Vers une modélisation graphique des applications ubiquitaires basée sur un Dsml intelligent : Covid-19 Contact-Tracer

Mohammed Fethi Fethi Khalfi, Mohammed Nadjib Tabbiche, Réda Adjoudj

Vers une modélisation graphique des applications ubiquitaires basée sur un Dsml intelligent : Covid-19 Contact-Tracer

Mohammed Nadjib TABBICHE [1], Mohammed Fethi KHALFI[2], ADJOUDJ Réda [3]
{nadjib.tabbiche@gmail.com}{fethi.khalfi,adjreda}@yahoo.fr
EEDIS laboratory, Dept. of computer science. Djillali Liabes University. Sidi Bel Abbes, Algeria

RESUME : Les environnements ubiquitaires ne sont pas figés dans le temps. Les entités de ce contexte évoluent constamment : elles sont dynamiques. Ce comportement est imprévisible car il est impossible de contrôler la disponibilité des entités du contexte. Les applications ubiquitaires ont alors fortement besoin de s'adapter durant leur exécution et réagir aux changements que le contexte peut subir, développer des applications ubiquitaires reste encore complexe. L'Ingénierie Dirigée par les Modèles (IDM) se présente comme une approche prometteuse pour résoudre cette complexité.

Les langages dédiés (DSL) sont de plus en plus utilisés parce qu'ils permettent aux utilisateurs qui ne sont pas des experts en programmation d'exprimer des solutions avec des langages dédiés graphiques ou textuelles qui ont un niveau d'abstraction plus élevé que les langages dédiés de programmation.

Dans cet article nous proposons de définir une syntaxe concrète sous la forme d'une représentation graphique basée sur un DSML permettant de guider et aider un expert de tout domaine d'ingénierie à définir et à formaliser une syntaxe concrète graphique. Nous l'illustrons notre approche dans le domaine de l'informatique ubiquitaire par la mise en œuvre d'un prototype d'application qui permet de suivre les déplacements des personnes testées positives pour anticiper les contaminations du COVID-19.

Mots clés : Applications ubiquitaires, IDM, Éditeurs Graphiques, COVID-19, DSML, Syntaxe concrète, Machine Learning.

1 INTRODUCTION

Le terme «informatique ubiquitaire» a été proposé pour la première fois en 1991 par Weiser [1] dans son article fondateur intitulé "The Computer for the 21st Century" [1] : que les technologies les plus réussies sont celles qui disparaissent. En effet, elles s'associent à la vie de tous les jours jusqu'à ce qu'il devienne difficile de les discerner. Cette vision décrit des environnements remplis de dispositifs informatiques miniaturisés et intégrés dans les objets du quotidien couplés à des infrastructures de communications. Ce couplage est réalisé dans le but de délivrer des services adéquats, de la manière la plus transparente, naturelle et efficace possible.

Les applications ubiquitaires sont des nouveaux types d'applications. Elles doivent être capables de s'adapter dynamiquement aux changements du contexte car celui-ci évolue dynamiquement en réaction à la disponibilité des multitudes objets communicants et de leurs mobilités. Les modèles du contexte proposés évoluent avec les technologies proposées sur le marché, Ainsi, une nouvelle vision est nécessaire pour minimiser ces dépendances technologiques, Cela nécessitent d'utiliser des démarches d'ingénierie (Architecture Dirigée par les Modèles) et l'usage de modèles génériques de haut niveau supportés par des métamodèles qui facilitent la conception des applications et permettant le support de toutes les caractéristiques ubiquitaires.

Dans notre travail nous donnons la main aux utilisateurs de créer leurs propres représentations de leurs futures applications ubiquitaires. L'objectif consiste à partir d'un modèle graphique exprimé à l'aide d'un DSML composé de symboles et de notations formelles générer le modèle de simulation pour une application ubiquitaire. Pour présenter l'utilisabilité du modèle graphique, Nous nous intéressons au domaine de la pandémie du Coronavirus, particulièrement à un prototype permettant de suivre, contextualiser et anticiper les risques de COVID-19.

