Plateforme robotique ubiquitaire pour la supervision sémantique et contextualisée des personnes dépendantes

Sofiane Bouznad & Lyazid Sabri
LISSI, Université UPEC
bouznad.sofiane@gmail.com, lyazidsabri@gmail.com

Abdelghani Chibani & Yacine Amirat
LISSI, Université UPEC
chibani,amirat@u-pec.fr

François Bremond
STARS, INRIA Sophia Antipolis
Francois.Bremond@inria.fr

RESUMÉ

Dans ce papier nous présentons une plateforme de supervision sémantique des usagers dans un espace intelligent qui est mise en œuvre en combinant la plateforme d'analyse de scènes et de reconnaissance d'activités physiques SUP avec la plateforme de supervision sémantique à base d'ontologies SembySem. L'objectif étant de répondre d'une part aux besoins de mise en ouvre de services d'assistance sensible au contexte de l'usager et d'autre part décharger les concepteurs des contraintes de mise en œuvre imposées par les systèmes à intelligence ambiante en général et les robots ubiquitaires en particulier. Pour les besoins de validation nous présentons un scénario mettant en oeuvre un agent de supervision contextuelle permettant de détecter des situations anormales et déclencher des actions quand il s'agit de situations d'urgence. La validation de ce scénario est effectuée en utilisant la plateforme *ubistruct* développée au laboratoire LISSI, incluant un robot compagnon et des capteurs intelligents.

MOTS CLÉS

Robotique ubiquitaire, Supervision, Sensibilité au contexte, Ontologies, Raisonnement réactif.

1. INTRODUCTION

Le chevauchement qui existe actuellement entre la robotique, l'informatique ubiquitaire et en particulier l'intelligence ambiante, fait que leur utilisation conjointe constitue un choix synergétique pour transformer des espaces de vie telle que la maison, le lieu travail, en espaces intelligents, centrés sur des robots service, permettant d'améliorer le cadre de vie des usagers et les assister dans leurs différentes tâches en tout lieux et en tout instant. On parle alors de robotique ubiquitaire. Il s'agit ici d'un nouveau domaine de recherche en pleine expansion dans lequel on crée des services d'assistance exploitant des objets communicants: des capteurs, actionneurs, terminaux numériques, artefacts intelligents et des robots de service de toute sorte. L'apparition récente du concept de l'informatique en nuage (cloud computing) va dans le même sens et va permettre l'émergence d'une nouvelle génération de robots ubiquitaires capables d'enrichir leurs capacités cognitives et partager leurs connaissances en se connectant à des infrastructures cloud [1] [2] [3].

La robotique ubiquitaire ouvre ainsi la voie à un nombre illimité d'applications mettant en œuvre des compagnons physiques et virtuels pour aider les personnes dans leur vie quotidienne et travailler à leurs côtés; ils agiront en tant que gardes physiques et/ou virtuels autonomes pour protéger les personnes, surveiller leur sécurité et leur venir en aide dans les espaces intérieurs et extérieurs. Selon cette nouvelle vision de la robotique, un robot ubiquitaire sera susceptible aussi de servir d'incarnation pour l'interaction avec les humains.

Dans ce papier nous présentons une plateforme de supervision sémantique des usagers dans un espace intelligent qui est mise en œuvre en combinant la plateforme SUP [4] avec la plateforme SembySem [5]. L'objectif étant de répondre d'une part aux besoins de mise en ouvre de services d'assistance sensible au contexte de l'usager et d'autre part décharger les concepteurs des contraintes de mise en œuvre imposées par les systèmes à intelligence ambiante en général et les robots ubiquitaires en particulier.

Le papier est organisé de la façon suivante. Dans la section 2 nous présentons l'architecture de la plateforme proposée pour mettre en œuvre les agents de supervision. Dans la section 3 nous décrivant la mise en œuvre d'un modèle d'agent de supervision contextuelle permettant de détecter des situations anormales et déclencher des actions quand il s'agit de situations d'urgence. La validation de ce travail est effectuée en utilisant la plateforme ubistruct [7] développée au laboratoire LISSI, incluant un robot compagnon et des capteurs intelligents. Une démonstration vidéo de l'expérimentation est disponible en suivant le lien (https://www.dropbox.com/sh/3jo9vjeclatocz2/AACgox8twsu8Kp60fckNiv9qa). Dans la section 4, nous présentons un travail encours concernant la modélisation des propriétés temporelles des observations contextuelles et leur prise en compte dans le processus d'inférence.

