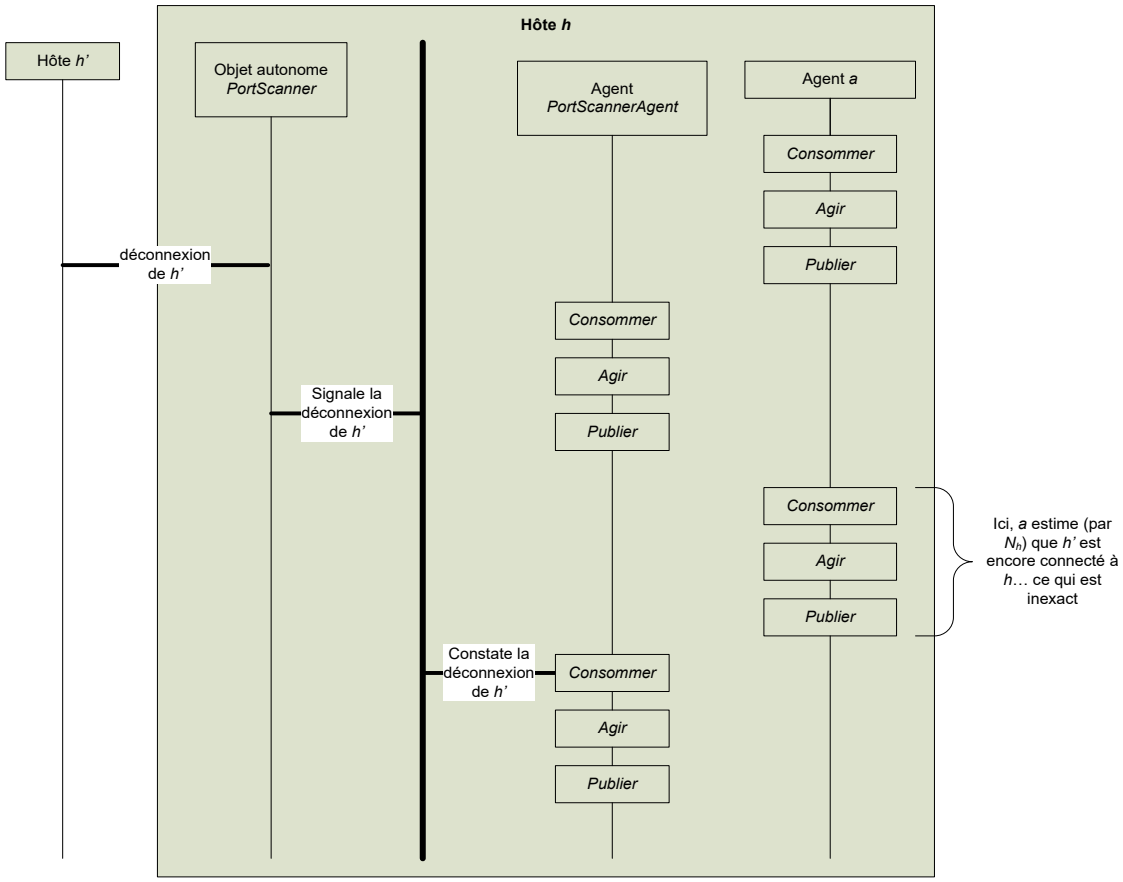


Figure 18 - Non-condordance du Contexte local avec la « réalité physique »

En effet, dans la situation décrite par la figure 18 :

- un objet autonome *PortScanner*, jouant le rôle de Découvreur, constate la déconnexion d'un Hôte h' de son propre Hôte h;
- *PortScanner* envoie une description de cette nouvelle réalité à un Agent, *PortScannerAgent*, à travers un canal de Contexte;
- ultérieurement, *PortScannerAgent* constate la déconnexion de h' et est en mesure de décrire cette réalité dans S_h;
- entre-temps, a a traversé son cycle normal d'opérations et a opéré au moins une itération de ce cycle avec, dans S_h, un descriptif inexact de la situation de h dans l'espace, soit $N_h \subset S_h \wedge h' \in N_h$;
- exprimé autrement, l'état de N_h à l'instant t n'est pas nécessairement isomorphe avec la topologie réseau de h, et a opère strictement sur la base de N_h.et non pas sur celle de la topologie réseau réelle.

Il existe plusieurs modèles de cohérence systémique pour les systèmes répartis, chacun avec ses coûts, ses avantages et ses inconvénients [98]. Un objectif classique de systèmes répartis est ce qu'on pourrait nommer la cohérence systémique globale, soit l'aspiration à un idéal selon lequel il serait raisonnable de s'attendre à ce que l'ensemble des composants d'un système ait une acception commune d'un état partagé donné. Comme le met en relief l'explication ci-dessus, ContextAA ne vise pas la cohérence systémique globale, privilégiant une cohérence interne à chaque Agent a et respectueuse de son cadre ontique. Le raisonnement réalisé par a se fait sur la base du Contexte cumulé dans S_h et dans S_a, dont la correspondance avec ce qui est externe à h est indirecte et différée.

La raison derrière ce choix est à la fois philosophique et pragmatique :

- sur le plan pragmatique, assurer une cohérence systémique globale entraînerait un coût prohibitif, forçant un effort algorithmique assurant que $\forall a, a' : A, \forall h, h' : H, a \in h \wedge a' \in h'$, a et a' soient en accord sur la réponse à toute requête $c : C (c \in O_a \wedge c \in O_{a'})$. Ceci signifie qu'à tout le moins dans le cas où $h \in N_{h'} \wedge h' \in N_h$, aucune divergence dans ce qui aura été rapporté à la fois lors de l'application de c pour a et de c pour a' ne doit être constatée. Ceci rejoint l'approche de plusieurs bases de données NoSQL [98] qui, pour des raisons d'efficacité, visent plutôt une cohérence éventuelle qu'une cohérence systémique globale;
- sur le plan philosophique, une exigence de cohérence systémique globale ne nous semble pas correspondre à la réalité empirique d'un système réparti ouvert. En l'absence de connectivité totale et d'une autorité centrale sur les données, comme dans le cas d'une base de données centralisée par exemple, les données accumulées – le Contexte acquis – par un composant à travers son cheminement sont fortement influencées par ce cheminement, et il est en général difficile de qualifier un cheminement d'objectivement meilleur qu'un autre *a posteriori*.

En résumé, il nous semble plus pertinent, pour un système réparti déployé dans l'espace intelligent ouvert tel que ContextAA, de mettre de côté toute préoccupation de cohérence systémique globale pour insister sur l'importance de la cohérence interne d'un Agent. Cette position tient au fait que tout savoir obtenu de composants dans un système réparti est essentiellement périmé, une photographie du passé en quelque sorte, et que ce constat s'applique aussi aux lectures des horloges relatives des nœuds. Il est possible d'imposer une sorte d'ordonnancement systémique par un mécanisme algorithmique tel que les horloges vectorielles [52], mais ContextAA préconise l'approche, plus légère sur le plan protocolaire, selon laquelle il vaut mieux ne pas chercher à déterminer un tel ordre total.

La stabilité de S_a pendant l'action de a permet d'exprimer *consommer*, *agir* et *publier* pour a de manière « naïve », sans considérer la concurrence et ses risques, donc de raisonner sur le présent, au sens de a. Cette liberté a un prix : opérer sur un Contexte qui peut ne pas être à jour, qui est au mieux le plus récent Contexte obtenu par a. Le monde représenté par S_a est stable aux yeux de a pendant que a agit, sans poser de jugement sur la qualité objective du Contexte logé dans S_a au même moment.

5.6.5 Mobilité et gestion de conflits

La nature décentralisée de l'approche proposée par ContextAA rend presque inévitable l'émergence de conflits, soit de situations où les missions d'au moins deux Agents a, a' mènent à des actions antagonistes de part et d'autre. Ces actions peuvent aider ou nuire aux humains, et une même action peut entraîner des effets positifs pour un individu tout en étant au détriment d'un d'autre individu lorsque leus besoins ne concordent pas.

Ces conflits sont de plusieurs natures. Par exemple :

- les deux Agents publient des Contextes décrivant une même réalité mais de manière non-concordante, comme par exemple si a décrit un individu comme étant agité et a' décrit le même individu comme étant calme; ou encore
- les deux Agents publient des demandes conflictuelles pour une même action, soit une demande par a d'ajuster un thermostat à une température de 20°C et une demande par a' d'ajuster le même thermostat à une température de 23°C.

Puisque les états des espaces intelligents sont susceptibles d'influencer le bien-être d'êtres humains ou l'intégrité d'appareils participant à la réalité enrichie, une résolution simpliste de conflit (p. ex. : traiter les demandes dans l'ordre où elles ont été constatées, en supposant – naïvement – l'absence de demandes simultanées), bien que possible, n'est pas nécessairement adéquate.

Il n'est toutefois pas possible de déterminer une stratégie universelle de résolution de conflits, le contexte jouant un rôle trop important dans ces situations. Il demeure malgré tout possible de dégager quelques considérations d'ordre général :

- chaque Agent est présumé autonome et responsable de ses actions. Pour cette raison, l'Agent doit traiter les demandes d'action en fonction de son cadre ontique et de sa mission;
- selon cette optique de base, un Agent placé face à deux descriptions antagonistes d'un même Contexte peut, suivant une évaluation subjective de la qualité du Contexte, choisir de privilégier la vision de l'un des Agents, de l'autre, ou même choisir une position mitoyenne. Dans un cas où il n'est pas possible de discriminer entre deux propositions, l'Agent consommant les Contextes peut même n'en choisir aucun, faute de pouvoir faire un choix éclairé;
- placé face à une demande d'action, un Agent procède aussi de manière subjective. En plus des possibilités mentionnées ci-dessus, l'Agent respectera les limites du matériel avec lequel il interagit, évitant de l'amener hors de ses paramètres d'utilisation sécuritaire. Si le contexte s'y prête, l'Agent consommateur peut même s'enquérir du Profil des individus dans son voisinage réseau pour voir s'il est possible d'aménager une action se situant à mi-chemin entre les diverses demandes reçues, tout en respectant les bornes acceptables pour tous les humains impliqués;
- peu importe le chemin choisi, l'Agent agissant par voie d'actuateurs doit prendre soin de ne pas outrepasser les paramètres acceptables pour les individus impliqués, et doit éviter tout bris matériel. S'il survient une situation où il n'est pas possible pour l'Agent de rencontrer ces exigences minimales, alors un mécanisme d'urgence doit être utilisé pour signaler ce constat d'impossibilité à une personne en position d'autorité.

ContextAA ne prend pas position sur les pratiques de résolution de conflits privilégiées par un Agent. Son rôle est d'ordre structurel : ContextAA permet l'expression des souhaits des demandeurs et permet la mise en place de stratégies de résolution de conflits résultant de souhaits concordants ou non. La responsabilité quant à la résolution éventuelle des conflits est, dans ContextAA, du ressort des expert(e)s du domaine.

En ceci, ContextAA se distingue d'applications au rôle plus spécifique, par exemple une application dont le rôle serait précisément d'assurer le maintien d'une température ambiante convenant aux besoins d'individus fiévreux, et où les Agents opéreraient nécessairement dans ce but. Si une application développée sur la base de ContextAA vise un tel objectif, il importera que l'Agent accédant directement à l'actuateur de climatisation, et assurant une forme de médiation entre les requêtes d'ajustement de température ambiante reçues de divers Agents tiers, soit tel que son cadre ontique (§4.8) tienne compte de cet objectif. Dans des circonstances analogues, un Agent ayant un objectif différent pourrait trancher entre les mêmes demandes selon d'autrs critères, et en arriver ainsi à un résultat différent.

5.7 Temps

Le temps dans ContextAA est une préoccupation multifacettes. Composant important des systèmes contextualisés en général, il intervient dans ContextAA à plusieurs niveaux, que ce soit pour le suivi de l'exécution d'un Hôte, pour la discrétisation du temps pour un Agent, ou pour l'ordonnancement des événements dans l'espace ouvert.

Dans ContextAA, une caractéristique du temps est qu'il est contextifié, décrit par du Contexte, et donc assujetti aux mêmes contraintes que l'ensemble des Contextes. En particulier, les considérations soulevées dans §5.6.5 s'appliquent pleinement au temps : aucune présomption n'est faite dans ContextAA quant à un temps global en tant qu'état partagé envers lequel tous les Agents se seraient mis d'accord. En ceci, ContextAA prend acte des constats faits par Sheehy [102] plutôt que de suivre la proposition mise de l'avant par Weiss et al. [65] à l'effet que l'IdO dépendrait d'horloges synchronisées à très haute précision.

La philosophie qui sous-tend ContextAA est à l'effet qu'il est contre-productif de chercher un état global dans un système réparti ouvert. Une plus forte synchronisation entre les horloges des Agents de h et de N_h y est vue comme un bonus, pas comme une nécessité, du fait que tout Agent doit être pensé au préalable pour tenir compte du fait que les horloges des Agents sur divers Hôtes peuvent diverger.

5.7.1 Moments d'un Agent

Dans la figure 15, la mention « … » apparaissant entre les opérations *publier* et *consommer* d'un Agent a représente le « moment » entre l'action de publier et l'action de consommer pour a. Pendant ce temps, au sens de a, le temps passe et des événements se produisent, et les conséquences de ces événements ne seront visibles pour a que lors de sa propre prochaine étape *consommer*.

L'exécution itérative d'un Agent a entraîne une discrétisation du temps pour a : le temps passe pour a: h de manière discrète, une itération à la fois dans le cycle d'opération de h. Ce que décrit « … » dans la figure 15 est l'ensemble des événements externes aux actions de a qui se produisent entre l'étape *publier* et l'étape *consommer* de l'opération de a, ce qui inclut les actions de $\forall a' \in h; a \neq a'$, de même que toute action prise par un Agent sur un hôte $h' \neq h$.

Nous nommons « moment de a » pour $a \in \mathrm{A}, a \in h$ une itération de a dans le cycle d'exécution de h. Les moments de a sont dénombrables et peuvent être annotés de manière telle qu'étant donné deux moments $t_i, t_j; i, j \in \mathbb{Z}$, nous avons $t_i < t_j \Leftrightarrow i < j$ pour représenter t_i précède t_j.

Deux moments t_i, t_j sont dits consécutifs si $\nexists t_k (t_i < t_k < t_j); i, j, k \in \mathbb{Z}$, ce qui équivaut en pratique au cas où $j = i + 1$. Par cette définition, l'annotation des moments s'incrémente de manière monotone avec chaque moment d'un même Agent, ce qui permet de construire un ordonnancement des moments [62] pour un Agent donné sur un Hôte donné.

Entre deux moments consécutifs de a, S_a est susceptible de changer de plusieurs manières, que ce soit par l'ajout de nouveaux Contextes (réponses à des requêtes soumises antérieurement, par exemple) ou par la suppression de Contextes périmés en accord avec les règles décrites dans O_a.

Il existe une dichotomie entre le temps qu'on pourrait qualifier de « physique », celui qui semble progresser de manière continue sans égard aux actions d'un Agent donné, et le temps au sens d'un Agent, qui progresse par sauts discrets tels que le « monde » perçu par un Agent (l'état de S_a ou de S_h, selon l'Agent).

Dans ContextAA, pendant qu'un Agent opère, son « monde » ne change pas si lui-même ne le change pas, puisque le gestionnaire d'Agents d'un Hôte opère sur chaque Agent en séquence et puisque les actions asynchrones ou parallèles au cycle normal d'opération d'un Hôte sont réalisées par des objets autonomes, et puisque ces actions ne prennent effet que lorsque les Agents standards correspondants en consomment les fruits par des canaux de Contexte pour les intégrer à S_h. Aucun événement asynchrone ne perturbe S_a pendant l'action de a.

Si cela s'avère nécessaire, il est possible de construire une histoire de a [17] à partir d'instantanés successifs de S_a, mais il est probable que cette histoire ne coïncide pas avec celle de $\forall a' \in h(a' \neq a)$ en raison des différences entre O_a et $O_{a'}$. En effet, ce qui constitue du Contexte dépendant de l'Agent, cette réalité distingue notre approche, qui utilise un cadre ontique par Agent et où chaque Agent n'observe qu'une partie des Contextes possibles, d'une approche avec Ontologie partagée où il est raisonnable de présumer que deux Agents auraient eu la même perspective sur le Contexte disponible.