Cet article s'organise comme suit : La section 2 introduit brièvement l'état de l'art des systèmes ubiquitaires, les principes de l'approche MDA et DSML. La section 3 détails les diverses contributions de MDA pour la conception d'applications ubiquitaires suivi par une synthèse. La section 4 détaille le model architecturale du système proposé ainsi que l'atelier de modélisation graphique à l'aide de l'outil Eclipse Sirius. La section 5 présente une DSML du métamodèle proposé qui permet de de prévenir les personnes qui ont été en contact avec un malade pour anticiper les risques de COVID-19. Dans la section 6 Nous concluons en établissant le bilan de nos contributions et en présentant les perspectives de recherche de nos travaux futurs.

2 THEORETICAL BACKGROUND : DSML POUR LES APPLICATIONS UBIQUITAIRES

2.1 TECHNOLOGIES OF UBIQUITOUS SYSTEMS :

L'informatique ubiquitaire résulte de la convergence des domaines de l'informatique mobile et des systèmes distribués, Figure1. Les systèmes distribués constituent la rencontre entre les ordinateurs personnels et les réseaux locaux. Ils permettent le partage des capacités et des res-

sources à travers un réseau et une infrastructure de communication. Cela constitue la première étape de l'informatique pervasive par l'introduction de l'omniprésence de l'information.

L'informatique mobile permet à l'utilisateur de gérer l'information depuis un terminal mobile. Elle est le résultat de l'intégration de la technologie cellulaire et du web pour donner la possibilité aux utilisateurs d'accéder à l'information à n'importe quel endroit et en utilisant différents équipements de petites tailles et de faibles coûts [2]. Les ordinateurs sont devenus plus petits et plus performants. Ils sont aussi embarqués dans les objets de la vie quotidienne (objets enrichis de capacités de traitement de l'information). Grâce aux technologies sans fils, les objets sont interconnectés et communiquent pour tout échange d'informations. Nous parlons dans ce cas d'informatique ubiquitaire où l'information est accessible de n'importe où et n'importe quand, [3].

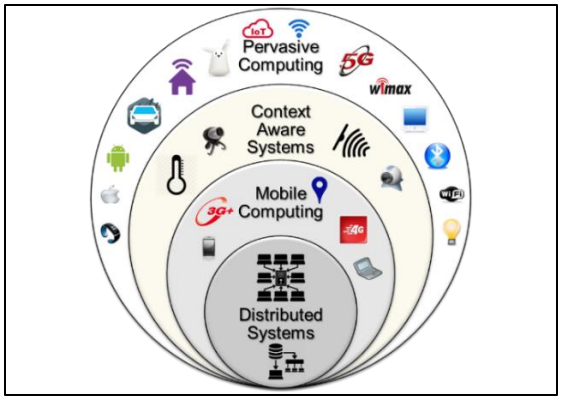

fig. 1 : Développement de l'informatique Ubiquitaire [2].

2.2 PRESENTATION OF MDE (MODEL-DRIVEN ENGINEERING):

Pour faire face à la complexité des applications ubiquitaires, L'Ingénierie Dirigée par les Modèles (IDM), ou Model-Driven Engineering (MDE) à adopter les modèles comme éléments centraux dans le processus de développement d'une application et à automatiser la transformation de ces modèles afin d'aboutir au code source, [13].

Le MDA met en œuvre les concepts IDM à travers une architecture à 4 niveaux illustrée dans la pyramide de la Figure 5.

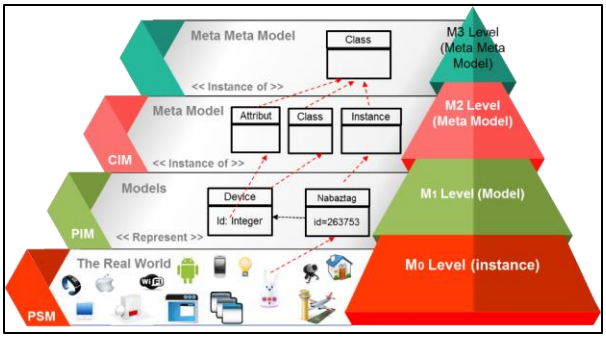

fig. 2 : niveaux du modèle MDA [4].