2. ARCHITECTURE

L'architecture de la plateforme est composée de trois gros modules qui sont basée sur un modèle sémantique de description de l'environnement ambiant. Le noyau de raisonnement est le module central de cette architecture. Il est en charge de gérer la base de connaissance d'un agent de supervision, ceci inclut d'une part la gestion d'insertion, la mise à jours et la suppression des faits à partir des observations des objets capteurs, et d'autre part le déclenchement d'actions dans monde réel avec des objets actionneurs. Le noyau de raisonnement intègre un moteur d'inférence permettant d'exécuter des règles d'inférence en mode chainage avant selon l'algorithme RETE. Il intègre aussi un sous module de vérification de contraintes d'intégrités, qui sont définies au niveau du modèle ontologique. Ceci permet de garantir d'une part la cohérence des inférences effectuées par le moteur d'inférence et d'autre part la bonne interprétation des informations échangées avec le monde réel. Les opérations de gestion la base de connaissances sont effectuées de manière synchrone avec les messages échangés avec les objets de l'environnement ambiant. Ainsi, l'inférence d'une action est suivi

automatiquement d'un envoie de message à un objet actionneurs qui prendra en charge l'exécution effective de cette action. Les communications entre le module noyau et les objets du monde réel sont prises en charge par le module Façade. Ce module est un intergiciel orienté message qui permet d'abstraire les détails de mise en œuvre de la communication avec les objets de l'environnement ambiant. Son implémentation est basée sur le standard JMS Java Messaging Services. Un mécanisme d'encodeurs de messages est utilisé pour transformer d'une part les données des capteurs sous forme de données structurées, pouvant être intégré dans la base de connaissance ontologique, et d'autre part les instances d'actions en commandes compréhensibles par les objets de l'environnement ambiant selon leurs implémentations respectives. Enfin, l'architecture de la plateforme est conçue comme un mécanisme de plug-in permettant l'insertion de modules supplémentaires selon une approche inspiré de l'environnement Eclipse. Cependant, l'extension de l'architecture pour supporter des modèles d'intégration tels qu'OSGi et iPOJO reste possible.

2.1 Modèle sémantique de l'environnement ambiant

Le modèle sémantique de description de l'environnement ambiant permet de décrire d'une manière riche les observations concernant la situation ou l'activité de l'utilisateur ainsi que tous les objets se trouvant dans son voisinage ou avec lesquels il est en interaction. Les informations d'état de ces objets permettent de décrire le contexte de chaque situation ou activité de l'utilisateur. Pour ce faire nous proposons une micro ontologie pour la robotique ubiquitaire qui est définie dans le langage µConcept [6]. Ce dernier est une variante du langage le web sémantique OWL2/RDF. Il permet de décrire sémantiquement les objets de l'environnement ambiant et les informations de contexte avec des notions de base suivantes: µConcept, propriété, instance et action. Un µConcept est similaire à la notion de classe dans OWL. µConcept est décrit sémantiquement à travers des relations avec d'autres µConcept et/ou avec des attributs littéraux. Une relation ou un attribut sont exprimées par le biais de la notion de propriété binaire. Comparé avec OWL2/RDF, une propriété d'un µConcept peut être aussi une propriété multi-value ou une propriété ordonnées. D'une manière similaire aux bases de données et inversement à OWL, une instance d'un µConcept est définie selon l'hypothèse du nom UNA. Cette hypothèse permet d'identifier d'une manière unique un objet de l'environnement ambiant. Une action est un élément conceptuel qui n'existe pas dans le standard web sémantique ; Il permet de définir des actions/commandes que chaque instance d'une entité du modèle sémantique est en mesure d'exécuter.

La gestion de la base de connaissances est effectuée par le biais de règles d'inférence, qui sont définies avec le langage *SmartRules*, voir figure 1. Ce dernier est indissociable du langage µConcept et permet de définir des règles de production du type: *Si condition(s) Alors exécuter des opération(s)/actions(s) sur les instances.*