5.7.2 Ordonnancement chronologique global

Le problème de l'ordonnancement des événements dans un système réparti a été adressé par plusieurs, dont Lamport [62] avec la relation *Happens-Before* et Chandy et Lamport [17] avec la constitution d'instantanés valides des états systémiques. Ces résultats sont toutefois soumis à des contraintes sévères, incluant celles d'un système fermé et de moyens de communication fiables entre les processus. La plupart des langages de programmation contemporains, par exemple C++ [51] décrivent des relations telles que *Happens-Before*, *Sequenced-Before*, *Inter-Thread Happens-Before*, etc. mais dans le contexte d'un seul et même programme lorsque celui-ci s'exécute sur une machine multi-cœurs.

ContextAA est conçu pour l'espace intelligent ouvert, dans lequel il n'est pas garanti que ces contraintes soient rencontrées. Pour cette raison, l'optique préconisée y est différente : les outils mis à la disposition des Agents favorisent le développement de perspectives subjectives sur le Contexte, et il n'est en général pas requis qu'il existe un ordonnancement global sur les événements.

Favoriser une perspective subjective sur le Contexte va de pair avec l'approche de ContextAA qui met de l'avant le Contexte-selon : les besoins, les règles de raisonnement, les traitements applicables au Contexte font tous, dans ContextAA, partie du cadre ontique d'un Agent, ou selon l'Agent. Ceci mène à une vision minimaliste du Contexte, ne visant à couvrir que les besoins des Agents véritablement déployés sur un Hôte donné. Conséquemment, un Agent ne requiert habituellement un ordonnancement global de certains événements que lorsque cet ordonnancement a des conséquences sur sa mission, ou impacte son cadre ontique.

5.7.3 Ordonnancement chronologique local

À l'intérieur d'un même Hôte h, chaque Agent $a \in h$ exécute tour à tour son cycle normal d'opération, ce qui induit une progression monotone et discrète des opérations pour a. Un Agent standard sur chaque Hôte est responsable de tenir à jour un compteur d'itérations du gestionnaire d'Agents de cet Hôte. Il est donc possible pour un Agent d'établir une chronologie des événements pour lui.

Au sens de l'ordonnancement des opérations entre Agents sur un même Hôte h, les actions d'un Agent du domaine apparaissent simultanées au sens de Lamport [62], c'est-à-dire qu'il n'est pas possible pour un Agent du domaine d'observer un ordonnancement entre ses actions et celles des autres Agents du domaine du même Hôte. Ainsi, pour un Agent du domaine $a \in h$, les opérations s'expriment au présent, en termes du contenu de S_a.

Les actions des Agents standards, pour leur part, suivent une séquence déterminée par le gestionnaire d'Agents de leur Hôte. Du point de vue des Agents du domaine, ces actions semblent faites de manière asynchrone, leurs effets ne pouvant être constatés qu'une fois qu'elles ont été complétées. Ainsi, au sens mis de l'avant par Herlihy et Wing [43], ces opérations sont linéarisables.

5.7.4 Ordonnancement chronologique entre Hôtes

Bien que la détermination d'un ordonnancement global d'événements entre Hôtes distincts dans l'espace ouvert semble hors de portée de manière générale, il est possible de tracer un ordonnancement entre Hôtes pour certains événements clés. Soit $h, h' : H$:

- lors de $h \xrightarrow{c} h'$, l'envoi de c par h précède globalement la réception de c par h';
- en particulier, lors de $h \xrightarrow{a} h'$, le début de la migration de a à partir de h précède son arrivée sur h';
- toutefois, placé face à $c, c' \in C$ tels que c est produit par $a \in h$ à un moment t selon a et c' est produit par $a' \in h'$ à un moment t' selon a', il n'est pas nécessairement possible de déterminer une chronologie entre t et t', donc d'ordonnancer c et c' sur la base de leur moment de production.

En pratique, pour un Agent a'' observateur de c et c', cet indéterminisme peut ne pas être problématique :

- si a'' suppose que les horloges sur lesquelles sont basés t et t' sont suffisamment près d'être synchronisées pour être considérées équivalentes en pratique, alors c précèdera c' au sens de a'' si $t < t'$;

- si a'' est en mesure d'estimer la différence entre les horloges sur lesquelles sont basés t et t', alors il est possible pour a'' d'appliquer une correction sur t et t' pour en arriver à établir un ordonnancement entre eux, ce qui peut se faire à la manière de ce que font des protocoles tels que NTP [68]. Dans ce cas, un Agent standard sur h peut tenir à jour les écarts apparents d'horloges $\forall h' \in N_h$ de manière à ce que $\forall a \in h$ ait accès localement au fruit de ce calcul des écarts.

La qualité de ces estimations peut être réduite si les horloges sont irrégulières, non-monotones, ou encore si les Contextes consultés n'ont pas été produits récemment, accroissant les risques que les estimés des différences entre les temps de production soient inadéquats. Conséquemment, il est préférable pour a'' de ne pas faire reposer son estimé de qualité relative entre deux Contextes sur la seule base de leur « fraîcheur », donc du caractère récent ou non de leur moment de production.

5.8 Modèle de programmation

Le modèle de programmation mis de l'avant par ContextAA se présente en deux volets, soit un volet interne et un volet externe. Le volet interne est surtout ouvert aux Agents standards et aux Agents du domaine rédigés dans un langage de programmation convenant à l'Hôte sur lequel l'Agent est déployé et à la plateforme sous-jacente. Le volet externe, lui, est pleinement contextifié. Dans les deux cas, l'objet sur lequel portent les opérations est le Contexte.

Le fait qu'un Hôte isole les Agents du réseau et des problèmes de concurrence permet d'exprimer les algorithmes sur du Contexte dans ContextAA de manière à ne se concentrer que sur le cœur du propos, sans égard aux problèmes qui pourraient survenir lors de modifications concurrentes d'un espace contextuel. Les particularités complexes de la réseautique et de la concurrence étant évacuées d'office, il devient possible pour des spécialistes d'autres domaines que l'informatique, mais ayant tout de même des connaissances de base en programmation, d'exprimer leur pensée de manière algorithmique, un atout significatif dans un créneau multidisciplinaire.

Le modèle interne est inspiré de l'approche mise de l'avant dans la bibliothèque STL et formalisée dans [106] : là où STL combine itérateurs, algorithmes et fonctions (en particulier, des prédicats), ContextAA combine Contexte, algorithmes et opérations, en particulier, des critères.

Lorsque l'action d'un Agent est exprimée par programmation conventionnelle plutôt que par du Contexte, il est d'usage de respecter certaines règles de bienséance dans la programmation de cet Agent. En particulier, il importe de ne pas lancer de tâches s'exécutant de manière concurrente, tout comme il importe de n'entreprendre aucun calcul dont la durée maximale ne serait pas bornée au préalable. Ceci s'applique à chaque opération d'un moment dans l'exécution d'un Agent.

Le modèle externe est proche de ce que proposent les langages fonctionnels, mais exprimé sous forme de Contexte [2], donc contextifiées. Il est décrit dans §4.4.7.

5.9 Spécifications techniques de ContextAA

Sur le plan technique, ContextAA est constitué de plus de 17 000 lignes de code C+ +17 réparties sur plus de 275 fichiers, et n'utilise aucune bibliothèque tierce outre celles des plateformes sous-jacentes (p. ex. : communication par voie de *sockets*) et la bibliothèque standard du langage en tant que telle. L'auteur a rédigé la totalité de ContextAA lui-même.

Les sources de ContextAA se veulent pleinement portables et compilent sur les trois principaux compilateurs pour ce langage, soit Cl.exe (Microsoft), Clang et g++.

Chargé de tous ses Agents standards et d'une dizaine d'Agents du domaine, un composant Hôte de ContextAA complet occupe durant son exécution à pleine plus d'un Mo en mémoire vive, un coût nettement inférieur à celui de l'exécution d'un programme de type « Hello World » dans la plupart des langages de programmation les plus connus. Plus d'informations sur la consommation de mémoire vive par ContextAA en fonction du nombre d'Agents du domaine sur un Hôte donné sont offertes à la section §6.1.

Chapitre 6
Validation

« On n'achète qu'au prix du travail un heureux succès »
(Sophocle / Électre - V^e s. av. J.-C.)

ContextAA vise à permettre à des Agents autonomes d'accomplir leur mission, seuls ou en interaction avec d'autres Agents, même dans un environnement intelligent hétérogène. À partir de ce que nous nommons espace intelligent ouvert, les éléments constitutifs de ContextAA ont été conçus à la fois pour profiter des espaces intelligents traditionnels et pour ouvrir la possibilité d'une contextualisation pour individus et Agents pleinement mobiles.

Dans une optique de validation, évidemment, il n'est pas possible de tester l'ensemble des configurations possibles de l'espace intelligent ouvert; dans les mots de Dijkstra [31], « *program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence* ». La complexité d'un système tel que ContextAA ne nous permet pas non plus d'offrir une preuve formelle de son fonctionnement, du moins à ce stade de nos travaux.

Nous sommes toutefois en mesure de valider certains volets de ContextAA de manière empirique : par exemple, la validation des fonctions de distance structurelle et de contenu sur la base d'un ensemble de Contextes étalons, ou encore la réalisation de quelques scénarios choisis, pour démontrer que ContextAA est en mesure de rencontrer les situations pour lesquelles il a été conçu.

En particulier, deux scénarios sont décrits ici, soit celui d'un ajustement des paramètres de confort d'un individu par voie d'interaction entre Agents et celui d'un effort de maintien de la continuité de service pour un Agent par voie de migration. Ces scénarios demeureront simples pour les besoins de cette validation; toutefois, nous discuterons, dans la conclusion de ce document, de quelques avenues de recherche ouvertes par son éventuelle extension.

6.1 Frugalité

Tel que déclaré d'entrée de jeu (§1.4), ContextAA a été pensé dès le tout début en vue d'une certaine frugalité, au sens où sa consommation de ressources doit être suffisamment limitée pour lui permettre d'opérer sur des appareils où ces ressources sont limitées.

Cet objectif est en partie rencontré : un Hôte par défaut, limité aux seuls Agents standards et aux objets autonomes permettant son fonctionnement et son interaction avec le monde extérieur à travers TCP/IP, montre une consommation initiale de mémoire vive d'un peu moins de 1,5 Mo. Cette configuration de base, au moment des tests, inclut dix Agents standards et quatre Agents du domaine, de même que cinq objets autonomes sur un Hôte donné, et permet d'assurer la communication TCP/IP, l'échange de Contextes entre Hôtes, la migration d'Agents du domaine au besoin, et assure le cycle d'exécution normal de l'Hôte et de tous ses Agents. Inclure quelques Agents du domaine supplémentaires pour fins de test n'impacte pas visiblement cette situation, dans la mesure où chacun de ces Agents soumet quelques requêtes et raisonne sur les résultats ce ces requêtes lorsque ceux-ci apparaissent dans son espace contextuel.

Il est à noter toutefois qu'il n'y a pas de limite théorique à la quantité de Contexte que peut produire ou demander un Agent, et donc qu'il est difficile de baliser les coûts en termes de ressources qu'encourt un Agent du domaine lorsqu'il est déployé sur un Hôte; pour cette raison, nos tests se sont limités à des Agents dont l'action se limitait à :

- soumettre des requêtes pour du Contexte (p. ex. : état de la pile de l'appareil, température ambiante exprimée en degrés Celsius);
- produire des méta-Contextes pour la gestion des Contextes périmés de son espace contextuel;
- réaliser des calculs simples (p. ex. : évaluer si une valeur se situe sous un seuil de tolérance donné); et
- poser des actions simples (p. ex. : publier un Contexte représentant une demande de migration ou un indice de confort relatif).

La croissance de la consommation de mémoire vive en fonction du nombre d'Agents du domaine se limitant à ce niveau de complexité sur un Hôte est reflétée dans la figure 19 ci-dessous :

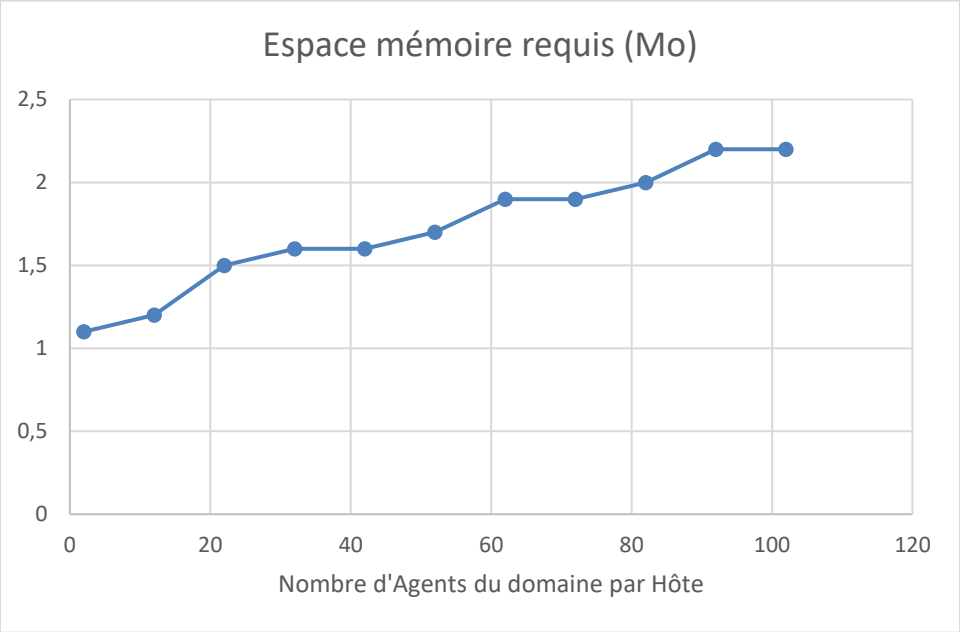


Figure 19 Croissance de la consommation de mémoire vive en fonction du nombre d'Agents du domaine

Cette consommation varie en fonction du temps, et tend à croître en fonction des Contextes produits, consommés, des Contextes disponibles dans le voisinage réseau en fonction des requêtes produites, etc. Comme mentionné plus haut, les Agents du domaine dans ce test font une tâche relativement simple, et l'accroissement de la consommation de mémoire est un reflet de la taille de leurs espaces contextuels respectifs.

Le véritable enjeu pour ContextAA est la capacité d'oublier le Contexte désuet, en particulier celui utilisé pour publier des requêtes destinées à des Hôtes du voisinage réseau. Cette partie de la mécanique est sous contrôle, mais demande à être simplifiée pour éviter qu'un Agent écrit de manière trop naïve de mène à une consommation excessive de mémoire pour son espace contextuel.

La consommation de temps sur le processeur exécutant le gestionnaire d'Agents, au moment d'écrire ces lignes, se situe à ≈16% avec cette configuration de base. La principale raison pour cette consommation semble être la paire faite du *ResolverAgent* et du *ContextManagerAgent*, qui opèrent sur une masse importante de Contexte à chaque itération de leur cycle de vie; nos travaux d'optimisation se poursuivent dans les deux cas.

Les objets autonomes associés à des ressources réseaux (communicateur, émetters, récepteurs, détecteurs de voisinage réseau, etc.) montrent une consommation de mémoire négligeable en moyenne, mais avec variations en fonction du trafic vers leur Hôte et à partir de ce dernier; cette variation n'a pas été préoccupante lors des tests menés jusqu'à ce jour. Les autres objets autonomes montrent tous une très faible consommation de ressources, qu'il s'agisse du temps d'exécution sur un processeur ou de la mémoire vive.

6.2 Distance sémantique

ContextAA met en place des mécanismes pour évaluer la distance (et, par le fait-même, la proximité) entre deux Contextes. Ceci se veut utile à plusieurs fins, incluant celle de permettre aux Agents de comparer les Contextes mis à leur disposition avec ceux auxquels ils sont intéressés, et évaluer quantitativement la continuité de service offerte en comparant la représentation sous forme de Contexte des objectifs atteints avec celle des objectifs visés.

Tel que rapporté dans [94], nous avons mené une évaluation des fonctions utilisées par ContextAA pour quantifier la distance sémantique sur la base d'un échantillon de Contextes de similitude apparente à divers degrés, du moins sur la base d'un regard posé par un observateur humain. Ces Contextes ont été construits pour fins de tests, et ne représentent pas des jeux d'essais associés à une situation spécifique; l'objectif visé par les tests reposant sur ces Contextes était d'obtenir une appréciation de la qualité des mécanismes en fonction des besoins qu'ils sont supposés couvrir.