2.3 LANGAGES DEDIES DE MODELISATION (DSML)

Le principe des langages spécifiques de modélisation ou Domain-Specific Modeling Language (DSML) est présenté pour construire des langages spécifiques aux domaines. Les DSML permettent de créer des modèles et de les manipuler simplement en utilisant des concepts métier. Chacun de ces langages est spécialisé dans un domaine particulier des systèmes. comme HTML qui est un DSL dédié à l'écriture de documents contenant des liens hypertextes, PHP qui est un DSL dédié à la production des pages web et à la gestion des bases de données, SQL qui est un DSL pour interroger ou pour manipuler les bases de données relationnelles,
Les langages dédiés fournissent tous les ´éléments de modélisation n´nécessaires pour répondre `a une problématique précise. Ces langages sont d´définis autour de deux principales syntaxes :

- **la syntaxe abstraite** : c'est la syntaxe utilisée pour d´définir les ´éléments ainsi que les relations entre ces ´éléments, c'est le méta-modèle.

- **la syntaxe concrète** : c'est une syntaxe, graphique ou textuelle, qui représente le système, c'est le modèle.

Un DSML n'a qu'une seule syntaxe abstraite, mais peut avoir plusieurs syntaxes concrètes. La syntaxe concrète fournit à l'utilisateur un ou plusieurs formalismes graphiques et/ou textuels, pour manipuler les concepts de la syntaxe abstraite et ainsi en créer des instances. L'instance ainsi obtenue sera conforme à la structure définie dans la syntaxe abstraite. La syntaxe concrète textuelle est utilisée, principalement, pour les DSL dits < de programmation > alors que la syntaxe concrète graphique est plutôt réservée aux DSML.

3 RELATED WORKS

L'ingénierie logicielle fournit sans cesse de nouvelles technologies facilitant la mise en œuvre des systèmes in-

formatiques. L'IDM est une forme d'ingénierie générative par laquelle tout ou partie d'une application informatique est générée à partir de modèles. Dans cette nouvelle perspective, les modèles occupent une place de premier plan parmi les artéfacts de développement des systèmes. Dans cette section, nous avons cherché à focaliser notre attention sur des exemples représentatifs de processus IDM liés à des domaines d'application ubiquitaires. Nous avons sélectionné des quelques travaux :