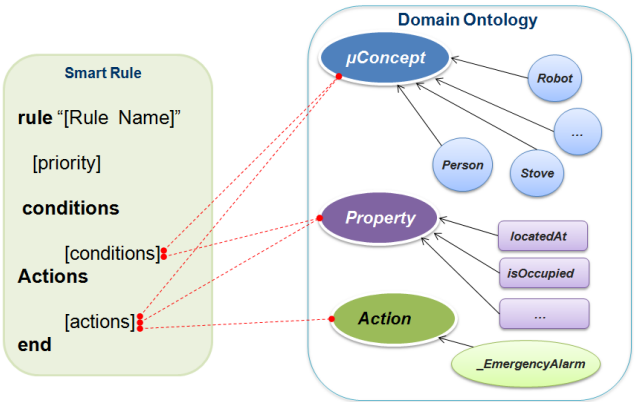

Figure 1. Schéma général d'une règle SmartRules

2.1.1 Supposition du nom unique - UNA

La représentation des objets de l'environnement ambiant dans une base de connaissance ontologique nécessite de respecter une contrainte importante qui est celle de l'utilisation d'une instance unique pour désigner / identifier chaque objet présent dans le monde réel. Ceci permet à un moteur d'inférence d'éviter la création de plusieurs instances de concepts et de propriétés associées à la même entité. Bien que cette contrainte soit implémentée dans la base de données à l'aide des clés, elle n'est pas prise en compte dans les ontologies du web sémantique. En effet, dans RDF/S ou OWL, les langages phares du web sémantique qui se basent sur la notion d'URI, si deux instances ont deux URIs différentes, cela ne signifie pas que ces deux instances soient différentes. Pour pallier au problème ci-dessus, l'alternative proposée est d'utiliser les propriétés OWL *sameAs* ou *differentFrom*. La propriété *sameAs* peut être utilisée pour spécifier que des instances de la même classe sont similaires. La propriété *differentFrom* exprime le fait que des instances du même concept sont différentes. Bien que les propriétés *differentFrom* et *sameAs* constituent une solution au problème soulevé ci-dessus, conserver toutes les instances associées à un capteur peut s'avérer coûteux en temps de traitement. En effet, il est nécessaire de vérifier, à chaque fois, que toutes les instances créées sont identiques ou différentes. La conservation de ces instances dans la base de connaissances peut engendrer des contradictions si elles ne sont pas estampillées explicitement avec le même référentiel de temps, et dans ce cas, les actions réactives qui peuvent être déclenchées vont être incohérentes.

2.1.2 Hypothèse du monde fermé - CWA

Les contraintes imposées par les applications d'intelligence ambiante en termes de dynamisme du contexte et de cohérence dans la prise de décision, exigent l'utilisation d'un système de raisonnement qui fonctionne selon l'hypothèse du monde fermé. Pour ce faire, nous utilisons le langage de règles *SmartRules* en association avec une micro ontologie. *SmartRules* permet de modéliser un processus de raisonnement réactif permettant de prendre une décision suite à la reconnaissance du contexte courant dans lequel se trouve l'utilisateur. Nous distinguons deux types de règles *SmartRules*: des règles d'inférence du contexte courant et des règles pour déclencher des actions réactives par le biais du robot ou des effecteurs disponibles dans l'environnement.

2.2 Modélisation sémantique du contexte

Le modèle proposé pour la description sémantique du contexte est basée sur quatre concepts de haut niveaux - Observation, Objet, Robot et Espace, qui sont articulés autours du concept Person. Le concept objet permet de décrire d'une part des objets physiques (capteurs et actionneurs ou robots) et des objets logiciels tels que des documents multimédia, des applications et des services web. Un objet peut être de trois types décrits à travers les concepts suivants: *ActiveObject*, *StaticObject* et *MovableObject*. Le concept robot permet de décrire les caractéristiques et les fonctionnalités de contrôle du robot. Le concept Robot peut être spécialisé pour définir plusieurs catégories de robots de service, comme par exemple un robot compagnon mobile, un robot d'aide à la locomotion, etc. Le concept *SpaceRegion* permet de décrire la composition de l'espace physique dans lequel seront localisés les objets et les usagers.

Les situations de l'usager sont inférées à partir des observations des capteurs. Elles peuvent être classées en trois catégories : situations normales, situations anormales ou situations d'urgences. Le contexte d'une situation est décrit en utilisant la propriété *contextOf* ou *hasContext*. Ces deux relations permettent de lier un concept décrivant une situation comme par exemple "chute grave" à partir des observations qui sont effectués à partir des objets capteurs. Pour ce faire nous considérons les concepts *Observation* et *SensorOuput*. Ce dernier est utilisé pour convertir les données des capteurs en instances de haut niveau qui peuvent désigner des observations quantitatives ou des observations qualitatives.