L'échantillon identifié dans cette étude fut :

Table 3 - Échantillon de test, évaluation de la distance sémantique

c_0	host0:agent0:1 { public { temperature { value { 10 } unit { celsius } precision { 0.001 } } source { agent0 } } }
c_1	host0:agent1:1 { public { temperature { value { 10 } } source { agent1 unit { celsius } precision { 0.01 } } } }
c_2	host0:agent2:1 { public { temperature { value { 50 } unit { fahrenheit } precision { 0.001 } } source { agent2 } } }
c_3	h:a:0 { value { 10 } }
c_4	h:a:1 { value { 50 } }
c_5	h:a:2 { value { 10 } unit { celsius } }
c_6	h:a:3 { value { 50 } unit { fahrenheit } }
c_7	h:a:4 { value { 10 } unit { celsius } precision { 0.001 } }
c_8	host0:agent2:2 { public { temperature { value { 10 } src { unit { celsius } precision { 0.001 } agent2 } } } }

c_9	host0:agent3:1 { public { temperature { value { 10 } unit { celsius } precision { 0.001 } } source { agent3 unit { celsius } precision { 0.001 } } } }
c_{10}	host0:agent4:1 { public { temperature { value { 10 } } source { agent4 unit { celsius } precision { 0.001 } } } }
c_{11}	host0:agent0:2 { public { temperature { value { 100 } unit { fahrenheit } precision { 0.1 } } source { agent0 } } }
c_{12}	host0:agent0:3 { public { humidity { value { 100 } unit { kpa } precision { 0.1 } } source { agent0 } } }
c_{13}	a
c_{14}	b { c }

Un ensemble de tests plus riche que celui du tableau 3 pour les fonctions $\sigma(c, c')$ et $\varrho(c, c')$ est disponible aux annexes G et H respectivement.

6.2.1 Distance structurelle

La distance structurelle $\sigma(c, c')$, décrite dans §4.6.2, est une composante du calcul de la distance sémantique telle que supportée par ContextAA. Cette fonction commutative accepte en paramètre deux Contextes c et c', et a pour image un nombre rationnel dans l'intervalle $[0..1]$ où $\sigma(c, c') = 0$ signifie que ces deux Contextes sont structurellement identiques, alors que $\sigma(c, c') = 1$ signifie qu'ils ne montrent pas de ressemblance sur le plan structurel.

Nous avons validé cette fonctionnalité à partir d'un ensemble de paires de Contextes, allant de Contextes identiques à Contextes pleinement dissemblables. Les fruits de ces tests apparaissent à l'annexe G où ils sont groupés en fonction de la valeur résultant du calcul.

Ces tests mettent entre autres en relief l'impact de la taille de $value(c)$ et de $value(c')$ sur le résultat du calcul de $\sigma(c, c')$ pour des Contextes de contenu *a priori* semblable. L'impact relatif du nom des Contextes comparés pour une profondeur donnée peut aussi être constatée à la lueur des calculs.

Pour l'échantillon plus restreint donné dans le tableau 3, un exemple d'une paire de Contextes structurellement rapprochés serait c_0 et c_{11}, du fait que $\sigma(c_0, c_{11}) = 0{,}117284$.

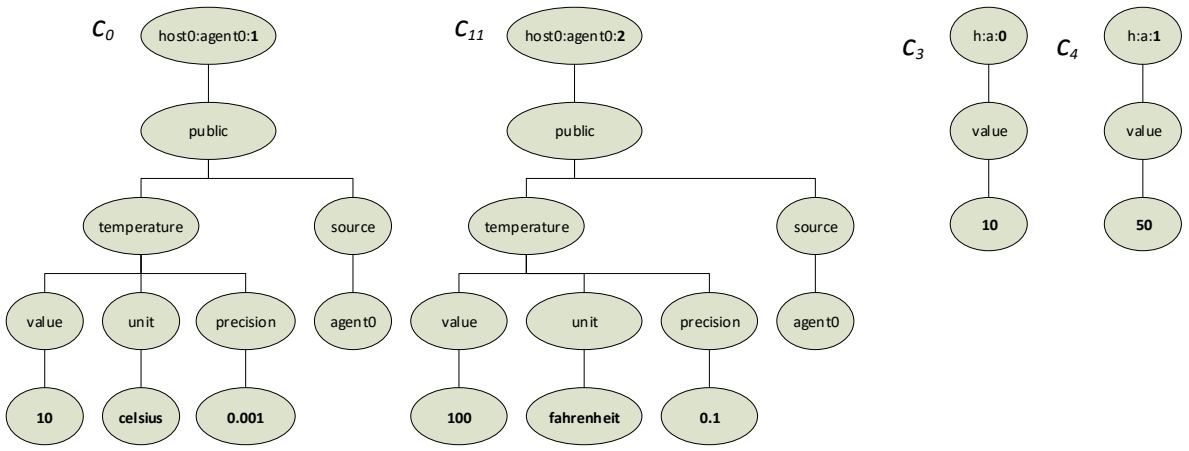


Figure 20 - Similitudes entre c_0 et c_{11}, de même qu'entre c_3 et c_4

La valeur de $\sigma(c_0, c_{11})$ dans la figure 20 se rapprocherait de 0 si les noms de ces deux Contextes étaient identiques, car la distance de $\frac{1}{3}$ entre ces noms influence le calcul de manière importante. En pratique, il est possible qu'un Agent à la recherche de c_0 puisse poursuivre sa mission en obtenant c_{11}.

Un exemple d'une paire Contextes similaires à $\approx 75\%$ serait celle faite de c_3 et c_4, car $\sigma(c_3, c_4) = 0{,}22222$. Ces deux Contextes sont en effet similaires dans leur structure, mais sont en contrepartie différents en termes de contenu. L'importance relative de ces différences pour un Agent a dépend de O_a.

Bien entendu, il existe des Contextes pleinement distincts l'un de l'autre sur le plan structurel; à titre d'exemple, $\sigma(c_6, c_{12}) = 1$.

6.2.2 Distance de contenu

La distance de contenu $\varrho(c, c')$, est une composante du calcul de la distance sémantique (§4.6.1) telle que supportée par ContextAA. À titre de rappel, cette fonction non-commutative accepte en paramètre deux Contextes c et c', et a pour image un nombre rationnel dans l'intervalle $[0..1]$ où $\varrho(c, c') = 0$ signifie que c' est inclus dans c, et où plus la valeur de $\varrho(c, c')$ se rapproche de 1 et moins grande est la part de c' qui est incluse dans c.

Nous avons validé cette fonctionnalité à partir d'un ensemble de paires de Contextes, allant de Contextes identiques à Contextes pleinement dissemblables. Les fruits de ces tests apparaissent à l'annexe H, où ils sont groupés en fonction de la valeur résultant du calcul.

Ces tests mettent entre autres en relief l'impact de noms composites en partie semblables sur le calcul de $\varrho(c, c')$. L'impact relatif du nom des Contextes comparés pour une profondeur donnée peut aussi être constatée à la lueur des calculs.

Les résultats permettent aussi de mieux comprendre l'importance pour un Agent de filtrer les Contextes divergents sur la base de ce qui est représenté avant d'évaluer une distance de contenu, pour éviter de considérer proches deux Contextes représentant des réalités distinctes (par exemple, une mesure de masse et une mesure de chaleur) mais avec suffisamment de sous-Contextes semblables pour entraîner une forme de confusion.

Enfin, ces tests ont été faits en l'absence de changements de référentiels, ce qui explique qu'un cas comme celui de deux Contextes décrivant une même réalité mais avec une unité de mesure différente résulte en une valeur de $\varrho(c, c')$ qui soit non-nulle.

À partir des Contextes listés dans le tableau 3, il est possible de mettre en relief les différences de comportement entre ϱ et σ lorsque celles-ci sont appliquées sur les mêmes paires de Contextes. Par exemple, $\sigma(c_0, c_{11}) = 0{,}117284$ alors que $\varrho(c_0, c_{11}) = \varrho(c_{11}, c_0) = 0{,}19791$. Ainsi, c_0 et c_{11} demeurent semblables sur la base de σ et de ϱ, mais le poids relatif de leur similitude structurelle a un impact plus grand avec σ. En retour, avec c_3 et c_4, $\sigma(c_3, c_4) = 0{,}22222$ alors que $\varrho(c_3, c_4) = \varrho(c_4, c_3) = 0{,}416667$, ce qui met en valeur que leur structure soit plus semblable que ne l'est leur contenu.

6.2.3 Autres constats empiriques

Appliquer les fonctions ρ, σ et ϱ sur un éventail plus large de Contextes permet de jauger l'utilité de ces métriques. Évidemment, comme prévu, la distance entre un Contexte et lui-même est 0 pour les trois fonctions.

En moyenne, ϱ tend à demander un effort de calcul un peu plus important que σ, ce qui est un reflet des calculs impliqués dans chacune des deux formules. Concrètement, supposant la fonction suivante :

$$time :: F \to \Delta t$$

permettant de calculer le temps d'exécution d'une fonction $f : F$, nos tests sur les Contextes répertoriés aux annexes G et H ont montré $\frac{time(\sigma(c,c'))}{time(\varrho(c,c'))} \cong 0{,}88$ pour 11881 appels distincts des fonctions σ et ϱ (109 Contextes comparés avec 109 Contextes), étant donné notre implémentation actuelle de σ et ϱ, les deux étant reproduites à l'annexe M. Ceci n'est bien sûr qu'indicatif, étant influencé par le substrat matériel et la qualité de l'implémentation; il est possible d'envisager une implémentation matérielle de telles fonctions, par exemple, qui influencerait ces résultats.

Sans surprises, $time(\rho(c, c')) \cong \left(time(\sigma(c, c')) + time(\varrho(c, c'))\right)$, mais il est probable que $time(\rho(c, c')) < \left(time(\sigma(c, c')) + time(\varrho(c, c'))\right)$ s'avère en pratique lorsque les implémentations de ContextAA gagneront en maturité et seront plus à même d'éviter d'évaluer de manière redondante les éléments communs à ϱ et à σ. Il est raisonnable de présumer que les variations de Contexte n'influencent pas le sens de cette relation, qui présume en fait qu'il soit possible de profiter dans ρ des parties communes de ϱ et de σ qui seraient évaluées pour chacune individuellement si ces deux fonctions étaient évaluées séparément, réduisant d'autant la quantité de calcul requise pour ρ.

L'enjeu devient donc à la fois de faire profiter au maximum chaque Agent de cette quantification de la distance entre les Contextes, et de faciliter la paramétrisation de $\rho(c, c')$ pour permettre une meilleure évolution de ContextAA alors que de nouveaux besoins et de nouveaux critères de qualification du Contexte seront définis.

6.3 Scénarios de tests

ContextAA a aussi été mis à l'essai dans quelques situations, avec plusieurs autres en cours de planification. Un survol de ces expériences, incluant l'ensemble des préoccupations au cœur de l'action d'un Agent $a \in h$ dans ContextAA, suit.

6.3.1 Scénario – Ajustement des paramètres de confort

Le premier tient au maintien des paramètres environnementaux accessibles à un Agent a à l'intérieur de bornes respectueuses du confort d'un individu. Les ressources disponibles sur l'Hôte ne sont pas un enjeu pour ce scénario. Plus en détail :

- soit un Agent a tel que O_a contient des demandes pour du Contexte exprimant température ambiante;
- soit un Agent de soutien a' faisant le pont entre une paire {actuateur, capteur} de température les requêtes pour du Contexte proposées par a dans S_a expriment cette réalité;
- à chacune de ses étapes d'action, a' réalise une lecture sur le capteur et publie le fruit de cette lecture sous forme de Contexte;
- quand a constate une température hors d'un intervalle défini dans O_a, il émet une demande d'actuation destinée à a' pour ramener ce paramètre environnemental à l'intérieur de l'intervalle en question;
- quand a' constate une demande d'actuation pour ce paramètre environnemental, il interface avec l'actuateur physique pour demander l'ajustement.

L'Agent requérant pour les besoins du test a la forme décrite à la figure 21 (page suivante).

La requête soumise par a' prend quant à elle la forme $**\{temperature\_update\{value\{?=val\}\}\}$. Dans son rôle d'Agent de soutien, a' détermine s'il y a lieu d'agir sur l'actuateur, et si oui, de quelle manière. Rappelons qu'en raison du Contexte-selon, une demande de a n'entraîne pas automatiquement une action de la part de a', ce dernier étant responsable de soupeser les demandes reçues et de déterminer les actions à prendre, le cas échéant.

```
conforteur {
  "human readable name" { conforteur }
  mission {
     eval { "temp_val <= 18 && temp_val <= 23" }
  }
  "ontic frame" {
    wants {
      ** { temperature {
        and { #0 { value { ?=temp_val } }
            { #1 { unit { Celsius }}
      } }
    }
    produces {
       temperature_update{value{eval{"sought_val"}}}
    }
    does {
      eval{
        "sought_val = temp_val < 18? 18 : 23 < temp_val? 23 : temp_val; commit"
      }
    }
  }
  restriction {
  }
  space {
  }
}
```

Figure 21 - Agent requérant (maintien du confort)

6.3.2 Scénario -- Continuité de service par voie de migration

Le second scénario présenté ici porte sur le maintien de la continuité des services offerts par un Agent a alors que les ressources de la plateforme sous-jacente à son Hôte fluctuent :

- un Agent a dit « migrateur » a, parmi les règles décrites par R_a, le mandat de quitter l'Hôte h sur lequel il se trouve si la pile du nœud y devient trop faible selon lui. Les règles décrivant « pile trop faible » sont décrites dans O_a (pour ce qui est de constater activement le problème) et dans R_a (pour automatiser la recherche par h d'un Hôte de destination $h' \in N_h$ adéquat);
- les requêtes pour du Contexte proposées par a dans S_a expriment cette réalité;
- éventuellement, la pile de h devient suffisamment faible pour que h ne satisfasse plus R_a. Le Contexte décrivant la pile de h est produit par le biais d'un Agent standard consultant l'état de l'appareil sous-jacent et l'exprimant sous forme de Contexte;
- a constate ce problème et produit un Contexte exprimant une demande de migration. Ceci est l'équivalent technique de la soumission d'une requête à un actuateur, et montre comment un Agent peut évaluer une valeur correspondant à une réalité physique pour produire un Contexte en tenant compte;
- h constate la demande de migration émise par a et entreprend la constitution de Ξ_a;

- le processus de $h \xrightarrow{a} h' : h' \in \Xi_a$ est réalisé;
- la disparition de a sur h est constatée;
- l'apparition de a sur h' est constatée.

Une demande de migration due à des ressources insuffisantes peut se produire dans une multitude de scénarios concrets : panne de GPS sur un appareil alors qu'un Agent dépend de cet instrument; mémoire dangereusement basse alors qu'un Agent en a besoin; pile trop faible (notre scénario); etc. Ce scénario, bien qu'artificiel, se veut représentatif d'une vaste classe d'irritants semblables; les exigences individuelles de tout Agent a apparaissent dans son composant R_a, et la production de la demande de migration est automatisée sur la base de ce Contexte.

Ce scénario fonctionne à partir d'un Agent du domaine pleinement contextifié, et constitue un guide pour la rédaction future d'Agents du domaine plus complexes. Il fait la démonstration de l'action d'un capteur (pour la lecture de la pile), d'un actuateur (pour le signalement d'une demande de migration, optionnelle dans le cas qui nous intéresse), de la prise en charge d'une migration par une paire d'Hôtes dans le respect des contraintes d'un Agent et de l'effet d'une telle migration.

L'Agent utilisé pour les besoins du test a de prime abord la forme décrite à la figure 22 :

```
idmigrateur {
  "human readable name" { migrateur }
  mission {
     eval { "battery_value < 98 != migrate" }
  }
  "ontic frame" {
    wants {
      BE6719FD-55E7-4E63-9AAB-130213588868 {
        ** { local { ** { battery { ** { remaining { value { ?=battery_value }}}} } }}
      }
    }
    produces {
       migration { request { eval { "migrate" } } }
    }
    does {
      compose {
         #0 { none }
         #1 { eval { "migrate = battery_value < 98; commit { migrate }" } }
      }
    }
  }
  restriction {
    eval { "battery_value >= 98" }
  }
  space {
  }
}
```

Figure 22 - Agent migrateur

Le seuil de 98% pour la pile est arbitraire et ne correspond pas à un scénario de réel besoin; il vise plutôt à permettre des tests simples (il suffit de débrancher l'appareil sur lequel réside un Hôte pour un court laps de temps pour que la mécanique s'enclenche). L'intention est de démontrer que la demande de migration se fait automatiquement du moment que l'Agent a constate que les conditions de son composant R_a ne sont plus rencontrées.