Approach			CaaSSE 2020, [5]	CIM-CSS 2019, [6]	SenSoMod 2019, [7]	UbiCARS, UbiRS4Tourism: 2018, 2019 [8,9]	AocML 2018, [4]	CAADA 2018, [5]	B.Boudaa 2016, [11]	LoCCAM 2014,[12]	Our Proposition
Application Domain			Tourism context service (TourismCS)	Smart Meeting Rooms	printer management	physical commerce and tourism recommendation	Smart meeting room	Healthcare	Healthcare SAMU	Android projects	Covid-19
Level of Abstraction	Abstract syntax		EMF (Ecore)	EMF (Ecore) + UMLClass Diagram	EMF (Ecore)	EMF (Ecore)	UMLClass Diagram	EMF (Ecore)	UMLClass Diagram + ODM	EMF (Ecore)	EMF (Ecore)
	Concrete syntax	Textual	✗	✗	✗	✗	AocML DSL	✗	✗	✗	Xtext/EMFText
		Graphical	GMF	GMF	Obeo Designer	GMF	✗	✗	✗	GMF	Sirius
	Graphical Or Textual Tool		CaaSSET graphical modeling tool	Eclipse-based graphical editor for CIM-CSS	SenSoMod-Modeler	UbiCARS Modeling Editor	Xtext	✗	✗	modelagem visual (GMF)	graphical modeling tool
	Transformation Tool		Acceleo : MOF Model to Text Language (MTL)	GMF Dashboard	✗	✗	Xtext, PAOC	AMG (ATLAS Mega Model Managemen t),	JDOM + FraSCAti	Xpand	Acceleo : MOF Model to Text Language
	PIM to PSM		PHP + MySQL scripts	Generation of Java code	Script SenSo-Mod2Java	Generation of files in (XML)	Generation of Java code and ontology	Eclipse Acceleo context model to Java Code	Generation of code FraSCAti	LoCCAM Lib	Graphical modeling tool to XML model
Context management	Context language modeling		EMF (Ecore) Graphical	EMF (Ecore) UML Class Diagrams + OCL	DSML SenSoMod Graphical	DSML for UbiCARS, UbiRS4Tourism Graphical	Specific textual DSL UML & Ontology	XML	Ontology& UML	DSML LoCCAM UML	EMF (Ecore) Graphical
	Context Reasoning Smartness		ECA (Event of condition do action)	First-order logic (FOL)	ECA (Event of condition do action)	CARSKIT Engine	Jena Rule-Based Reasoner	ECA (Event of condition do action)	SWRL (Semantic Web Rule Language)	ECA (Event of condition do action)	Supervised Machine Learning
Adaptation	services		Context Triggering	Context Triggering	Context Triggering	adaptation Algorithms	Context Triggering	Context Triggering	Context Binding, Triggering and Behaviour Adaptation	Context Triggering	Unsupervised Machine Learning
	Smartness		ECA Rules	ECA Rules	✗	Recommendation Engine	proactive context manager	ECA Rules	ECA Rules	Rules IF-THEN-ELSE	Machine Learning

Tab1 – Synthèse des approches de développement des applications ubiquitaires

Afin de caractériser et comparer les travaux liés à notre proposition, nous utilisons Certains critères identifiés sur la base des exigences de notre problème, Table 4 :

• **Tool for development :** Nous allons utiliser l'outil Eclipse Sirius remplaçant avantageusement le calamiteux GMF. Il permet de définir une syntaxe graphique pour un langage de modélisation décrit en Ecore et d'engendrer un éditeur graphique intégré à Eclipse.

• **Context management & Adaptation :**

- Pour la représentation du contexte, certaines approches utilisent les modèles spécifiques (DSL) ou des API dédiées à des applications bien ciblées, des modèles génériques adaptés (ContextUML, UML+OCL …etc.).
- Le métamodèle de contexte proposé ne couvre pas les principales entités contextuelles.
- La plus part des approches ont utilisées le modèle de raisonnement et d'adaptation simple ECA (Event of condition do action). ce que ne satisfait pas une des exigences et la nature des systèmes ubiquitaires qui est l'incertitude et le changement fréquent du contexte. Afin d'améliorer l'efficacité et de rendre ces modèles intelligents. nous avons décidé de combiner l'apprentissage automatique avec EMF

4 PROPOSITION

Notre travail s'intéresse à la conception d'applications ubiquitaires à partir d'une approche IDM. Un environnement de méta-modélisation permet de définir un DSML et d'en générer un éditeur permettant d'exprimer des modèles conformes au langage. En partant de ce constat, Ce processus se fait généralement en plusieurs étapes :

4.1 ARCHITECTURE

Nous présentons l'architecture de notre solution, Figure 6.

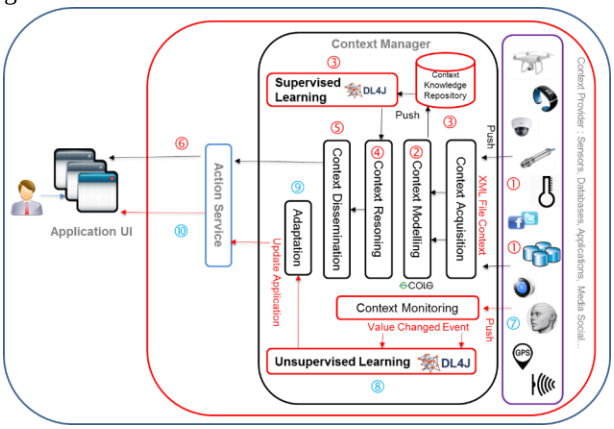

fig 3 : Architecture.