Notre modèle prend en compte seulement les dernières instances valides des propriétés afin d'éviter de recourir à l'annotation de chaque entité avec les propriétés *sameAs* ou *differentFrom* pour gérer une forme d'unicité de description des objets, ce qui va compliquer la définition des règles. Ce qui fait que seulement ces instances seront conservées dans la mémoire de travail du moteur d'inférence pour être utilisée par les règles de raisonnement.

2.3 Objets Virtuels

L'objet virtuel est utilisé comme un élément d'abstraction de la mise en œuvre des opérations effectives qui peuvent être effectuées par/sur des objets de type capteur, actionneur ou robot. Un objet virtuel est considéré comme une boite noire contrôlable par le noyau de raison Core via la couche Façade. Chaque objet virtuel est représenté dans la couche Core par un µConcept. Un objet virtuel peut être déployé au niveau de la plateforme de l'objet physique si cette dernière possède suffisamment de ressources pour prendre en charge les opérations de l'objet virtuel. Par ailleurs, il peut être déployé au niveau de plateforme d'accueil connecté au web. Le capteur virtuel, une fois déployé et mis en marche, s'enregistre au près de la façade. Cet enregistrement se fait à travers l'envoi d'un message XML vers le noyau de raisonnement contenant les propriétés du µconcept qu'il le décrit. Le message d'enregistrement contient en plus des paramètres pour permettre à la façade de dialoguer avec le capteur virtuel notamment par l'envoi de commandes pour exécuter des actions de l'environnement ambiant. Une fois l'enregistrement effectué, le capteur virtuel commence à transmettre les données des capteurs sous forme de messages XML. L'architecture d'un objet virtuel est définie selon un canevas spécifique, illustrée dans la figure 5, qui est composée de trois modules.

Le module accès à la source permet d'implémenter le protocole de lecture des données du capteur réel qui peut être un capteur physique connecté avec un port série, une base de données ou un système externe. L'accès à un système externe peut être effectué en implémentant un code propriétaire à ce dernier ou peut être avec un protocole applicatif standard tel que JMS, XMPP ou REST. Le module bootstrap de transmission est chargé de la gestion de la communication avec la façade. Trois options sont offertes au concepteur d'un objet virtuel pour gérer les communications avec la façade qui dépendent du cas de déploiement de ce dernier. La première option concerne les objets capteurs. Elle permet d'assurer que la fonction de formatage et renvoie des données brutes des capteurs vers la façade sans traitement particulier. Cette option est envisagée quand la plateforme physique liée au capteur ne permet pas de supporter des traitements de transformation des données des capteurs. La deuxième option permet d'encoder ou décoder les données de la façade. Ce qui permet d'éviter décharger la façade de fonctions d'encodage spécifiques. La dernière option nécessite que l'objet ait suffisamment de ressources de traitement pour charger l'ontologie dans son espace mémoire afin de traiter directement les données des capteurs ou les actions sous forme d'instances de concepts de l'ontologie. Cette option est envisagée quand l'objet virtuel est déployé dans le cloud et le débit de communication entre la source et l'objet virtuel est suffisent et la source est un système implémentant un protocole applicatif standard tel que XMPP, JMS ou REST.

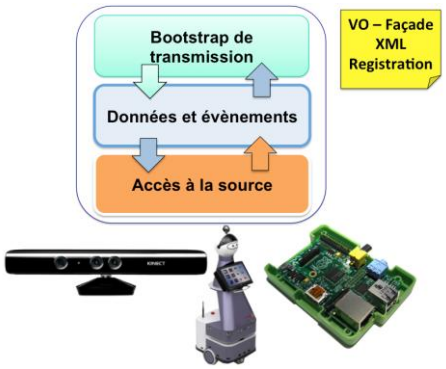

Figure 2. Architecture de mise en œuvre d'un objet virtuel

2.3.1 Capteur SUP

SUP (*Scene Understanding Platform*) est une plateforme de reconnaissance d'activités à partir de flux vidéo, en l'occurrence des flux 3D qui peuvent être récupérée depuis des caméras RGB-D telles que Microsoft Kinect, développé par l'équipe STAR de l'INRIA [4]. La plateforme SUP est encapsulée en tant qu'un capteur virtuel dont l'architecture, elle est composée de trois modules, voir figure 3 :

-Le Module Vision : permet l'indentification et la poursuite d'objet mobile dans la scène, en utilisant des algorithmes de traitement d'images (segmentation, classification, etc.)

-Le Module de Reconnaissance d'événement : permet la reconnaissance des scénarios, relatifs aux activités des objets mobiles évoluant dans la scène, définis dans la base de connaissances.