Le test décrit ici est fait sur un nœud, et la pile examinée est celle de ce nœud. Le besoin de migrer et la production de la demande de migration correspondante sont des actions locales à un nœud; l'interaction avec les Hôtes $h' \in \Xi_a$ s'ensuit. Pour les besoins du test, nous nous sommes limités à un seul nœud de destination, donc $|\Xi_a| = 1$. Des tests ultérieurs viseront à développer de bonnes stratégies pour choisir un Hôte de destination dans le cas où $|\Xi_a| > 1$, sur la base des règles exprimées dans O_a.

Comme c'est en général le cas dans ContextAA, le constat du besoin pour un Agent de migrer n'est pas instantané : les demandes associées à R_a sont émises dans le cadre du cycle d'exécution de a, et le constat par $a \in h$ que les exigences de R_a ne peuvent plus être rencontrées sur h se fait au cours d'une exécution subséquente de son cycle. La conception de R_a pour un Agent a donné doit donc, en pratique, tenir compte de cette non-instantanéité décrit dans §5.6.4.

Ce test simple vise à illustrer le fonctionnement global de ContextAA, et fait la démonstration que l'ensemble des mécanismes de base décrits dans cette thèse opèrent dans le respect des attentes. Ce que ce test ne montre pas est comment ContextAA soutient une utilisation à grande échelle; c'est par le déploiement de ce système dans un contexte plus réel que nous aurons un portrait plus clair de ses forces et de ce qui doit y être ajusté.

6.3.3 Construction d'un modèle d'activité

Dans [81], ContextAA est utilisé comme substrat pour construire un modèle d'activité permettant de répondre à la question « *Can programming language constructs support domain experts to increase (sic.) knowledge representation of dynamic activities, for improving the interaction and adaptation of context-aware appplications?* ». Le cadre ontique, que l'article nomm *ontic knowledge*, soutient le modèle qui y est construit.

L'approche développée dans cet article repose sur des tâches (*Assignments*), où chaque tâche repose sur un agrégat comprenant un cadre ontique, un ensemble de variables, un ensemble de conditions, un ensemble de prédicats et un ensemble de variables modifiées (le Contexte généré suivant les actions posées en réaction aux prédicats qui se seront avérés). Ceci nous a permis de construire un système à base de règles sur le substrat qu'est ContextAA, pour générer des automates capables de suivre un plan pour réaliser des actions qui, prises ensemble, constituent des tâches plus complexes.

Plus en détail, ce projet construit sur ContextAA utilise des $CTerm$, qui sont en pratique des paires de Contexte où l'un des éléments représente un type de Contexte et l'autre élément représente le Contexte de ce type, pour fins de classification. Les Contextes d'un $CTerm$ y prennent une forme spécifique convenant aux besoins du projet (nom, relation, poids, etc.), ce que permet le langage du Contexte mis de l'avant dans ContextAA. Les $CTerm$ sont une première approche dans la formalisation de types de Contextes dans un contexte applicatif autre que ContextAA lui-même.

Le projet implémente un dépôt mutable et synchronisé de $CTerm$, analogue à une base de données NoSQL de paires de Contexte. Bien que ContextAA ne propose pas de tel composant à même le modèle que nous mettons de l'avant, [81] fait la démonstration empirique qu'il est possible d'utiliser ContextAA dans une architecture plus typique, comme celle décrite par la figure 3.

Le projet utilise des variantes de $CTerm$ pour représenter des règles exprimées sous forme de conditions et des uplets décrivant des relations entre des Contextes, organisées sous forme d'un graphe acyclique parcouru par un automate pour réaliser, à partir d'un algorithme générique, les diverses actions décrites lorsque les conditions associées sont rencontrées (nommées $CTermActivity$). La génération de tâches complexes à partir d'un graphe de $CTermActivity$ y est réalisée par une accumulation des Contextes de ce graphe. Ce faisant [81] démontre la capacité qu'a ContextAA de jouer un rôle architectural, de soutenir le fonctionnement d'applications contextualisées concrètes, de définir des relations et reconstruire les éléments de Contexte « classiques » que sont les activités, les entités, les actions et les conditions, etc.

6.3.4 Construire des domaines sémantiques dynamiques

Dans [82], nous mettons de l'avant ce que nous avons nommé les domaines sémantiques dynamiques, soit des domaines bien délimités de connaissances pour la mise en place d'applications contextualisées, et utilisent le domaine restreint du confort, défini comme le contrôle de la température ambiante, pour construire des applications contextualisées « micro » [2].

Pour ce faire, un ensemble de types valeurs ont été définis. Ces types sont construits à partir de Contexte, et vont de types « primitifs » comme des booléens, des entités représentant des valeurs pour fins de calculs en logique floue ou des nombres à des types composites (listes, conteneurs associatifs) pour aller jusqu'à des abstractions plus complexes (processus, service), le tout en reprenant l'idée des $CTerm$ mise de l'avant par [81] et décrite dans §6.3.3.

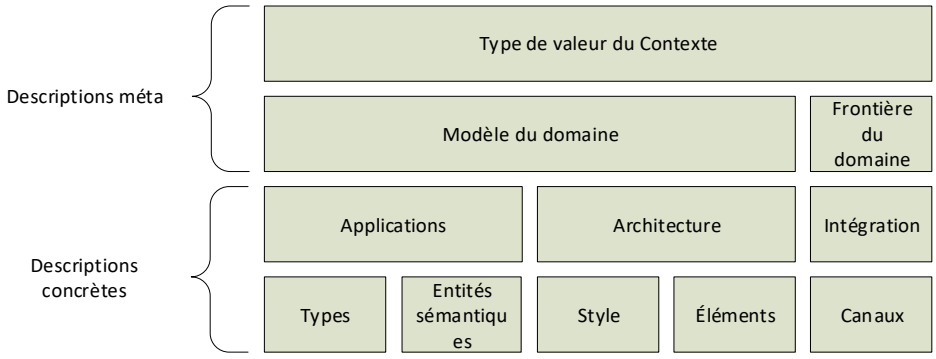


Figure 23 Types valeurs basés sur le Contexte selon [82]

Nous avons mis en place un cadriciel nommé Ami-DEU-Semantics pour générer des méta-descriptions de domaines sémantiques, desquelles sont construites des représentations concrètes de services; la figure 23 illustre ces deux catégories descriptives. Ce cadriciel adopte la représentation du Contexte mise de l'avant par ContextAA de même que sa possible transformation en d'autres formats, commerciaux, pour fins d'interopérabilité. La modélisation canonique du Contexte y est privilégiée, et des structures de données permettant la description de valeurs sur la base de logique floue y est mise de l'avant.

Les tests en soi sont faits en situation de simulation sur la base de logique floue. Un domaine sémantique représentant des paramètres de confort est représenté par du Contexte. L'entrée d'un Agent dans l'environnement provoque des actuations demandant une adaptation des conditions de température et de luminosité. Un service de logique floue assure la correspondance entre des degrés qualitatifs de température (froid, normal, chaud) et de luminosité (sombre, bonne, illuminée) et le type d'ajustement à apporter aux conditions ambiantes. La correspondance entre les concepts qualitatifs et des équivalents quantitatifs est représentée par du Contexte, et ce Contexte est traité comme une opération d'actuation par un Agent dédié à cette tâche et en interaction avec les actuateurs physiques.

Nos travaux montrent, sur la base de cette représentation, la capacité de modéliser l'évolution d'une représentation quantifiée du confort pour étudier dans un environnement donné en fonction du temps et de l'évolution des conditions ambiantes.

6.4 Discussion

Sur le plan architectural, nous savons que les Agents ne consomment pas tous la même proportion du temps de calcul à chaque itération du cycle normal d'opérations d'un Hôte h. En particulier, le *ResolverAgent*, chargé de résoudre les requêtes posées par l'ensemble des Agents sur h, est nettement plus gourmand que les autres, ce qui est raisonnable compte tenu de sa charge de travail et de la complexité structurelle du Contexte.

Une option pour adresser cet irritant serait de découpler le traitement fait par cet Agent du cycle normal d'opération de son Hôte, et de reléguer ce traitement dans un objet autonome. Ceci ramènerait cet Agent au rôle de tiers interne responsable d'interagir avec S_h. Il y a par contre un coût à ce changement structurel, soit celui de dupliquer en tout ou en partie S_h lorsque ce nouvel objet autonome prendrait en charge une nouvelle requête.

Si dupliquer la totalité de S_h semble trop coûteux, et si découpler la résolution des requêtes pour du Contexte de l'opération régulière du *ResolverAgent* apparaît nécessaire, alors il est possible que des mécanismes de duplication partielle d'éléments de S_h soient investigués. Évidemment, un tel changement impliquerait une forme de synchronisation sur S_h, et ne sera fait que si des tests empiriques montrent que les impacts du changement sont positifs.

Conclusion

« “La fin, quand c’est fini, comment le sait-on, comment fait-on, comment ça finit? » (Camille Laurens / Dans ces bras-là)

ContextAA est un système de composants Hôtes et d’Agents répartis mettant en applications une approche reposant sur le Contexte, à la fois en tant que langage et en tant que format de données, tout en utilisant des adaptateurs vers des technologies tierces sous la forme d’objets autonomes. Ce système propose une perspective architecturale « micro » reposant sur des Agents autonomes et un cadre ontique par Agent, le « Contexte-selon », qui accorde à tout Agent une forme d’indépendance face aux autre Agents et à d’hypothétiques composants spécialisés.

Le tout fonctionne, consomme peu de mémoire vive en pratique, et commence à être déployé dans des environnements de test. Par exemple, Ponce et al. [82] construit une strate sémantique enrichie sur la base de ContextAA et intègre le tout avec une communication par MQTT. Ces expérimentations mettent en relief des raffinements possibles au modèle de base de ContextAA, des améliorations à la mécanique des requêtes, des opportunités de simplification, etc. mais permettent de constater qu’il est possible de construire sur les fondations que ContextAA met en place; la conception d’outils pour assister la création d’applications sur la base de ContextAA est pour nous un objectif à court et à moyen terme.

ContextAA a été construit pour répondre à un ensemble de besoins, mis de l’avant dans §3.1. Certains de ces besoins portent sur les environnements intelligents contemporains, urbains et ouverts, que nous avons groupés sous le vocable d’espace intelligent ouvert. Nous avons conçu ContextAA dans une optique d’adaptabilité, d’interaction avec une diversité technologique croissante, en mettant en place dès le début des ponts technologiques sous la forme d’objets autonomes, capables d’enrichir ContextAA avec le passage du temps sans affecter le cœur du système. Des projets sont déjà en cours pour démontrer la capacité d’adaptation de ContextAA sur la base d’interopérabilité avec des technologies tierces. Nous avons aussi placé au cœur de notre approche le Contexte-selon, qui accorde aux Agents une capacité d’opération autonomique et accroît leur indépendance dans une perspective de pleine mobilité des individus.

Le Contexte, format sur lequel ContextAA repose en grande partie, a été pensé pour ne pas se limiter aux catégories traditionnelles de contexte ou à nécessiter une Ontologie partagée, mais bien pour offrir une capacité de concordance syntaxique entre Contexte produit et Contexte demandé. Ceci permet à chaque Agent de décrire le Contexte-selon lui, et participe à son autonomie en réduisant sa dépendance envers des composants tiers et envers une plateforme particulière.

La définition que donne ContextAA d’un Agent, particulièrement d’un Agent du domaine, permet de représenter les Agents sous forme de Contexte et de le faire participer pleinement à la démarche de contextualisation. Les distinctions entre formes de Contexte que d’autres technologies apportent sont ramenées, par le Contexte-selon, à une perspective par Agent. En plus de l’indépendance accrue face aux composants tiers et à une Ontologie, la contextualisation opérée dans ContextAA mène à des modalités opératoires allégées par le soutien intrinsèque d’un seul format de données, et permet de réduire les besoins en termes de ressources des Hôtes, comme en font foi les données de la section §6.1.

Le Contexte-selon participe d'une autre manière à la réduction des besoins en ressources d'un Hôte, c'est-à-dire en permettant à chaque Agent de circonscrire ce qui tient lieu de Contexte pertinent pour lui, et d'exclure d'office le Contexte qui ne rencontre pas ces exigences. Jointe à l'ensemble de l'architecture de ContextAA, cette réduction des besoins pour les nœuds fait en sorte de créer un environnement d'exécution pour les Agents qui soit adapté à une variété de nœud et qui fasse l'économie de choses telles qu'une dépendance envers un serveur de Contexte ou un raisonneur.

Le choix fait pour ContextAA de privilégier la continuité de service plutôt que la qualité du service (sans négliger cette dernière quand cela s'avère possible) entraîne pour ContextAA le choix de ne pas chercher à garantir la disponibilité en tout temps de tout le Contexte, et de mener plutôt les Agents à raisonner sur le Contexte acquis dans le respect de leurs requêtes, en acceptant que ce Contexte ne sera actualisé que lorsque cela s'avèrera possible. ContextAA demande bel et bien une distinction de perspective avec les divers systèmes décrits dans §2.4, et nous sommes d'avis que cette architecture sied mieux à l'espace intelligent ouvert que celles qui l'ont précédé.

ContextAA s'inscrit dans une démarche plus large, dans laquelle s'inscrivent d'autres projets de recherche. En parallèle, les travaux de raffinement et d'amélioration sur ContextAA en tant que tel se poursuivent. Entre autres raffinements envisagés, notons que nous savons d'ores et déjà qu'une utilisation à plus grande échelle de ContextAA demandera la mise en place d'autres outils tiers, par exemple des environnements de développement, facilitant l'expression sous forme de Contexte de considérations dites « de haut niveau », moins attachées aux détails qui ne touchent pas au cœur du propos qui préoccupe les expertes et les experts du domaine étudié.

Bien que nous ne le fassions pas encore, nous savons qu'il est possible pour un Hôte h d'éviter la duplication de certains Contextes dans S_h en les partageant entre plusieurs Agents. Ceci peut représenter une économie d'échelle significative en pratique, surtout dans le cas de Contextes récurrents comme ceux décrivant la représentation de la plateforme sous-jacente à h ou ceux exprimant des transformations de référentiels. Une éventuelle croissance de la consommation de ressources par Hôte pourrait être compensée par un effort d'optimisation en ce sens, et il en serait de même si le nombre d'Agents par Hôte vient à augmenter.

Pour l'instant, ContextAA n'offre pas de soutien spécifique pour des Agents assujettis à des contraintes de temps réel, ou à tout le moins des Agents assujettis à des contraintes de basse latence (les contraintes exprimées à §5.7.2 et à §5.7.3 laissent quelques pistes en ce sens). Une adaptation du cycle d'opération d'un Hôte pourrait mener à une ouverture de ContextAA vers le soutien de tels Agents.

La modalité opératoire de ContextAA repose pour l'instant sur le sondage, au sens où un requérant questionne des tiers suivant un rythme qui lui est propre. Adapter les objets autonomes pour adopter une modalité d'abonnement pourrait permettre de réduire les besoins en ressources d'un Hôte. Ceci demande toutefois à être mesuré, ce qui demandera la mise en place de tests comparatifs.

Le langage du Contexte bénéficierait d'être enrichi à certains égards. Un volet qui ressort de nos expérimentations jusqu'ici est qu'il serait judicieux de faciliter la prise en compte des séquences de valeurs, qu'il s'agisse de tableaux, de listes ou d'autres structures de données incluant une forme d'ordonnancement. Ce problème est contourné pour le moment par l'ajout d'une racine dont la valeur croît de manière monotone, mais une meilleure forme syntaxique rendrait le Contexte plus facile d'approche. Puisque l'on parle d'ajustements à la langue du Contexte pour faciliter l'expression des idées, le vocable « nos expérimentations » ici réfère l'acte de programmer les tests et au constat que, par exemple pour une opération $compose$, la syntaxe utilisée, bien que fonctionnelle, pourrait être raffinée pour facilier la programmation. Nous pensons que l'impact de l'amélioration de la fluidité de l'expression dans la langue du Contexte se mesurera d'abord en termes de productivité de la part des programmeuses et des programmeurs.

Un autre volet pour lequel la syntaxe du Contexte pourrait être améliorée est celle de la composition d'opérations. En effet, bien que les opérations du type $compose$ existent et fonctionnent en pratique, il serait avantageux sur le plan de l'utilisabilité de faciliter le séquencement des opérations, peut-être en s'inspirant des « pipes » du monde Unix / Linux.