4.2 Un métamodèle de contexte

Dans l'ingénierie dirigée par les modèles, l'utilisation des métamodèles permet d'assurer une syntaxe précise à différents modèles et une sémantique propre à un domaine d'application. La construction des applications ubiquitaires nécessite une modélisation correcte de toutes les informations de contexte afin de les stocker dans un référentiel pour faciliter par la suite leur exploitation. Dans le contexte de l'IDM, la syntaxe abstraite est généralement décrite en premier et sert de base pour spécifier la syntaxe concrète.

4.2.1 Spécification d'une syntaxe abstraite

Dans le cadre du présent travail de recherche, Nous avons opté pour EMF pour définir le métamodèle ou la syntaxe abstraite du langage (exprimé en Ecore). La Figure 4 illustre notre métamodèle. Ecore est un langage graphique de haut niveau qui spécifie la syntaxe abstraite permettant de créer des métamodèles satisfaisant au standard EMOF. Il utilise la syntaxe graphique d'UML qui est relativement bien connue et de ce fait est assez intuitif. Les concepts qu'il fournit (Packages, Classes, associations, attributs) sont assez génériques pour convenir à tous les domaines. On a pensé à rendre ces modèles intelligents en combinant l'apprentissage automatique avec EMF. Le système ubiquitaire est représenté par :

• **L'élément (System)** : il peut contenir d'autres systèmes grâce à la relation «ownedSystem». La relation

l'agrégation «ItsEntity» précise qu'une instance de System est composée de plusieurs entités contextuelles impliquées dans une application ubiquitaire

• **ContexteEntity :** Est la classe centrale de ce métamodèle, elle représente les entités qui s'identifient au système par la relation «itsEntity». Une entité contextuelle est une représentation concrète d'un objet du monde réel jugé pertinent pour décrire un domaine. Elle peut être composée d'autres entités par la relation «ownedEntity».

Le fonctionnement des applications ubiquitaires nécessite la collecte de toutes les informations de contexte pertinentes depuis de diverses sources. Il existe deux catégories de contexte :

- **Contexte atomique** : représente des informations contextuelles de bas niveau qui n'est pas reliée à un autre contexte et elle peut être obtenue par des : Capteurs physiques (GPS, Wi-Fi, Bluetooth, NFC), Capteurs Virtuels (web services), Capteurs sociaux service Flickr. microblog (Twitter),
- **Contexte composite** : représente des informations contextuelles de haut niveau, il se compose de contextes multiples. Il peut être obtenue par des : Logical Sensor

• **ContextSource** : représente les ressources à partir de lesquelles les informations contextuelles sont récupérées. il permet de relier les éléments de contexte à leurs sources pour identifier la manière utilisée pour l'acquisition de la donnée contextuelle. L'acquisition de ces données peut être : Static ("static"), Profilé ("Profiled"), capturée ("sensed") et dérivée d'autres données ("derived")

• **ContextProvider** : est un composant logiciel lié au capteur physique, logique ou social. Le "ContextProvider" fournit une interface uniforme pour récupérer et collectes les informations à partir de (RFID tag, microphone, or GPS), Logical/soft Sensor (e.g., digital clock), online profiles, or Databases et à proposer suivant le mode d'observation "Notify Event".

• **ContextObservable** : décrivent les données collectées par les capteurs. Il contient "Value" est une valeur de type simple (Integer, Long, Float, String ou Boolean) mesurée par le capteur. L'information contextuelle Observable peut être soit atomique, soit composite. L'acquisition de l'information contextuelle peut avoir une contrainte temporelle qui peut être soit un intervalle absolu soit une date d'expiration exprimée dans ce métamodèle par la métaclasse "Constraint".