-La base de connaissances : fournie d'une part des informations 3D de la scène et d'autre part la description par des experts des événements d'intérêts.

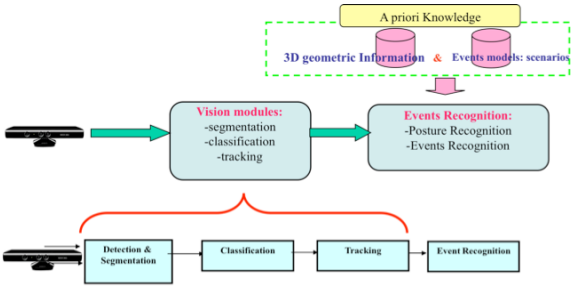

Figure 3. Architecture interne de SUP

La plateforme SUP est utilisé dans notre plateforme pour détecter trois types d'événements dans une scène fixe: posture et changement de posture de la personne, sa position dans la scène, sa proximité par rapport aux objets définis dans la scène.

2.3.2 Robot de service

Il s'agit ici d'un objet virtuel qui permet d'encapsuler les fonctions et services implémentés au niveau de la plateforme d'un robot mobile. Le noyau de raisonnement agit comme un module de contrôle de haut niveau pour adapter de manière transparente l'exécution des services d'assistance selon le contexte courant. La notion de service d'assistance correspond d'une part à une tache physique complexe qui est exécutée par le robot dont l'objectif est d'aider l'utilisateur ou à des fonctions logicielles.

3. AGENT DE SUPERVISION CONTEXTUELLE D'UNE PERSONNE DEPENDANTE

Un agent de supervision contextuelle peut reconnaitre des situations non-triviales en agrégeant directement des connaissances contextuelles obtenues à parti des observations des capteurs. Dans cette section nous présentons le processus d'inférence par lequel un agent de supervision est capable de reconnaître différents types de situations de l'usager. En effet, une situation peut-être caractérisée de diverse manière selon le contexte courant. Ce dernier est constitué d'un ensemble d'observations transmises au noyau de raisonnement de l'agent par le biais des capteurs virtuels inscrits à la façade. Le processus d'inférence de l'agent de supervision permet aussi de détecter et de réagir aux situations anormales observées, soit en interagissant avec l'usager soit en agissant directement sur les objets d'environnement ambiant, voir figure 4. En effet, une situation anormale peut donner lieu à une situation d'urgence si elle est confirmée; un mécanisme de levée de doute basé sur l'interaction avec l'usager est proposé pour détecter ce type de situation.

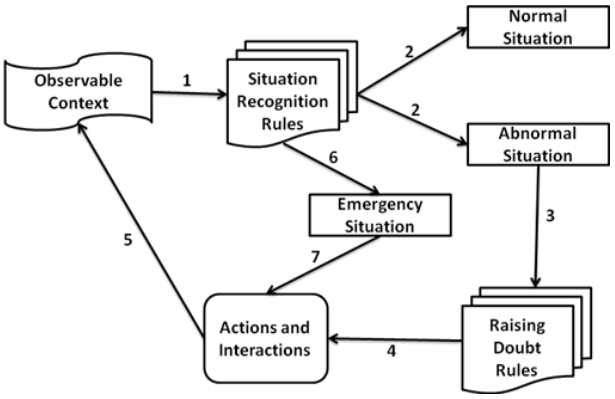

Figure 4. Etapes de Reconnaissance de situations

3.1 Scénario

Considérons le cas de personnes en perte d'autonomie, nous nous intéressons à la détection de la situation de chute. Cette situation est d'abord détectée par l'agrégation de contextes observables, elle est classée comme situation anormale qui doit être confirmée. La levée de doute est déclenchée en conséquence est consiste en une ou plusieurs actions d'interactions avec l'usager ; soit en utilisant des dispositifs d'affichage, soit un robot compagnon pour confirmer si la personne est consciente ou non.

Pour ce faire nous avons mis en œuvre une plateforme d'expérimentation avec différents composants hétérogènes:

- SUP connecté à une camera Kinect pour l'identification de la posture et du mouvement de la personne ainsi que la suivit de ses déplacements.
- Un système de localisation indoor (Cricket) pour le suivit de certains objets mobiles de l'environnement (Fauteuil, table, etc.). Un ensemble de capteurs permettant d'observer des évènements liés au contexte d'une personne âgée à domicile (capteurs de présence, capteur de chute, etc).
- Des dispositifs d'affichages dotés de hauts parleurs.
- Un robot compagnon équipé d'un dispositif d'affichage et d'un système de reconnaissance et synthèse vocale.