Un volet plus litigieux de la syntaxe du Contexte, mais qui pourra tout de même être investigué, est l'enrichissement du sous-langage des expressions $eval$ pour faire en sorte qu'il soit possible d'y représenter des fonctions, en particulier des fonctions récursives, mais ceci pourrait faire de ce métalangage une entité complète au sens de Turing, ce qui entraîne des enjeux de sécurité qui méritent réflexion.

La τ-équivalence décrite à §4.5.3.2 n'est encore qu'à un stade embryonnaire et demande à être élargie et étoffée. Nous y voyons un créneau d'intérêt pour le proche futur. De même, à moyen terme, ContextAA bénéficierait de l'automatisation des rapprochements de Contextes, mais il faudrait pour ce faire trouver un algorithme permettant de réaliser cette tâche en conservant la complexité algorithmique à un niveau acceptable en pratique.

Sur le plan architectural, les canaux entre le Communicateur et les Agents standards avec lesquels il transige constituent un possible goulot d'étranglement dans l'architecture d'un Hôte pour ContextAA. Cette situation n'est pas un problème en pratique, du moins au moment d'écrire ces lignes, mais il est possible qu'il y ait un jour lieu de repenser cette partie de l'infrastructure.

Les facultés d'oubli mises en place actuellement répondent aux attentes, mais nous savons qu'il arrivera en pratique qu'une requête n'ait jamais de réponse. Il n'y a présentement pas de méta-Contextes permettant de donner une « date de péremption » aux requêtes ou un mécanisme équivalent; c'est une avenue que nous comptons explorer.

Enfin, nous souhaitons en arriver à de meilleurs diagnostics lors d'erreurs : le système est raisonnablement robuste, mais la gestion des échecs lors de l'exécution d'un Agent mériterait d'être améliorée – poursuivre l'exécution sans expliquer n'est pas nécessairement la chose à faire. Nous tenons un journal des erreurs pour fins d'analyse *a posteriori*, mais ce journal pourrait être enrichi pour accroître la productivité des gens utilisant ContextAA.

Il reste beaucoup à explorer sur les chemins dans lesquels ContextAA nous a permis de nous engager. Bien que le cœur conceptuel et architectural de ContextAA apparaissent solides sur la base des expérimentations faites jusqu'ici, que le tout soit stable à l'exécution et qu'il ait jusqu'ici été possible de faire évoluer ContextAA de l'idée qu'il était à l'origine jusqu'à l'architecture présentée ici, les contours de ce que deviendra ce système restent en partie à définir. ContextAA est un système stimulant et pertinent. Nous en sommes fiers.

ContextAA sous-tend des approches que nous estimons fécondes et valables, en particulier le Contexte-selon et le développement « micro » qui favorise l'autonomie des Agents. Il répond aux besoins des applications contextualisées traditionnelles, mais pose les questions différemment et choisit des objectifs différents : continuité de service, autonomie sans autorité centralisée, cadre ontique plutôt qu'Ontologie, approche « pur-Contexte » plutôt que reposant sur des technologies commerciales mais plus gourmandes sur le plan des ressources, etc.

Nous pensons, avec ContextAA, avoir mis en place une infrastructure qui nous permettra d'appréhender l'espace intelligent ouvert, dans toute sa richesse, sa diversité et sa complexité. Notre tâche sera maintenant de l'accompagner dans son évolution et d'accompagner celles et ceux qui feront le choix d'en enrichir l'écosystème.

Annexe A
Vocabulaire des noms de Contextes

Le vocabulaire associé aux noms de Contextes est donné par les règles EBNF [50] suivantes. Notez que chaque $primitive\_operation$ est l'expression par un nom de Contexte d'une opération qui sera réalisée par un Agent standard sur l'Hôte qui la prendra en charge :

Table 4 - Vocabulaire de noms de Contextes

$\langle whitespace \rangle ::=$	$[\backslash t \backslash n \backslash r \backslash v \backslash b\ ]$
$\langle digit \rangle ::=$	$[0-9]$
$\langle integer \rangle ::=$	$[+-]\langle digit \rangle +$
$\langle letter \rangle ::=$	$[a-zA-Z]$
$\langle punct \rangle ::=$	$[\hat{}\langle digit \rangle \langle letter \rangle \langle whitespace \rangle \backslash \# \backslash : \backslash " \backslash \{ \backslash \} \backslash * \backslash \_]$
$\langle acceptable\_symbol \rangle ::=$	$\langle digit \rangle \mid \langle letter \rangle \mid \langle punct \rangle$
$\langle basic\_name\_element \rangle ::=$	$\langle acceptable\_symbol \rangle + (\_\langle acceptable\_symbol \rangle) *$ $\mid \backslash "[\cdot] * \backslash "$
$\langle name\_element \rangle ::=$	_ | ? $\mid \langle basic\_name\_element \rangle$
$\langle primitive\_operation \rangle ::=$	$leaf\_nodes$ $\mid is\_number$ $\mid is\_date$ $\mid is$ $\mid and$ $\mid anyof$ $\mid or$ $\mid not$ $\mid compose$ $\mid eval$ $\mid has$ $\mid keep\_if$ $\mid filter\_if$ $\mid none$ $\mid structured\_like$ $\mid valued\_like$ $\mid exhaustive$ $\mid nameof$ $\mid valueof$

$Name ::=$	ε $\mid *$ $\mid **$ $\mid \langle primitive\_operation \rangle$ $\mid \#\langle integer \rangle$ $\mid ? = \langle basic\_name\_element \rangle$ $\mid = [\langle basic\_name\_element \rangle]$ $\mid \langle name\_element \rangle (: \langle name\_element \rangle) *$

Annexe B
Expressions *eval*

Les expressions *eval* suivent la forme suivante :

Table 5 - Expressions ***eval***

⟨*exprs*⟩ ::=	⟨*expr*⟩[;⟨*expr*⟩] + [;]
⟨*expr*⟩ ::=	⟨*or_expr*⟩
⟨*or_expr*⟩ ::=	⟨*and_expr*⟩[(||⟨*and_expr*⟩|?⟨*expr*⟩:⟨*expr*⟩)]
⟨*and_expr*⟩ ::=	⟨*eq_expr*⟩[&&⟨*eq_expr*⟩]
⟨*eq_expr*⟩ ::=	⟨*ord_expr*⟩[(== ⟨*ord_expr*⟩|! = ⟨*ord_expr*⟩)]
⟨*ord_expr*⟩ ::=	⟨*add_expr*⟩[(< ⟨*add_expr*⟩|≤ ⟨*add_expr*⟩|> ⟨*add_expr*⟩| ≥ ⟨*add_expr*⟩)]
⟨*add_expr*⟩ ::=	⟨*mult_expr*⟩[(+⟨*mult_expr*⟩|−⟨*mult_expr*⟩)]
⟨*mult_expr*⟩ ::=	⟨*prim*⟩[(∗ ⟨*prim*⟩|/⟨*prim*⟩|%⟨*prim*⟩)]
⟨*prim*⟩ ::=	*nombre*|?*name*|*commit*[\{*name*[*name*] +\}]|;|−⟨*prim*⟩|!⟨*expr*⟩|\(⟨*expr*⟩\)

Les expressions *eval* forment un sous-langage du Contexte permettant à un Agent de réaliser des calculs arithmétiques et logiques de son choix sur du Contexte et de collecter les résultats de ces calculs dans un dictionnaire pour usage ultérieur.

Annexe C
Critères de Contexte

Les opérations qui suivent sont exprimées sous forme de Contexte, et ont la particularité de s'appliquer à un Contexte pour produire l'un de deux Contextes que sont $true$ et $false$.

Table 6 - Critères de Contexte

Critère	Effet
$has\{patt\}$	S'avère pour $c: C$ seulement s'il y a correspondance entre $patt$ et c
$is\{patt\}$	S'avère pour $c: C$ seulement s'il y a correspondance entre $patt$ et $name(c)$
is_date	S'avère pour $c: C$ seulement si c exprime une date
is_number	S'avère pour $c: C$ seulement si c exprime un nombre
not	S'avère pour $c: C$ seulement si c est $false$

La liste des critères retenues jusqu'ici a évolué de manière organique à partir des besoins. L'intention est de limiter les « mots-clés » au minimum dans ContextAA; en ce sens, un nom de critère peut être vus comme une sorte des « mot-clé contextuel », donc ne jouant un rôle de mot-clé que s'il apparaît à certains endroits dans un Contexte (typiquement, à la racine), mais rappelons qu'un Agent peut transformer un Contexte (p. ex. : abstraire ou supprimer une racine) ce qui demande une certaine prudence.

Les usages quant aux critères se dessinent avec l'usage, mais la liste présentée ici suffit aux besoins de ContextAA tel qu'il a été présenté ici.

Annexe D
Transformations de Contexte

Les opérations qui suivent sont exprimées sous forme de Contexte, et ont la particularité de s'appliquer à un Contexte pour produire un Contexte.

Table 7 - Transformations de Contexte

Transformation	Effet
$exhaustive$	Produit la liste des sous-Contextes de c
$or\{m_0, m_1, \dots, m_n\}$	Produit la fusion de la mise en correspondance de c avec chacun des modèles $m_0, m_1, \dots, m_n$
$filter\_if\{m\}$	Produit $\left[c' : C \parallel \neg\left(isSubCtx(c', c) \wedge matches(m, c')\right)\right]$
$keep\_if\{m\}$	Produit $[c' : C \parallel isSubCtx(c', c) \wedge matches(m, c')]$
$leaf\_nodes$	Produit la liste des sous-Contextes terminaux de c
cut_root	Produit $value(c)$
$none$	Produit un Contexte vide. Ceci permet d'exprimer une requête compose (§Annexe E) dont une des étapes ne donne aucun Contexte, et peut accélérer le traitement d'une demande à partir de laquelle l'Agent ne s'intéresse qu'au contenu de son dictionnaire

Les remarques sur le choix de la liste des critères de Contexte (annexe C) s'appliquent aussi pour le choix de la liste des transformations de Contexte.

Annexe E
Autres opérations de Contexte

Les opérations qui suivent sont exprimées dans ContextAA sous forme de Contexte mais ne conviennent pas directement aux rubriques de §0 ou §Annexe D.

Table 8 - Autres opérations de Contexte

Opération	Effet
$compose\{\#0\{c_0\}\#1\{c_1\}\ldots\#n\{c_n\}\}$	Applique c_0 à un Contexte c, puis applique $c_1$au Contexte résultant, puis…, puis applique c_n au Contexte résultant
$structured\_like\{c, c'\}$	Évalue $\sigma(c, c')$
$valued\_like\{c, c'\}$	Évalue $\varrho(c, c')$

Les remarques sur le choix de la liste des critères de Contexte (annexe C) s'appliquent aussi pour le choix de la liste des autres opérations de Contexte présentées ici.

Annexe F
Principaux Agents standards

Les Agents standards jouent un rôle structurel pour leur Hôte, assurant pour lui certains services de base. Parmi ces Agents se trouvent :

- un Agent responsable de prendre en charge les requêtes consommées à la ligne de commande, ou *CommandLineInterfaceAgent* servant de contrepartie à l'objet autonome *CommandLineInterface* chargé d'offrir une interaction simple avec l'usager. Cet Agent est surtout utile dans un environnement de développement muni d'une interface de type console, mais peut avoir une contrepartie adaptée aux réalités des divers nœuds selon les besoins;
- un Agent intermédiaire, ou *ProxyAgent*, qui a pour rôle de prendre en charge sur $h: H$ les requêtes reçues d'un Agent logé sur $h': H(h' \neq h)$;
- un Agent assurant le lien entre la plateforme sous-jacente à son Hôte de même qu'avec les ressources de cette plateforme, ou *PlatformInterfaceAgent*;
- un Agent assurant la publication des requêtes pour du Contexte faites par $\forall a \in h$ vers $h' \in N_h$, ou *RequesterAgent*;
- un Agent assurant la résolution des requêtes pour du Contexte exprimées par $\forall a \in h$, ou *ResolverAgent*;
- un Agent assurant l'entretien de $S_a \forall a \in h$, ou *ContextManagerAgent*; et
- un certain nombre d'Agents auxiliaires offrant des services tels que compter les itérations du cycle d'exécution du gestionnaire d'agents ou assurer la mise à jour de N_h.

Un Agent standard joue un rôle structurel dans la résolution des requêtes pour du Contexte dans ContextAA. À titre d'exemple, si $a \in h$ publie dans S_a une requête q pour du Contexte privé, alors le *ResolverAgent* de h, en s'exécutant, construit $r: C\big(value(r) = \bigcup_{c \in S_a} q(c)\big)$ pour l'insérer dans S_a.

Si $a \in h$ publie dans S_a une requête q pour du Contexte local, alors le *ResolverAgent* de h, lorsque vient son tour de s'exécuter, construira $r: C\left(value(r) = \left(\bigcup_{c \in S_h} q(c) \wedge \big(c \in S_a \vee \neg privé(c)\big)\right)\right)$ pour l'insérer dans S_a.

Si $a \in h$ publie dans S_a une requête q pour du Contexte public, alors les actions à prendre sont plus complexes. En effet :

- sur h, les actions prévues pour une fouille de Contexte local sont alors effectuées;
- de plus, le *RequesterAgent* de h prend q et l'envoie vers $\forall h': h' \in N_h$. Cet envoi sera fait en construisant q', version pleinement qualifiée par a et h de q, puis en publiant q' sur le canal de Contexte du composant *Communicateur* qui, de son côté, le distribue à chaque h' par les composants émetteurs qui leur sont associés;
- sur réception dans un h' donné par un composant *Récepteur*, q' est placée sur un canal de Contexte pour être consommée par le *ProxyAgent* de h' qui la publie ensuite sous forme q'' dans h' en tant que requête pour du Contexte local;

- lorsque le *ResolverAgent* de h' a traité q'' pour produire r'' dans l'espace contextuel du *ProxyAgent*, ce dernier transforme r'' en r' de manière à ce que $h{:}\,a$ en soit le destinataire, puis publie r' sur le canal de Contexte du *Communicateur* de h'. Ce dernier place alors r' dans le tampon sortant du composant émetteur associé à h;
- sur réception par le composant récepteur sur h de r' destiné à $h{:}\,a$, le *ProxyAgent* de h dépose enfin r' dans S_a.

Procédant ainsi, les Agents standards sur les divers Hôtes opèrent localement sur le Contexte ou relaient à d'autres Hôtes le Contexte par la voie des composants émetteur et récepteur appropriés.

Annexe G
Distance structurelle pour un ensemble de cas de tests

Pour fins de tests, la distance structurelle σ a été évaluée pour un ensemble de Contextes distincts, chacun étant comparé à tous les autres de cet ensemble. Le tableau 9 ci-dessous liste les valeurs calculées dans chaque cas. Les résultats proposés sont placés en ordre croissant de distance structurelle, donc de la paire la plus semblable à la paire la plus dissemblable.

Puisque $\forall c \in C(\sigma(c,c) = 0)$, les cas de comparaison structurelle d'un Contexte avec lui-même n'ont pas été conservés explicitement (quelques-uns sont présentés en début de liste, pour illustrer le propos, sans plus). Aussi, puisque $\sigma(c,c') = \sigma(c',c)$, nous avons choisi de faire en sorte qu'une fois le résultat de $\sigma(c,c')$ présenté, le résultat de $\sigma(c',c)$ ne le soit pas (par souci d'économie).

Toutes les distances structurelles évaluées le sont avec un ensemble de transformations vide ($T = \emptyset$). L'alternance entre lignes grisées et lignes non-grisées vise à grouper des résultats de même valeur pour faciliter la lecture.