O **ObservationMode** : Le rôle d'un capteur consiste à effectuer les mesures puis à envoyer les données à son ou ses clients ou à permettre à son ou ses clients de venir les récupérer. L'obtention des données peut se faire selon le mode d'observation ("ObservationMode") Par notification ("Notify" ou "Push"), c'est le capteur qui envoie les données à chaque fois que ces informations changent pour notifier les entités intéressées.

- **NotificationEvent** : Le mode "Notify" se décompose, quant à lui, en trois modes : "SimpleNotify", "PeriodicNotify" et "ConditionalNotify".

• **ContextProperty** décrit les propriétés de la class ContextEntity. Chaque ContextProperty est décrit par son nom et son type.

• **ContextRepository** : Les applications peuvent utiliser non seulement le contexte actuel, mais également le contexte perçu pour adapter leurs comportements afin de mieux interagir avec les utilisateurs. Ainsi, nous stockons les contextes en continu, au fur et à mesure qu'ils se produisent, dans une base de données.

• **ContextAssociation** : est utilisé pour décrire la relation entre deux entités ou la relation qui relie l'entité de contexte à ses propriétés. Chaque association est décrite par un nom.

Généralement, les actions d'adaptation de se basent principalement sur des structures conditionnelles, c'est-à-dire une fois qu'une condition devient vraie, le système doit changer de comportement de la manière convenable. L'inconvénient de ces structures est leur aspect statique, qui ne prend pas en considération la nature dynamique d'un environnement ubiquitaire dû à la mobilité des utilisateurs et des équipements. Les développeurs sont contraints de prévoir toutes les situations et les scénarios possibles durant la définition des contraintes et avant que le système ne soit fonctionné.

• **CAMechanism** : Pour tenir compte de l'incertitude des environnements ubiquitaire cette partie permet la modélisation des actions d'adaptation selon le contexte, et toutes ces actions sont représentées par la classe CAMechanisme., nous avons décidés de retenir la relation stéréotypée « ContextBinding » et introduire les techniques basées sur l'apprentissage non supervisé :

O **ContextBinding** une association direct entre l'information de contexte et les objets sensibles au contexte de la classe CAObject. Ce mapping permet de récupérer automatiquement des informations pour les utilisateurs en fonction de l'information de contexte disponible. Dans exemple Covid19, à chaque fois que le service «Track_Users» est invoqué, il fournit automatiquement des informations correspondant à l'endroit du patient Covid-19 et son contexte localisation (GPS),

O **ContextMachineLearning** : Nous proposant un métamodèle qui regroupe les entités communes du processus de machine Learning. Il comporte un ensemble de techniques basées sur l'apprentissage non supervisé et d'actions à prédire représentées respectivement par les classes «UnSupervised_ML_Algorithm» et «Action_Prediction» dans le méta-modèle.

O **MachineLearning** : cette classe utilise des algorithmes d'apprentissage supervisé et non supervisé pour former des modèles. Ensuite, les résultats de chaque algorithme sont évalués pour déterminer la précision. À la fin, il sélectionne le modèle approprié pour la prédiction.
O **Input** : Les entrées sont un ensemble de données utilisées par les algorithmes d'apprentissage automatique pour entraîner le modèle.
O **Output** : sont un ensemble de données prédites par le modèle entraîné.
O **Score** : Pour vérifier l'exactitude de l'apprentissage et de l'inférence, des mesures d'évaluation (score) sont nécessaires. Les métriques d'évaluation sont utilisées pour la phase de sélection de modèle qui est un processus pour sélectionner un modèle approprié à partir de modèles candidats et à partir d'informations données liées aux performances de chaque modèle. La précision des modèles d'apprentissage automatique doit être évaluée et le modèle le plus performant est sélectionné pour la prédiction.

O **Action Prédiction** : Le résultat de la prédiction est renvoyé pour la prise de décision.
Notre méta modèle peut être aussi étendu en intégrant les aspects confidentialité et respect de la vie privée de l'entité source du contexte. En effet, la collecte des données contextuelles d'une personne (par exemple sa localisation) doit se faire sous l'accord de la personne en question et donc en fonction de ses préférences.