Sans être exhaustifs, nous présentons dans ce papier deux types de contextes de chute qui peuvent être prise en charge par des règles génériques indépendantes des technologies de capteurs utilisées.

3.2 Premier contexte

Dans le premier contexte, la situation de chute est détectée par l'agrégation des contextes suivant:

- La posture couchée de l'usager
- La position de l'usager
- L'absence d'un objet de repos (fauteuil ou lit) dans cette position

Les deux premiers contextes sont directement observables par des capteurs virtuels, SUP dans notre scénario ; alors que le troisième

contexte est inféré par la négation du contexte de présence d'objets dans une zone (négation par l'échec).

3.2.1 Identifier la posture de l'usager

La posture de l'usager est détectée par la plateforme SUP conformément à la définition de l'événement composé suivant :

```
CompositeState(Person_LyingDown,
   PhysicalObjects((p1 : Person))
   Components ((c1: PrimitiveState Lying(p))
   Constraints ((c1 duration > 5))
   Alarm ((Level : URGENT)))
```

Le capteur virtuel SUP génère et envoi un événement vers la façade à chaque changement de posture, la règle qui suit est alors déclenchée et met à jour le modèle sémantique.

rule "PostureObservation"
conditions
PostureObservation(?posture:=hasQualityValue);
?inhabitant:= Person();
actions
?inhabitant->hasPosture:= ?posture;
update(?inhabitant);
end

Figure 2. Détection de la marche et de postures effectuée par SUP

3.2.2 Déterminer la localisation de l'usager

La plateforme SUP est aussi utilisée pour la détection de la localisation de l'habitant. Cette localisation est qualitative elle permet de renseigner la présence de l'usager dans une zone prédéfinie dans l'ontologie. La définition de l'événement localisation dans SUP est comme suit :

```
PrimitiveState(Person_Inside_ZoneA,
   PhysicalObjects((p1 : Person), (z1 : Zone))
   Constraints ((p1->Position in z1->Vertices)
   Alarm ((Level : URGENT))
)
```

Le capteur virtuel génère et envoi un événement vers la façade à chaque changement de position, la règle qui suit est alors déclenchée et met à jour le modèle sémantique.

rule "LocationObservation"
conditions
LocationObservation(?location:=hasQualityValue);
?inhabitant:= Person();
actions
?inhabitant->hasLocation:= ? location;
update(?inhabitant);
end

En effet, la propriété *hasLocation* permet d'associer, à un haut niveau sémantique, une localisation dans un espace donné à l'usager représenté dans l'ontologie *AmiOnt* par le concept Person.

3.2.3 Identifier la localisation des objets déplaçables

Pour inférer l'absence d'une observation de proximité à un mobilier dédié pour l'activité repos, dans la position de chute, le système doit poursuivre la localisation des objets déplaçable dans l'environnement. La règle suivante permet la mise à jour de la propriété *hasLocation* pour le concept *MovaleObject*; la nouvelle localisation étant détectée par notre système de localisation indoor Cricket. La liste d'objets présents dans cette localisation, définie par la propriété *contains*, est elle aussi mise à jour.

rule «Update Movable Objects Location»
conditions
LocationSensorOutput(?virtual_object:=isProducedBy,
?location:=hasQualityValue)
?virtual_object(?object:=isReferenceOf,?object
isInstanceOf(MovableObject));
actions
?object ->hasLocation:=? location;
addPropertyValue(?location->contains,?object);
update(?object);
update(?location);
end

3.2.4 Détecter la situation de chute

La règle suivante permet d'agréger les contextes posture, localisation, et la négation du contexte présence d'objet (*Chair* ou *Bed*) dans l'espace localisé, pour inférer la situation anormal de Chute. Une instance de Chute est alors créer et insérer dans le modèle sémantique, avec une propriété *locatedAt* pour garder trace de l'endroit de cette situation.

rule " Fall Situation in Context 1"
conditions
Person(?location:=hasLocation,
hasPosture==‘LyingDown’);
?location (?object := one (contains)) ;
(not?object (isInstanceOf(Chair))and
not?object (isInstanceOf(Bed)))
actions
FallSituation ?fall := **createInstance**(FallSituation);
?fall->locatedAt:=?location;
insert(?fall);
end

(z1->Name = ZoneA))

3.2.5 Deuxième contexte

Dans ce contexte, la situation de chute est détectée par un dispositif permettant la détection des événements de haut niveau (ZCare Celode™). La règle ci-après permet de mettre à jour le modèle sémantique.