Table 9 - Distance structurelle (cas de tests)

c	c'	$\sigma(c,c')$
a	a	0
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0
h:a:0 { value { 10 } }	h:a:0 { value { 10 } }	0
… et ainsi de suite $\forall c, c'(c = c')$		
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,117284
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,119342
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,12963
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,131687
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,131687
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,133745

c	c'	$\sigma(c, c')$
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,135802
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,141975
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,146091
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,154321
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,154321
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,158436
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,160494
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,201646
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,201646
h:a:1 { value { 50 } }	h:a:0 { value { 10 } }	0,222222
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:2 { unit { celsius } value { 10 } }	0,222222
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:2 { unit { celsius } value { 10 } }	0,222222
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,244856
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,259259
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,265432

c	c'	$\sigma(c, c')$
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,271605
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,271605
h:a:0 { value { 10 } }	h:a:2 { unit { celsius } value { 10 } }	0,277778
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,277778
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:1 { value { 50 } }	0,277778
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,279835
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,279835
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,296296
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,296296
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,306584
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,306584
h:a:0 { value { 10 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,333333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,333333
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,349794
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:0 { value { 10 } }	0,388889
h:a:1 { value { 50 } }	h:a:2 { unit { celsius } value { 10 } }	0,388889
h:a:1 { value { 50 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,444444

c	c'	$\sigma(c, c')$
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:1 { value { 50 } }	0,5
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:1 { value { 50 } }	0,5
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:1 { value { 50 } }	0,5
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:1 { value { 50 } }	0,5
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:1 { value { 50 } }	0,5
h:a:0 { value { 10 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,555556
h:a:1 { value { 50 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,555556
h:a:1 { value { 50 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,611111
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:1 { value { 50 } }	0,611111
h:a:0 { value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,611111
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:0 { value { 10 } }	0,611111
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:0 { value { 10 } }	0,611111
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:0 { value { 10 } }	0,611111
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:0 { value { 10 } }	0,611111

c	c'	$\sigma(c, c')$
h:a:0 { value { 10 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,611111
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:0 { value { 10 } }	0,611111
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	0,611111
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	b { c }	0,666667
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	b { c }	0,666667
h:a:0 { value { 10 } }	b { c }	0,666667
h:a:1 { value { 50 } }	b { c }	0,666667
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,666667
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	b { c }	0,666667
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	b { c }	0,666667
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	b { c }	0,666667
b { c }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,666667
b { c }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,666667
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:2 { unit { celsius } value { 10 } }	0,666667
b { c }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,666667
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,722222

c	c'	$\sigma(c, c')$
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,777778
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,777778
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	0,777778
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,777778
h:a:2 { unit { celsius } value { 10 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,777778
h:a:2 { unit { celsius } value { 10 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,777778
h:a:2 { unit { celsius } value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,777778
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,777778
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,777778
h:a:2 { unit { celsius } value { 10 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,777778
h:a:2 { unit { celsius } value { 10 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,777778
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,777778
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,777778
h:a:3 { unit { fahrenheit } value { 50 } }	b { c }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,833333

c	c'	$\sigma(c, c')$
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,833333
h:a:2 { unit { celsius } value { 10 } }	b { c }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,833333
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,833333
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	b { c }	0,888889
a	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	a	1
a	h:a:2 { unit { celsius } value { 10 } }	1
a	h:a:1 { value { 50 } }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	a	1
h:a:0 { value { 10 } }	a	1
a	b { c }	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	a	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	a	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	a	1

c	c'	$\sigma(c, c')$
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	a	1
a	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	a	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	a	1

Annexe H
Distance de contenu pour un ensemble de cas de tests

Pour fins de tests, la distance de contenu ϱ a été évaluée pour un ensemble de Contextes distincts, chacun étant comparé à tous les autres de cet ensemble. Les tableau 10 ci-dessous liste les valeurs calculées dans chaque cas. Les résultats proposés sont placés en ordre croissant de distance de contenu, donc de la paire la plus semblable à la paire la plus dissemblable.

Puisque $\forall c \in C(\varrho(c,c) = 0)$, les cas de comparaison de contenu d'un Contexte avec lui-même n'ont pas été conservés explicitement (quelques-uns sont présentés en début de liste, pour illustrer le propos, sans plus). Puisque $\varrho(c,c') \neq \varrho(c',c)$ sauf dans quelques cas particuliers, nous avons présenté les résultats de la fonction dans un sens comme dans l'autre.

Toutes les distances de contenu évaluées le sont avec un ensemble de transformations vide ($T = \emptyset$).

Un problème potentiel apparaît à partir de $\varrho(c,c') = 0{,}239583$ avec la comparaison entre un Contexte décrivant de l'humidité et un autre décrivant une température, mais avec des précisions identiques et produites par le même Agent, ce qui entraîne une distance de contenu peu élevée. Un tel résultat est rendu possible par notre approche décrivant le Contexte d'une manière *a priori* plus syntaxique que sémantique.

En pratique, les probabilités que cette situation survienne sont plus faibles qu'on ne pourrait le croire, du fait qu'il est probable qu'un Agent décrive ses requêtes pour du Contexte en indiquant des noms permettant de discriminer la nature du Contexte souhaité, filtrant au préalable les Contextes de nature vraiment différente avant que les calculs de distance entre Contextes ne soient réalisés.

Dans certains cas, particulièrement ceux pour lesquels $\varrho(c,c') = 0{,}25$, il est probable que la distance de contenu serait nettement plus faible si une transformation menant à un changement de référentiel avait été appliquée.

Certains cas pour lesquels $\varrho(c,c') = 0{,}260417$ mettent en relief la force de cette fonction pour tracer des correspondances entre des Contextes qui auraient pu paraître semblables pour un humain.

Pour les cas à partir desquels $\varrho(c,c') = 0{,}395833$, les noms racines sont à une distance de $\frac{2}{3}$ l'un de l'autre, et le poids des noms racines dans le calcul de la distance de contenu explique que les distances évaluées soient relativement élevées. Il faut toutefois noter que, dans bien des cas, un Agent requérant ne souhaitera pas que le Contexte consommé provienne d'un émetteur spécifique; ainsi, appliquer une transformation telle que suppression de racine ou abstraction de racine mènerait à des distances plus faibles sans altérer la qualité relative du Contexte consommé.

Pour les cas tels que $\varrho(c,c') = 1$, les raisons varient : des Contextes très simples pour lesquels les noms ne correspondent pas (p. ex. : un cas tel que $c = a; c' = b\{c\}$, par exemple, ou encore des cas où les Contextes sont plus complexes et où il existe des similitudes de contenu, mais à des profondeurs trop différentes pour être détectées à l'aide de la fonction ϱ, du moins sans transformation. Un exemple serait :

$$c = h0{:}\,a0{:}\,1\left\{public\left\{src\{a0\}temp\{prec\{0.001\}unit\{cels\}value\{10\}\}\right\}\right\};$$

$$c' = h{:}\,a{:}\,4\{prec\{0.001\}unit\{cels\}value\{10\}\}$$

Dans ce dernier cas, des transformations pourraient amener l'Agent à reconnaître une similitude forte entre une partie de c et c', plus précisément : une suppression de racine suivie d'une abstraction de racine.

L'alternance entre lignes grisées et lignes non-grisées vise à grouper des résultats de même valeur pour faciliter la lecture.

Table 10 - Distance de contenu (cas de tests)

c	c'	$\varrho(c, c')$
A	a	0
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0
h:a:0 { value { 10 } }	h:a:0 { value { 10 } }	0
…et ainsi de suite $\forall c, c'(c = c')$		
h:a:1 { value { 50 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,166667
h:a:1 { value { 50 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,166667
h:a:0 { value { 10 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,166667
h:a:2 { unit { celsius } value { 10 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,166667
h:a:0 { value { 10 } }	h:a:2 { unit { celsius } value { 10 } }	0,166667
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,1875
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,197917
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,197917
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,197917
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,197917
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,197917

c	c'	$\varrho(c, c')$
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,229167
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,229167
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,229167
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,229167
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,229167
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,239583
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,239583
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,239583
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,25
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,25
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,25
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,25
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,260417

c	c'	$\varrho(c, c')$
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,260417
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,260417
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,260417
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,260417
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,260417
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,270833
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,270833
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,28125
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,28125
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:2 { unit { celsius } value { 10 } }	0,333333
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,34375
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,34375
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,395833
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,395833
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,395833

c	c'	$\varrho(c, c')$
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,395833
h:a:0 { value { 10 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,416667
h:a:0 { value { 10 } }	h:a:1 { value { 50 } }	0,416667
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:1 { value { 50 } }	0,416667
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,416667
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,416667
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:2 { unit { celsius } value { 10 } }	0,416667
h:a:1 { value { 50 } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	0,416667
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,416667
h:a:1 { value { 50 } }	h:a:0 { value { 10 } }	0,416667
h:a:2 { unit { celsius } value { 10 } }	h:a:0 { value { 10 } }	0,416667
h:a:1 { value { 50 } }	h:a:2 { unit { celsius } value { 10 } }	0,416667
h:a:2 { unit { celsius } value { 10 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,416667
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,427083
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,427083
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,427083
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,427083
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,447917
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,447917

c	c'	$\varrho(c, c')$
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,458333
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,458333
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,489583
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,489583
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,489583
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,489583
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,489583
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,489583
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:0 { value { 10 } }	0,5
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,5
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,5
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,5
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,5
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,5
h:a:3 { unit { fahrenheit } value { 50 } }	h:a:0 { value { 10 } }	0,541667

c	c'	$\varrho(c, c')$
h:a:2 { unit { celsius } value { 10 } }	h:a:1 { value { 50 } }	0,541667
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,552083
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,572917
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	h:a:1 { value { 50 } }	0,583333
h:a:1 { value { 50 } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,833333
h:a:1 { value { 50 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	0,833333
h:a:2 { unit { celsius } value { 10 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	0,833333
h:a:2 { unit { celsius } value { 10 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	0,833333
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:1 { value { 50 } }	0,833333
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	0,833333
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:2 { unit { celsius } value { 10 } }	0,833333
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:3 { unit { fahrenheit } value { 50 } }	0,833333
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:1 { value { 50 } }	0,833333
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	0,833333
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:1 { value { 50 } }	0,833333

c	c'	$\varrho(c, c')$
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:1 { value { 50 } }	0,833333
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:1 { value { 50 } }	0,833333
h:a:1 { value { 50 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	0,833333
h:a:1 { value { 50 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	0,833333
h:a:1 { value { 50 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	0,833333
a	b { c }	1
a	h:a:0 { value { 10 } }	1
a	h:a:1 { value { 50 } }	1
a	h:a:2 { unit { celsius } value { 10 } }	1
a	h:a:3 { unit { fahrenheit } value { 50 } }	1
a	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
a	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
a	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1
a	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
a	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1
a	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
a	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1

c	c'	$\varrho(c, c')$
a	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	a	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	b { c }	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:0 { value { 10 } }	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:1 { value { 50 } }	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1

c	c'	$\varrho(c, c')$
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	b { c }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	a	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	b { c }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:0 { value { 10 } }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:1 { value { 50 } }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	a	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1

c	c'	$\varrho(c, c')$
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	a	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	b { c }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:0 { value { 10 } }	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:3 { unit { fahrenheit } value { 50 } }	b { c }	1
h:a:3 { unit { fahrenheit } value { 50 } }	a	1
h:a:2 { unit { celsius } value { 10 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1
h:a:2 { unit { celsius } value { 10 } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:2 { unit { celsius } value { 10 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
h:a:2 { unit { celsius } value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1

c	c'	$\varrho(c, c')$
h:a:2 { unit { celsius } value { 10 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
h:a:2 { unit { celsius } value { 10 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	a	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	b { c }	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:0 { value { 10 } }	1
h:a:2 { unit { celsius } value { 10 } }	b { c }	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:2 { unit { celsius } value { 10 } }	a	1
h:a:1 { value { 50 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1
b { c }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:1 { value { 50 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
h:a:1 { value { 50 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1
h:a:1 { value { 50 } }	b { c }	1
h:a:1 { value { 50 } }	a	1
h:a:0 { value { 10 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1

c	c'	$\varrho(c, c')$
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	a	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	b { c }	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:0 { value { 10 } }	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:1 { value { 50 } }	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:0 { value { 10 } }	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:0 { value { 10 } }	host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:0 { value { 10 } }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1
h:a:0 { value { 10 } }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
h:a:0 { value { 10 } }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1
h:a:0 { value { 10 } }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
h:a:0 { value { 10 } }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1
h:a:0 { value { 10 } }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
h:a:0 { value { 10 } }	b { c }	1

c	c'	$\varrho(c, c')$
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	a	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	b { c }	1
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:0 { value { 10 } }	1
host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	a	1
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
h:a:0 { value { 10 } }	a	1
b { c }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1
b { c }	host0:agent2:2 { public { temperature { src { agent2 precision { 0.001 } unit { celsius } } value { 10 } } } }	1
b { c }	host0:agent2:1 { public { source { agent2 } temperature { precision { 0.001 } unit { fahrenheit } value { 50 } } } }	1
b { c }	host0:agent1:1 { public { source { agent1 precision { 0.01 } unit { celsius } } temperature { value { 10 } } } }	1
b { c }	host0:agent0:3 { public { humidity { precision { 0.1 } unit { kpa } value { 100 } } source { agent0 } } }	1
b { c }	host0:agent0:2 { public { source { agent0 } temperature { precision { 0.1 } unit { fahrenheit } value { 100 } } } }	1

c	c'	$\varrho(c, c')$
b { c }	host0:agent0:1 { public { source { agent0 } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	a	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	b { c }	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:0 { value { 10 } }	1
h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:2 { unit { celsius } value { 10 } }	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:3 { unit { fahrenheit } value { 50 } }	1
host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
b { c }	h:a:4 { precision { 0.001 } unit { celsius } value { 10 } }	1
b { c }	h:a:3 { unit { fahrenheit } value { 50 } }	1
b { c }	h:a:2 { unit { celsius } value { 10 } }	1
b { c }	h:a:1 { value { 50 } }	1
b { c }	h:a:0 { value { 10 } }	1
b { c }	a	1
a	host0:agent4:1 { public { source { agent4 precision { 0.001 } unit { celsius } } temperature { value { 10 } } } }	1
host0:agent3:1 { public { source { agent3 precision { 0.001 } unit { celsius } } temperature { precision { 0.001 } unit { celsius } value { 10 } } } }	b { c }	1

Annexe I
Modèles de Contexte et mécanique de prise en charge

Chaque Contexte dans ContextAA peut prendre une forme convenant aux besoins des Agents qui le consomment ou le produisent. Toutefois, les Agents standards utilisent des formes prédéfinies pour simplifier leur tâche et accélérer leur opération. En utilisant des noms

Requête

Dans ce qui suit, un Agent a sur un Hôte h soumet une demande c respectant un modèle de Contexte donné.

```
h:a:c {
   query:qid {
      qualif {
        patt
      }
   }
}
```

Ici, le nom $query$ est un marqueur identifiant l'intention de modéliser une requête de la part de l'Agent producteur a. Par la suite : qid est le numéro de la requête, ce qui permet de grouper plusieurs requêtes sous un même identifiant de Contexte; $qualif$ est typiquement $public$, $local$ ou $private$ et sert à décrire la visibilité de la requête; enfin, $patt$ constitue le modèle de Contexte décrivant la forme du Contexte recherché.

Réponse

Dans ce qui suit, un Agent a sur un Hôte h constate une réponse c à une demande respectant un modèle de Contexte donné.

```
h:a:c {
   response:qid {
     ctxs
   }
}
```

Ici, le nom $response$ est un marqueur identifiant le fait que c modélise une réponse à la requête qid, alors que $ctxs$ est une séquence de Contextes correspondant à la requête d'origine.

Requêtes prises en charge

Chaque Hôte possède un Agent standard de type *ProxyAgent*, dont le rôle est de consommer les requêtes entrant de $h' \in N_h$.

Son format entrant est :

```
h':a':c' {
   query:qid' {
      qualif {
        patt
      }
   }
}
```

suivant le format d'une requête (ci-dessus) où h', a', c' et qid' identifient la requête du point de vue de l'Agent a' producteur.

Le *ProxyAgent* fait sienne chacune de ces requêtes, et soumet une requête équivalente sur son propre Hôte, mais avec qualification de visibilité $local$. À ce titre, il joue le rôle d'entremetteur entre l'Hôte sur lequel il réside et le producteur de la requête.

Lorsque le *ProxyAgent* consomme une réponse, il construit un Contexte destiné au producteur de la requête auquel il a prêté assistance et publie ce dernier.

Transmission d'une requête publique – *RequesterAgent*

Sur tout Hôte, un Agent nommé *RequesterAgent* se saisit des requêtes avec qualification de visibilité $public$ et les publie vers $h' \in N_h$ pour que celles-ci y soient prises en charge par le `ProxyAgent` qui y loge.

Le *RequesterAgent* fait donc le pont entre un Hôte h et son voisinage N_h par la médiation des ses objets autonomes.

Prise en charge d'une requête locale – *ResolverAgent*

Sur tout Hôte, un Agent nommé *ResolverAgent* prend en charge l'ensemble des requêtes de Contexte qualifiées $private$ ou $local$, et résout les modèles demandés par celles-ci en interagissant avec les espaces contextuels appropriés.

Concrètement, de par la nature de son mandat, cet Agent est le plus important consommateur de temps de calcul sur un Hôte. En retour, il s'agit d'un lieu propice pour l'optimisation : avoir un point focal pour traiter de multiples requêtes de Contexte permet diverses tactiques de mémoïsation, de mise en commun de résultats intermédiaires, d'économie d'espace, etc.