• **Contexte Security** : Le méta-modèle élargi consiste à modéliser des politiques de contrôle d'accès et de confidentialité intégrées dans un processus de développement logiciel dirigé par les modèles. Le méta-modèle spécifie est basé sur les modèles ABAC et RBAC.

On propose d'utiliser les deux modèles RBAC et ABAC sans considérer les rôles comme attributs des utilisateurs. Sinon, le fait de considérer un rôle comme un attribut, fait perdre tous les avantages de RBAC parce qu'on ne peut plus considérer qu'un rôle représente un ensemble de permissions. Notre modèle prend en compte les attributs contextuels de l'Entity Costumer et la ressources du système susceptibles de faire l'objet d'accès.

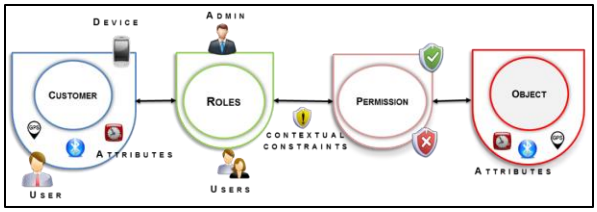

fig 4 : les deux modèles RBAC et ABAC combinés.

la gestion des droits d'accès et le respect de la vie privé sont conditionnés par l'exploitation des données contextuelles mesurées directement par les capteurs embarqués dans l'environnement et d'autres issues de capteurs portés par l'utilisateur Nous l'avons étendu pour être adapté aux changements de l'environnement. Les différentes extensions du modèle portent sur l'intégration des contraintes contextuelles, L'autorisation dynamique nécessite d'être réévaluée en fonction des changements dans les informations contextuelles utilisées pour obtenir l'accès. Cela nécessiterait que le système de sécurité soit averti des changements de contexte tout au long de leur utilisation du système. Dans ce cas, le système réagirait aux changements de contexte en évaluant les règles d'autorisation pour déterminer si le sujet est toujours autorisé.

• **QoC** : Une information contextuelle doit présenter des propriétés de qualité très particulières pour pouvoir être prise en compte d'une manière efficace, elle peut être évaluée selon différents critères : Confiance, Précision, Fraîcheur, Résolution.
Selon notre modèle, chaque critère de QoC peut prendre une valeur comprise entre [0, 1]. Le calcul des métriques de QoC est en fonction du type de source de contexte (Capteurs physiques, Virtuels et sociaux).

• **Modality** : Ces relations permettent de retrouver les périphériques actuellement en interaction avec l'utilisateur mais également de savoir comment ils interagissent avec l'utilisateur.

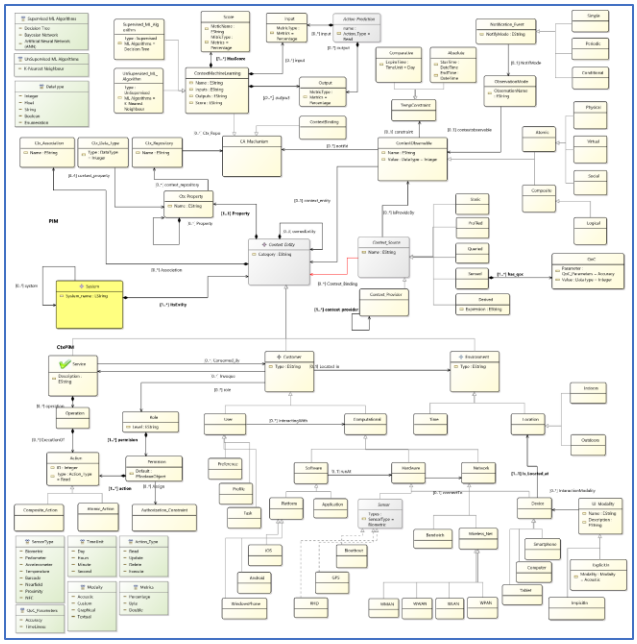

fig 5 : Architecture.