Figure 2. Evénement de chute détecté par Zcare

rule « Fall Situation in Context 2»
conditions
?ac:= FallSensorOutput (?virtual_object:=isProducedBy);
Person(?location:=hasLocation);
actions
Fall Situation ?fall := **createInstance**(Fall Situation);
?fall->observedBy:=?virtual_object;
?fall->observationResult:=?ac;
?fall->locatedAt:=?location;
insert(?fall);
end

3.3 Mécanisme de levée de doute

Le mécanisme de levée de doute proposé pour le scénario de la chute permet de réduire considérablement les faux positives de situations d'urgence. Ce mécanisme est décrit par deux règles génériques chacune faisant référence à une action d'interaction avec l'usager. Ces deux règles sont complémentaires.

La première règle est conditionnée par l'existence d'un dispositif d'interaction de type *MultimediaDevice* (téléphone portable, écran de PC, écran du robot, synthétiseur vocal du robot, écran TV) prés de l'usager, la propriété *nearTo* nous renseigne sur cette proximité.

rule « Fall Reaction 1»
conditions
FallSituation ();
Person(?deviceId := nearTo);
?deviceId (isInstanceOf (MultimediaDevice)) ;
actions
VisualMessage?message:=**createAction**(?deviceId,
VisualMessage);
?message->content:= "If you fill good, press Ok";
execute(?message);
end

La deuxième règle est exécutée en cas d'échec de déclenchement de la première règle (absence d'un dispositif d'interaction de type *MultimediaDevice* prés de l'usager), elle consiste à l'envoi du robot prés de la position de chute. L'envoi du robot est aussi conditionné par l'accessibilité à l'espace dans lequel a été observée la chute. Une fois le robot présent sur place, la propriété *nearTo* est mise à jour et la première règle se déclenchera. La deuxième règle est définie comme suit :

rule "Fall Reaction 2"
conditions
FallObservation (?location := locateAt,
?location ->isAccessibleSpace);
Person(not (nearTo isInstanceOf (MultimediaDevice)));
?robot:= CompanionRobot();
actions
MoveRobot ?action:= **createAction**(?robot, MoveRobot);
?action->moveTo:= ?location;
execute(?action);
end

La réponse de l'usager permet la levée de doute sur la situation anormale inférée. En effet, dans notre scénario, l'usager répond aux fausses alertes par un événement traduit par une observation de type *ButtonObservation* ayant comme valeur "*False Alarm*". L'absence de ce type de contexte dans une situation anormale de chute permet d'inférer une situation d'urgence qui donnera lieu à l'envoi d'une alerte (alerter les pompiers et les proches par exemple).

Un temps d'attente doit néanmoins être observé entre l'inférence de la situation de chute et la décision de levée de l'alerte. Deux techniques permettent de réaliser ce temps d'attente. La première technique consiste à utiliser un paramètre *duration* permettant de définir le délai fixe en secondes pour retarder l'exécution de la règle et de ré-exécuter la règle après expiration de ce délai, si les conditions de la règle sont toujours vérifiées.

rule "Fall Emergency 1"
duration 20
conditions
?fall := FallSituation (?location := locateAt);
Not exists(ButtonObservation(
observationResult->hasQualityValue=="False alarm"));
actions
FallEmergency ?alarm:= **createInstance**(FallEmergency);
? alarm ->locatedAt:=?location;
insert(?alarm);
retract(?fall);
end

La deuxième technique préconise la définition explicite du paramètre temps d'attente, dans les conditions de la règle, par le biais du concept *DeadlineExpiryObservation*. Un événement d'expiration de délai d'attente est alors généré par le capteur virtuel représentant le dispositif d'interaction.

rule "Fall Emergency 2"
conditions
?fall := FallSituation (?location := locateAt);
not exists (ButtonObservation(
observationResult->hasQualityValue==’false alarm’);
?timer:=DeadlineExpiryObservation(
observationResult->hasQualityValue==’fall deadline’)) ;
actions
FallEmergency ?alarm:= **createInstance**(FallEmergency);
?alarm ->locatedAt:=?location;
insert(?alarm);
retract(?fall);
retract(?timer);
end

La situation d'urgence doit aussi être inférée en cas d'échec de levée de doute. Dans notre scénario, l'absence de dispositifs d'interaction et l'inaccessibilité de l'espace est considéré comme échec de levée de doute pour la situation anormal de chute, qui sera traduit par considérer comme situation d'urgence.