Annexe J
Traduire de XML à Contexte et inversement

Le passage de Contexte à XML est trivial, les deux formats constituant une représentation structurée où les éléments ne se chevauchent pas. Le petit programme décrit à la figure 24 montre qu'il est possible de réaliser cette conversion en quelques lignes à peine, incluant l'indentation du code XML :

```
#include "../../Host/Host/contexte.h"
#include "../../Host/Host/text_utils.h"
#include "../../Host/Host/contexte_tools.h"
#include <iostream>
#include <string>
using namespace std;
struct xmlify {
   int level = {};
   string indent() const {
      return string(static_cast<string::size_type>(level * 2), ' ');
   }
   string operator()(const contexte &ctx) {
      string result = indent();
      if (ctx.terminal())
         result += ctx.name().text() + '\n';
      else {
         result += surround_with(ctx.name().text(), string{"<"}, string{">\n"});
         ++level;
         for (auto & c : ctx)
            result += (*this)(c);
         --level;
         result += indent() + surround_with(ctx.name().text(), string{"</"}, string{">\n"});
      }
      return result;
   }
};
int main() {
   auto ctx = "host:agent:3{temperature{unit{Celsius}precision{0.0001}value{20.5}}}"_ctx;
   cout << indenter(ctx) << '\n';
   cout << indenter(make_canonical(ctx)) << '\n';
   cout << xmlify{}(ctx) << '\n';
   cout << xmlify{}(make_canonical(ctx)) << endl;
}
```

Figure 24 - Traduction de XML à Contexte et inversement

Dans cet exemple, le Contexte est converti en une forme « texte » simple qu'un outil consommant du XML peut traiter.

L’opération inverse n’est pas nécessairement aussi immédiate, notre format de Contexte exigeant l’unicité du nom dans son Contexte, ce que XML n’impose pas. Cependant, les interactions susceptibles de poser problème ici sont de deux ordres :

- consommer des interfaces de services; et
- consommer des données.

Cette fonctionnalité est utilisée dans des projets en cours visant l’interopérabilité entre ContextAA et des technologies tierces.

Annexe K
Traduire de Contexte à JSON et inversement

Le passage de Contexte à JSON est trivial, les deux formats constituant une représentation structurée où les éléments ne se chevauchent pas. Le petit programme décrit à la figure 25 montre qu'il est possible de réaliser cette conversion en quelques lignes à peine, incluant l'indentation et le formatage du code JSON :

```
#include "../../Host/Host/contexte.h"
#include "../../Host/Host/text_utils.h"
#include "../../Host/Host/contexte_tools.h"
#include <iostream>
#include <string>
using namespace std;
struct jsonify {
   int level = {};
   string indent() const {
      return string( static_cast<string::size_type>(level * 2), ' ' );
   }
   string format_name(const string &s) {
      return enquote(strip_surrounding_quotes(s));
   }
   string operator()(const contexte &ctx) {
      string result = indent() + format_name(ctx.name().text()) ;
      if (ctx.terminal())
         result += ";\n";
      else if (ctx.size() == 1 && ctx.front().terminal())
         result += " : " + format_name(ctx.front().name().text()) + ";\n";
      else {
         result += " : {\n";
         ++level;
         for (auto & c : ctx)
            result += (*this)(c);
         --level;
         result += indent() + "}\n";
      }
      return result;
   }
};
int main() {
   auto ctx = "host:agent:3{temperature{unit{Celsius}precision{0.0001}value{20.5}}}"_ctx;
   cout << indenter(ctx) << '\n';
   cout << indenter(make_canonical(ctx)) << '\n';
   cout << jsonify{}(ctx) << '\n';
   cout << jsonify{}(make_canonical(ctx)) << endl;
}
```

Figure 25 - Traduire de Contexte à JSON et inversement

Dans cet exemple, le Contexte est converti en une forme « texte » simple qu'un outil consommant du JSON peut traiter.

L'opération inverse n'est pas nécessairement aussi immédiate, du fait que JSON supporte des tableaux de manière « native », que ce le Contexte ne supporte pas (pour le moment). Deux palliatifs simples sont possibles à court terme, soit :

- traiter les tableaux JSON comme des chaînes de caractères délimitées par des guillemets dans le Contexte; ou
- appliquer une numération explicite aux éléments du tableau (ce qui peut être coûteux en termes d'espace).

Cette fonctionnalité est utilisée dans des projets en cours visant l'interopérabilité entre ContextAA et des technologies tierces.

Annexe L
Automatiser l'interface avec des services tiers

Pour ContextAA, interopérer avec des technologies tierces signifie définir une interface entre la démarche « pur Contexte » d'une part et celle, typiquement orientée services, de son homologue. Pour fonctionner, cette interface doit exposer au tiers orienté services une interface modélisant l'accès au Contexte sous forme de services, et doit exposer la description des services à ContextAA sous forme de Contexte.

Comme le démontrent les annexes J et K, la transformation bidirectionnelle entre le Contexte et les formats protocolaires les plus répandus ne représente pas de réel défi technique; dans les projets en cours qui les utilisent, cette traduction n'a pas été un obstacle. Il est à noter que, dans une perspective de support de plateformes à faibles ressources, ContextAA fait certains choix économiques comme celui de ne pas supporter la pleine et entière gamme des caractères Unicode, ce qui peut représenter un obstacle technique dans certains cas, particulièrement pour interopérer avec des technologies utilisant cette famille d'encodage. En ceci, ContextAA paie pour avoir fait le choix de la frugalité.

Rédiger manuellement des mécanismes de traduction entre Contexte et formats tiers est la solution adoptée jusqu'ici pour assurer l'interopérabilité entre ces technologies. Les travaux en cours sur ContextAA visent toutefois à automatiser la production de ces mécanismes, pour faire en sorte qu'un Hôte placé face à un service Web exposé sous forme WSDL ou ReST par exemple puisse faire apparaître ce tiers comme un Hôte offrant une gamme de Contexte décrite par les services qu'il expose.

L'approche poursuivie implique le recours à des objets autonomes modélisant chacun une technologie tierce distincte, de telle sorte que le modèle de communication pour un Hôte demeure inchangé. L'objet autonome a pour responsabilité de produire le pont technologique entre la technologie tierce et le Contexte, et de jouer le rôle d'intermédiaire entre les deux.

L'approche générale a fait ses preuves par le passé, incluant dans le cas des *Com-Callable Wrappers* [24] et des *Runtime-Callable Wrappers* [95] faisant le pont entre technologies natives et prises en charge sur certaines plateformes. En automatisant la génération des intermédiaires, ContextAA vise à se rapprocher de la fluidité permise par le recours à des intermédiaires dynamiques (*Dynamic Proxies*) comme ceux du monde des services Web.

Annexe M
Implémentation de σ et de ϱ

L'implémentation de σ utilisée dans ContextAA va comme suit :

```
rationnel structural_distance(contexte c0, contexte c1) {
   if (c0.terminal() != c1.terminal()) return { 1 };
   const auto name_part = rationnel{ 1 } - equivalence(c0.name(), c1.name());
   if (c0.terminal()) return name_part; // c1.terminal() implicitement vrai
   if (c0.size() > c1.size() || (c0.size() == c1.size() && c0.name() > c1.name()))
      swap(c0, c1);
   auto best_struct = [](const contexte &ctx, const contexte::container_type &v) -> rationnel {
      if (v.empty()) return { 1 }; // ICI: assert(!v.empty())?
      const auto res = structural_distance(ctx, v.front());
      return accumulate(next(begin(v)), end(v), res, [&](auto && so_far, const contexte &c) {
         return min(so_far, structural_distance(ctx, c));
      });
   };
   const auto structure_part = rationnel{ 1 } - rationnel{ c0.size(), c1.size() };
   const auto value_part =
      accumulate(begin(c0), end(c0), rationnel{},
                 [&, value = c1.value()](auto && so_far, const contexte &c) {
         return so_far + best_struct(c, value);
      }) / rationnel{ c0.size() };
   return moyenne<rationnel>(name_part, structure_part, value_part);
}
```

L'implémentation de ϱ utilisée dans ContextAA va comme suit :

```
rationnel content_wise_distance(const contexte &c0, const contexte &c1) {
   const auto name_part = name_distance(c0, c1);
   if (c0.terminal()) return name_part;
   auto best_equiv = [](const contexte &ctx, const contexte::container_type &v) {
      if (v.empty()) return rationnel{ 1 };
      auto it = begin(v);
      const auto res = content_wise_distance(ctx, *it);
      return accumulate(next(it), end(v), res, [&](auto && so_far, const contexte &c) {
         return min(so_far, content_wise_distance(ctx, c));
      });
   };
   const auto value_part =
      accumulate(begin(c0), end(c0), rationnel{},
                 [&, value = c1.value()](auto && so_far, const contexte &c) {
      return so_far + best_equiv(c, value);
   }) / rationnel{ c0.size() };
   return moyenne<rationnel>(name_part, value_part);
}
```

Le type `rationnel` utilisé dans chaque cas modélise un rationnel normalisé (donc $\frac{4}{6}$ y est modélisé par $\frac{2}{3}$). Le type `contexte` représente le Contexte de ContextAA.

Bibliographie

[1] Abdulrazak, B., Chikhaoui, B., Vallerand, C.G. and Fraikin, B. 2010. A standard ontology for smart spaces. *International Journal of Web and Grid Services*. 6, 3 (2010), 244. DOI:https://doi.org/10.1504/ijwgs.2010.035091.

[2] Abdulrazak, B., Roy, P., Gouin-Vallerand, C., Giroux, S. and Belala, Y. 2010. Macro and micro context-awareness for autonomic pervasive computing. *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services - iiWAS '10* (New York, New York, USA, 2010), 427.

[3] Atzori, L. papers/Ubiquitous computing in the real world-lessons learnt from large scale R. deployments. pdf., Iera, A. and Morabito, G. 2010. The Internet of Things: A survey. *Computer Networks*.

[4] Baldauf, M., Dustdar, S. and Rosenberg, F. 2007. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*. 2, 4 (2007), 263. DOI:https://doi.org/10.1504/IJAHUC.2007.014070.

[5] Bardram, J.E. 2005. Design, Implementation, and Evaluation of the Java Context Awareness Framework (JCAF). *Context*. April (2005), 1–14.

[6] Bardram, J.E. 2005. Tutorial for the Java Context Awareness Framework (JCAF), version 1.5. *Context*. April (2005).

[7] van Benthem, J. and Pacuit, E. 2010. Temporal Logics of Agency. *Journal of Logic, Language and Information*. 19, 4 (2010), 389–393. DOI:https://doi.org/10.1007/s10849-009-9120-y.

[8] Bisgaard, J.J., Heise, M. and Steffensen, C. 2004. How Is Context and Context-awareness Defined and Applied? A Survey of Context-awareness. *Aalborg university*. (2004), 31–40.

[9] Biswas, J. et al. 2011. From context to micro-context - Issues and challenges in sensorizing smart spaces for assistive living. *Procedia Computer Science*. 5, (2011), 288–295. DOI:https://doi.org/10.1016/j.procs.2011.07.038.

[10] Bosse, S. 2015. Unified distributed computing and co-ordination in pervasive/ubiquitous networks with mobile multi-agent systems using a modular and portable agent code processing platform. *Procedia Computer Science*. 63, (2015), 56–64. DOI:https://doi.org/10.1016/j.procs.2015.08.312.

[11] Bosse, T., Duell, R., Memon, Z.A., Treur, J. and Van der Wal, C.N. 2014. Agent-Based Modeling of Emotion Contagion in Groups. *Cognitive Computation*. 7, 1 (2014), 111–136. DOI:https://doi.org/10.1007/s12559-014-9277-9.

[12] Boukhechba, M. et al. 2017. A novel Bluetooth low energy based system for spatial exploration in smart cities. *Expert Systems with Applications*. 77, (2017), 71–82. DOI:https://doi.org/10.1016/j.eswa.2017.01.052.

[13] Brookshier, D., Govoni, D., Krishnan, N. and Soto, J.C. 2002. *JXTA : Java P2P programming*. Sams.

[14] de Bruijn, J. and Polleres, A. 2004. Towards an Ontology Mapping Specification Language for the Semantic Web. (2004).

[15] Castells, M. 1998. *La Société en réseaux. Tome 1 : L'ère de l'information*.

[16] Castillejo, E., Almeida, A. and López-de-Ipiña, D. 2014. Modelling users, context and devices for adaptive user interface systems. *International Journal of Pervasive Computing and Communications*. 10, 1 (2014), 69–91. DOI:https://doi.org/10.1108/IJPCC-09-2013-0028.

[17] Chandy, K.M. and Lamport, L. 1985. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*. 3, 1 (Feb. 1985), 63–75. DOI:https://doi.org/10.1145/214451.214456.

[18] Chen, G. and Kotz, D. 2001. A Survey of Context-Aware Mobile Computing Research. Dartmouth College.

[19] Chen, G. and Kotz, D. 2002. Context Aggregation and Dissemination in Ubiquitous Computing Systems. *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (Washington, DC, USA, 2002), 105--.

[20] Chen, H., Finin, T. and Joshi, A. 2003. *An intelligent broker for context-aware systems*.

[21] Chlamtac, I. and Conti, M. 2003. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks*. 1, (2003), 13–64. DOI:https://doi.org/10.1016/S1570-8705(03)00013-1.

[22] Chmiel, K., Gawinecki, M., Kaczmarek, P., Szymczak, M. and Paprzycki, M. 2005. Efficiency of JADE Agent Platform. *Scientific Programming*. 13, 2 (2005), 159–172. DOI:https://doi.org/10.1155/2005/973963.

[23] Clark, J.H. and Ashby, W.R. 1965. *An Introduction to Cybernetics*.

[24] COM Callable Wrapper: 2017. *https://docs.microsoft.com/en-us/dotnet/framework/interop/com-callable-wrapper*.

[25] Contexte: *http://www.linternaute.com/dictionnaire/fr/definition/contexte/*.

[26] Coulson, G. et al. 2008. A Generic Component Model for Building Systems Software. *ACM Trans. Comput. Syst.* 26, 1 (2008), 1:1--1:42. DOI:https://doi.org/10.1145/1328671.1328672.

[27] Danziger, M. 2008. *Information Visualization for the People*. Massachusetts Institute of Technology.

[28] Dey, A.K. and Abowd, G.D. 2000. Towards a better understanding of context and context-awareness. *Workshop on who, where, when, and how of context-awareness*. 4, (2000), 1–6.

[29] Dey, A.K., Abowd, G.D. and Salber, D. 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human–Computer Interaction*. 16, 2–4 (Dec. 2001), 97–166. DOI:https://doi.org/10.1207/S15327051HCI16234_02.

[30] Dey, A.K. and K., A. 2001. Understanding and Using Context. *Personal and Ubiquitous Computing*. 5, 1 (Feb. 2001), 4–7. DOI:https://doi.org/10.1007/s007790170019.

[31] Dijkstra, E.W. 1972. The humble programmer. *Communications of the ACM*. 15, 10 (1972), 859–866. DOI:https://doi.org/10.1145/355604.361591.

[32] Erfani, M., Zandi, M., Rilling, J. and Keivanloo, I. 2016. Context-awareness in the software domain—A semantic web enabled modeling approach. *Journal of Systems and Software*. 121, (2016), 345–357. DOI:https://doi.org/10.1016/j.jss.2016.02.023.

[33] Faculty, T.A., Dey, A.K. and Fulfillment, I.P. 2000. Providing Architectural Support for Building Context-Aware Applications. *Sensors Peterborough NH*. 16, November (2000), 97–166. DOI:https://doi.org/10.1207/S15327051HCI16234_02.

[34] Fielding and Thomas, R. 2000. Architectural styles and the design of network-based software architectures. (2000).

[35] FIPA 2017. FIPA / The Foundation for Intelligent Physical Agents. (2017), http://fipa.org/.

[36] Fujii, A. 2008. Trends and Issues in Research on Context Awareness Technologies for a Ubiquitous Network Society. *Science and Technology Trends*. Quarterly, 26 (2008). DOI:https://doi.org/doi=10.1.1.584.8235.

[37] Gouin-Vallerand, C. and Giroux, S. 2007. Managing and deployment of applications with OSGi in the context of Smart Homes. *3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2007*. (2007). DOI:https://doi.org/10.1109/WIMOB.2007.4390864.

[38] Greenwood, D. 2004. *FIPA The Foundation for Intelligent, Physical Agents*.

[39] Gruber, T.R. A translation approach to formal ontologies. *Knowledge Acquisition*. 5, 25, 199–200.