rule "Fall Emergency 3"
conditions
?fall := FallObservation (?location := locateAt,

not (?location ->isAccessibleSpace));
Person(not (nearTo isInstanceOf (MultimediaDevice)));

actions

FallEmergency ?alarm:= **createInstance**(FallEmergency);
? alarm ->locatedAt:=?location;
insert(?alarm);
retract(?fall);

end

4. TRAVAUX ENCOURS

La notion du temps est très importante si on veut mettre en œuvre un processus de raisonnement qui tient compte des relations de précédence entre les observations. Ainsi la reconnaissance contextuelle d'une situation est mieux élaborée si nous lui associant des contraintes temporelles, par exemple l'activité *PrepareBreakfast* se déroule généralement le matin alors que *PrepareDinner* le soir. Une activité (ou situation) peut-être aussi défini par un ordre partielle ou totale des contextes (observations) qui la compose, par exemple dans l'observation *OvenUsed* peut être défini par trois événement *OvenOpenOnservation*, *OvenClosedObservation* et *PowerConsumptionObservation* et des contraintes temporelles entre ces trois événements.

Pour appliquer le raisonnement temporel dans notre plateforme SEMbySEM, nous avons ajouté quelques propriétés à l'ontologie *AmIOnt*. De plus, nous avons implémenté dans notre raisonneur les relations d'Allen (en plus de la relation Duration qui permet d'informer sur la durée d'une observation ou activité), pour raisonner sur ces propriétés temporelles.

- timeOfDetection : c'est une propriété de type Date permettant de dater les événements (instances) du concept *SensorOutput*

- hasStatTime, hasEndTime : ces deux propriétés permettant de définir explicitement l'intervalle de validité d'une observation ou activité. Les relations d'Allen s'appliquent sur ces propriétés.

- hasDuration : une propriété permettant de préciser la durée moyenne d'une activité. Cette propriété est importante pour raisonner (agréger ou fusionner) sur les événements qui se sont produit pendant toute la durée de l'activité.

- hasOccurrenceTime : cette propriété permet de définir le moment de la journée de l'occurrence d'une activité (matin, midi, après midi, soir, ou toute la journée), l'activité *PrepareBreakfast* se déroule le matin. Cette propriété permet de filtrer les activités possibles lors du processus de raisonnement.

L'exemple ci-après nous montre comment on peut utiliser les propriétés énumérées ci-dessus pour représenter l'action d'utilisation du frigo par le biais d'un capteur d'ouverture de porte.

rule "UsingFridge Activity"

conditions

?ac:=DoorStatusOutput(hasQualityValue=="opened",
?virtual_object:=isProducedBy,
?dTime :=timeOfDetection);
?virtual_object (?appliance := installedIn);
?appliance (**isInstanceOf** (Fridge));

actions

UsingFridge ?activity := **createInstance**(UsingFridge);
? activity ->observedBy:=?virtual_object;
? activity ->observationResult:=?ac;
? activity ->hasStartTime:=?dTime;
? activity ->hasEndTime:=null;
update(?activity);

end

5. REMERCIEMENTS

Nous remercions Sofiance Ait Arab et Lotfi Zaouche pour leur préciseuse aide dans la mise en oeuvre de la démonstration.

6. REFERENCES

[1] K. Kamei, S. Nishio, N. Hagita, and M. Sato. "Cloud networked robotics", Network, IEEE, volume 26, pages 28 – 34, 2012.

[2] H. Guoqiang, P.T. Wee, and W. Yonggang. "Cloud robotics: architecture, challenges and applications", Network, IEEE, volume 26, pages 21–28, 2012.

[3] J. A. Kuffner. "A Crowd of Quantum", Ieee Spectrum 48, pages 16–18, 2011.

[4] https://team.inria.fr/stars/software/sup/

[5] J.S. Brunner, J.F. Goudou, P. Gatellier, J. Beck, and C.E. Laporte. "SEMbySEM : a framework for sensor management", In Proc. of the 1st Int. Workshop on the Semantic Sensor Web (SemSensWeb), 2009.

[6] L. Sabri, A. Chibani, Y. Amirat and G. P. Zarri. G. P. Semantic reasoning framework to Supervise and Manage Contexts and Objects in pervasive computing environments; AINA Workshops, pages 47-52, 2011.

[7] ubistruct living lab, http://ubistruct.ubiquitous-intelligence.eu