[40] Grubert, J., Langlotz, T., Zollmann, S. and Regenbrecht, H. 2017. Towards pervasive augmented reality: Context-awareness in augmented reality. *IEEE Transactions on Visualization and Computer Graphics*. 23, 6 (2017), 1706–1724. DOI:https://doi.org/10.1109/TVCG.2016.2543720.

[41] Guttman, E., Perkins, C., Veizades, J. and Day, M. 1999. *Service Location Protocol, Version 2*.

[42] Henricksen, K., Indulska, J., McFadden, T. and Balasubramaniam, S. 2005. Middleware for distributed context-aware systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2005), 846–863.

[43] Herlihy, M.P. and Wing, J.M. 1990. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*. 12, 3 (1990), 463–492. DOI:https://doi.org/10.1145/78969.78972.

[44] Hewitt, C., Bishop, P. and Steiger, R. 1973. A Universal Modular ACTOR Formalism for Artificial Intelligence. *Proceeding IJCAI'73 Proceedings of the 3rd international joint conference on Artificial intelligence* (1973), 235–245.

[45] Hofer, T. et al. 2003. Context-awareness on mobile devices - The hydrogen approach. *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, HICSS 2003*. (2003). DOI:https://doi.org/10.1109/HICSS.2003.1174831.

[46] Indulska, J. and Sutton, P. 2003. Location management in pervasive systems. *ACSW Frontiers '03 Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*. 21, (2003), 143–151. DOI:https://doi.org/10.1145/830000/828003.

[47] Intille, S.S. 2004. A New Research Challenge: Persuasive Technology to Motivate Healthy Aging. *Technology*. 8, 3 (2004), 235–237. DOI:https://doi.org/doi:10.1.1.123.3807.

[48] Iso.org 2016. *Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1*.

[49] ISO/IEC 2014. *ISO/IEC 9834-8:2014 Information technology -- Procedures for the operation of object identifier registration authorities -- Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers*.

[50] ISO/IEC 14977 1996. *Information technology - Syntactic metalanguage - Extended BNF*.

[51] ISO JTC 1/SC 22 2017. *ISO/IEC 14882:2017 Programming languages -- C++*.

[52] J. Fidge, C. 1988. Timestamps in message-passing systems that preserve partial ordering. *Proceedings of the 11th Australian Computer Science Conference*. 10, (1988), 56–66.

[53] Jacob, C., Linner, D., Radusch, I. and Steglich, S. 2007. Loosely coupled and context-aware service provision incorporating the quality of rules. *ICOMP 07: Proceedings of the 2007 International Conference on Internet Computing* (2007).

[54] JAVA Agent DEvelopment Framework: *http://jade.tilab.com/*.

[55] Jayaputera G., L.S.Z.A. 2003. Mission impossible? Automatically assembling agents from high-level task descriptions. *Proceedings - IEEE/WIC International Conference on Intelligent Agent Technology, IAT'03*. (2003), 161–167.

[56] Kalfoglou, Y. and Schorlemmer, M. 2003. Ontology mapping: The state of the art. 2, (2003).

[57] Kara, N. and Dragoi, O.A. 2007. Reasoning with contextual data in telehealth applications. *3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2007*. (2007). DOI:https://doi.org/10.1109/WIMOB.2007.4390863.

[58] Kephart, J.O. and Chess, D.M. 2003. The vision of autonomic computing. *Computer*. 36, 1 (Jan. 2003), 41–50. DOI:https://doi.org/10.1109/MC.2003.1160055.

[59] Kim, E. and Choi, J. 2007. A context-awareness middleware based on service-oriented architecture. *Ubiquitous Intelligence and Computing, Proceedings*. 4611, (2007), 953–962. DOI:https://doi.org/10.1007/978-3-540-73549-6_93.

[60] Kjaer, K.E. 2007. A Survey of Context-Aware Middleware. *Proc. Software Engineering 2007, ACTA*. (2007), 148–155. DOI:https://doi.org/10.1109/MPRV.2002.1012334.

[61] Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H. and Malm, E.-J. 2003. Managing context information in mobile devices. *IEEE Pervasive Computing*. 2, 3 (Jul. 2003), 42–51. DOI:https://doi.org/10.1109/MPRV.2003.1228526.

[62] Lamport, L. and Leslie 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*. 21, 7 (Jul. 1978), 558–565. DOI:https://doi.org/10.1145/359545.359563.

[63] Lee, E.A. and A., E. 2009. Computing needs time. *Communications of the ACM*. 52, 5 (May 2009), 70. DOI:https://doi.org/10.1145/1506409.1506426.

[64] Lieberman, H. and Selker, T. 2000. Out of context: Computer systems that adapt to, and learn from, context. *IBM Systems Journal*. 39, 3.4 (2000), 617–632. DOI:https://doi.org/10.1147/sj.393.0617.

[65] Marc Weiss, John Eidson, Charles Barry, David Broman, Leon Goldin, Bob Iannucci, Edward A. Lee, K.S. 2015. *Time-Aware Applications, Computers, and Communication Systems (TAACCS)*.

[66] Maxfield, C. and Julien, C. 2017. Data-directed contextual relevance in the IoT. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*. (2017), 43–46. DOI:https://doi.org/10.1109/ICSE-C.2017.13.

[67] McCarthy, J. and Buvač, S. 1997. Formalizing Context (Expanded Notes). *Theory & Psychology*. 15, 1 (1997), 128–131. DOI:https://doi.org/10.1177/095935430501500107.

[68] Mills, D.L. 2006. Computer network time synchronization: the Network Time Protocol. *Group*. (2006), 286.

[69] Moraïtis, P., Petraki, E. and Spanoudakis, N.I. 2003. Engineering JADE Agents with the Gaia Methodology. *Workshop on "Agent Technology and Software Engineering" (AgeS 2002)*. 2592, (2003), 77–91. DOI:https://doi.org/10.1007/3-540-36559-1.

[70] Nawrocki, P., Śniezyński, B. and Czyzewski, J. 2016. Learning agent for a service-oriented context-aware recommender system in heterogeneous environment. *Computing and Informatics*. 35, 5 (2016), 1005–1026.

[71] Neovius, M., Sere, K., Yan, L. and Satpathy, M. 2006. A formal model of context-awareness and context-dependency. *Proceedings - 4th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2006*. (2006), 177–185. DOI:https://doi.org/10.1109/SEFM.2006.2.

[72] Nwana, H.S. 1996. Software agents: An overview. *The knowledge engineering review*. 11, 03 (1996), 205–244. DOI:https://doi.org/10.1017/S026988890000789X.

[73] Olaru, A., Florea, A.M. and El Fallah Seghrouchni, A. 2013. A context-aware multi-agent system as a middleware for ambient intelligence. *Mobile Networks and Applications*. 18, 3 (2013), 429–443. DOI:https://doi.org/10.1007/s11036-012-0408-9.

[74] Ontology: 2009. *http://tomgruber.org/writing/ontology-definition-2007.htm*.

[75] Ontology related concepts: *http://www.webkb.org/kb/ontology.html*.

[76] OSGi Concepts: *http://www.eclipse.org/virgo/documentation/virgo-documentation-3.7.0.M01/docs/virgo-user-guide/html/ch02s02.html*.

[77] OWL 2 Web Ontology Language Document Overview (Second Edition): 2012. *http://www.w3.org/TR/owl2-overview/*.

[78] Panagiotakis, S. 2012. Context-Awareness via Ubiquitous User Profiling: An Implementation Paradigm. *International Journal of Software Engineering*. 2, 3 (2012), 77–90. DOI:https://doi.org/10.5923/j.se.20120203.05.

[79] Picon, A. 2016. L'avènement de la ville intelligente. *Societes*. 132, 2 (2016), 9–24. DOI:https://doi.org/10.3917/soc.132.0009.

[80] Plesa, R. and Logrippo, L. 2007. An Agent-Based Architecture for Context-Aware Communication. *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on* (2007), 133–138.

[81] Ponce, V. and Abdulrazak, B. 2017. Activity model for interactive micro context-aware well-being applications based on ContextAA. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 10461 LNCS, (2017), 99–111. DOI:https://doi.org/10.1007/978-3-319-66188-9_9.

[82] Ponce, V., Roy, P. and Abdulrazak, B. 2017. Dynamic Domain Model for Micro Context-Aware Adaptation of Applications. *Proceedings - 13th IEEE International Conference on Ubiquitous Intelligence and Computing, 13th IEEE International Conference on Advanced and Trusted Computing, 16th IEEE International Conference on Scalable Computing and Communications, IEEE Internationa*. (2017), 98–105. DOI:https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0036.

[83] Pratistha, I. M., Zaslavsky, A., Cuce, S. and Dick, M. 2005. Improving Operational Efficiency of Web Services with Mobile Agent Technology. *IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (2005).

[84] Rabari, C. and Storper, M. 2015. The digital skin of cities: Urban theory and research in the age of the sensored and metered city, ubiquitous computing and big data. *Cambridge Journal of Regions, Economy and Society*. 8, 1 (2015), 27–42. DOI:https://doi.org/10.1093/cjres/rsu021.

[85] Rakotonirainy, A. and Maire, F. 2005. Context-aware driving behavioural model. *19th International Technical Conference on the Enhanced Safety of Vehicles* (2005), 1–6.

[86] Rashvand, H.F. and Hsiao, K.F. 2013. Smartphone intelligent applications: a brief review. *Multimedia Systems*. 21, 1 (2013), 103–119. DOI:https://doi.org/10.1007/s00530-013-0335-z.

[87] Rodríguez, L.F. and Ramos, F. 2012. Computational models of emotions for autonomous agents: major challenges. *Artificial Intelligence Review*. 43, 3 (2012), 437–465. DOI:https://doi.org/10.1007/s10462-012-9380-9.

[88] Rodriguez, N.D. 2011. A Framework for Context-Aware Applications for Smart Spaces. *2011 IEEE/IPSJ International Symposium on Applications and the Internet* (2011), 218–221.

[89] Rogers, Y. 2006. Moving on from Weiser's Vision of Calm Computing: Engaging UbiComp Experiences. (2006), 404–421. DOI:https://doi.org/10.1007/11853565_24.

[90] Rose, K., Eldridge, S. and Chapin, L. 2015. THE INTERNET OF THINGS: AN OVERVIEW. Understanding the Issues and Challenges of a More Connected World. *The Internet Society*. October (2015), 80. DOI:https://doi.org/10.5480/1536-5026-34.1.63.

[91] Rotem-Gal-Oz, A. 2006. Fallacies of distributed computing explained. *URL http://www. rgoarchitects. com/Files/fallacies. pdf*. (2006), 11.

[92] Roy, N., Julien, C., Misra, A. and Das, S.K. 2016. Quality and Context-Aware Smart Health Care: Evaluating the Cost-Quality Dynamics. *IEEE Systems, Man, and Cybernetics Magazine*. 2, 2 (2016), 15–25. DOI:https://doi.org/10.1109/MSMC.2015.2501163.

[93] Roy, P., Abdulrazak, B. and Belala, Y. 2008. Approaching context-awareness for open intelligent space. *6th International Conference on Advances in Mobile Computing and Multimedia*. (2008), 422–426. DOI:https://doi.org/10.1145/1497185.1497276.

[94] Roy, P., Abdulrazak, B. and Belala, Y. 2015. Quantifying Semantic Proximity Between Contexts. *Smart Homes and Health Telematics: 12th International Conference, ICOST 2014, Denver, CO, USA, June 25-27, 2014, Revised Papers*. C. Bodine, S. Helal, T. Gu, and M. Mokhtari, eds. Springer International Publishing. 165–174.

[95] Runtime Callable Wrapper: 2017. *https://docs.microsoft.com/en-us/dotnet/framework/interop/runtime-callable-wrapper*.

[96] Russell, S.J. and Norvig, P. 2003. Artificial Intelligence - A Modern Approach - 2nd Edition. (2003), 1109.

[97] Ryan, N., Pascoe, J., and Morse, D. 1999. Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant. *Computer Applications in Archaeology* (1999), 269–274.

[98] Sadalage, P. and Fowler, M. 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*.

[99] Salim, F. and Haque, U. 2015. Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and Internet of Things. *International Journal of Human Computer Studies*. 81, (2015), 31–48. DOI:https://doi.org/10.1016/j.ijhcs.2015.03.003.

[100] Schilit, B., Adams, N. and Want, R. 1994. Context-aware computing applications. *IEEE W rkshop on Mobile Computing Systems and Applications*. (1994), 1–7. DOI:https://doi.org/10.1109/MCSA.1994.512740.

[101] Schmohl, R. and Baumgarten, U. 2008. Context-aware computing: A survey preparing a generalized approach. *IMECS 2008 Int. MULTICONFERENCE Eng. Comput. Sci. VOLS I II; 744-750 2008*. I, (2008), 744–750.

[102] Sheehy, J. 2015. There is No Now Problems with simultaneity in distributed systems. (2015), 1–8.

[103] Sheng, Q.Z. and Benatallah, B. 2005. ContextUML: A UML-based modeling language for model-driven development of context-aware Web services. *4th Annual International Conference on Mobile Business, ICMB 2005*. (2005), 206–212. DOI:https://doi.org/10.1109/ICMB.2005.33.

[104] Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence*. 60, 1 (1993), 51–92. DOI:https://doi.org/10.1016/0004-3702(93)90034-9.

[105] Spanoudakis, N. and Moraitis, P. 2008. An Ambient Intelligence Application Integrating Agent and Service-Oriented Technologies. *Interface*. (2008), 393–398. DOI:https://doi.org/10.1007/978-1-84800-094-0.

[106] Stepanov, A.A. and McJones, P. 2009. *Elements of Programming*. Addison-Wesley Professional.

[107] Strang, T. and Linnhoff-Popien, C. 2004. A Context Modeling Survey. *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing* (2004), 1–8.

[108] Tao, M., Zuo, J., Liu, Z., Castiglione, A. and Palmieri, F. 2018. Multi-layer cloud architectural model and ontology-based security service framework for IoT-based smart homes. *Future Generation Computer Systems*. 78, (2018), 1040–1051. DOI:https://doi.org/10.1016/j.future.2016.11.011.

[109] Technology for Enabling Awareness: 2000. *http://www.teco.edu/tea/tea_vis.html*.

[110] The eight fallacies of distributed computing: 2001. *http://wiki.aardrock.com/images/2/21/Network_Fallacies.pdf*.

[111] URN Namespaces: 2001. *http://www.w3.org/TR/uri-clarification/#urn-namespaces*.

[112] Vila, X., Schuster, A. and Riera, A. 2007. Security for a Multi-Agent System based on JADE. *Computers and Security*. 26, 5 (2007), 391–400. DOI:https://doi.org/10.1016/j.cose.2006.12.003.

[113] Weiser, Mark 1991. The Computer for the 21st Century. *Scientific american*. 265, 3 (1991), 94–104.

[114] Weiser, M. and Brown, J.S. 1996. Designing Calm Technology. 1, 1 (1996), 75–85. DOI:https://doi.org/10.1.1.135.9788.

[115] Winograd, T. 2001. Architectures for context. *Hum.-Comput. Interact.* 16, 2 (2001), 401–419. DOI:https://doi.org/10.1207/S15327051HCI16234_18.

[116] Yaiz, R.A., Selgert, F. and Den, H.F. 2006. On the definition and relevance of Context-Awareness in Personal Networks. *2006 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, MobiQuitous*. (2006). DOI:https://doi.org/10.1109/MOBIQ.2006.340427.

[117] Yang, H., Lim, S., King, J. and Helal, A. 2006. Open Issues in Nomadic Pervasive Computing. *Proceedings of UbiSys 2006, the Workshop on System Support for Ubiquitous Computing*. (2006).

[118] Yu, Z., Zhou, X., Yu, Z., Zhang, D. and Chin, C.-Y. 2006. An OSGi-based infrastructure for context-aware multimedia services. *IEEE Communications Magazine*. 44, 10 (Oct. 2006), 136–142. DOI:https://doi.org/10.1109/MCOM.2006.1710425.

[119] Zaslavsky, A. 2004. Mobile agents: Can they assist with context awareness? *Proceedings - 2004 IEEE International Conference on Mobile Data Management*. (2004), 304–305. DOI:https://doi.org/10.1109/MDM.2004.1263080.

[120] Zheng, Y., Capra, L., Wolfson, O. and Yang, H. 2014. Urban Computing: Concepts, Methodologies, and Applications. *ACM Transactions on Intelligent Systems and Technology*. 5, 3 (2014), 1–55. DOI:https://doi.org/10.1145/2629592.