4.3.) LA SYNTAXE CONCRETE (MODELEUR GRAPHIQUE).

4.3 1 Définition et Spécification d'une syntaxe concrète graphique :

Les syntaxes concrètes (CS) d'un langage fournissent à l'utilisateur un ou plusieurs formalismes, textuelle (sous forme de pseudocode), soit graphique (comme les diagrammes UML) selon les besoins des utilisateurs pour manipuler les concepts de la syntaxe abstraite. Elle permet à l'utilisateur d'exprimer de manière formelle un modèle puis d'instancier et de manipuler les concepts et les relations définis dans la syntaxe abstraite. Le modèle ainsi obtenu est conforme au méta-modèle du langage de modélisation.

Notre contribution repose sur la spécification de la syntaxe abstraite formalisée dans le métamodèle présenté dans la section précédente pour définir une syntaxe concrète graphique via l'outillage Sirius qui pourra être adapté à des besoins métier spécifiques.

4.3.2 Eclipse Sirius :

Sirius est un outil Open Source de la Fondation Eclipse développé par les sociétés Obeo et Thales. Le framework permet de concevoir un atelier de modélisation graphique en s'appuyant sur les technologies Eclipse Modeling, en particulier EMF, GEF et GMF qui offre les éléments de base pour construire un éditeur graphique.

L'atelier de modélisation permet aux utilisateurs de créer, éditer et visualiser graphiquement des modèles EMF. Il fournit un outillage pour définir des représentations graphiques telles que les diagrammes, les tableaux ou les arbres avec des règles de validation et actions utilisant des descriptions déclaratives.

Sirius permet de représenter des modèles graphiquement sous la forme d'un graphe composé de nœuds et d'arcs. Il est possible de choisir la forme d'un nœud (rectangle, ellipse, etc.) et la forme d'un arc (décoration à l'extrémité, tracé du trait, etc.).

La grande force de Sirius est de permettre la spécification des éditeurs de manière totalement graphique aux concepts du méta-modèle, sans avoir à écrire de code. Le rendu peut être observé en temps réel à partir de l'instanciation du modèle que l'on veut représenter. Une fois terminé, l'atelier de modélisation peut être déployé comme un plugin Eclipse standard.

4.3 3 Atelier graphique pour la représentation des PIM

La syntaxe concrète graphique fournit un moyen de visualiser et/ou éditer plus agréablement et efficacement un métamodèle. À cette étape, l'atelier doit permettre de spécifier toutes les notations graphiques qui seront utilisées pour créer des modèles du DSML. À chaque concept de la syntaxe abstraite sera associé un symbole graphique qui permettra de représenter le concept associé au niveau de l'éditeur de modèles. Le défi ici c'est que les symboles graphiques doivent évoquer les concepts du domaine qu'ils représentent. Cela nécessite de pouvoir combiner librement tous les éléments graphiques de bases à savoir lignes, courbes, texte . . . pour faire des symboles qui soient spécifiques au domaine.

Dans notre travail nous donnons la main auxutilisateurs de créer leurs propres représentations de leurs futures applications ubiquitaires. Le projet Eclipse Sirius facilite cette tâche en se basant sur la représentation du DSML au sein d'un modèle EMF, nous allons détailler les différentes étapes de spécification de l'éditeur avec Sirius, en expliquant les principaux concepts associés.

Après la conception de notre méta-modèle, Nous allons maintenant définir un éditeur graphique qui permet d'instancier notre méta-modèle d'une façon graphique afin d'obtenir des modèles PIM, qui seront plutôt transformer en PSM. Nous ne traiterons pas de cet aspect dans ce papier pour nous concentrer sur la définition du méta-modèle, de sa syntaxe concrète.

4.4. SEMANTIQUE

Généralement, la syntaxe abstraite ne comporte pas assez d'information pour définir le sens des constructions du DSML. Des informations supplémentaires sont nécessaires afin de pouvoir déterminer ce sens. Le but de la sémantique est donc de décrire le sens des constructions du langage. On distingue deux types de sémantique :

- **La sémantique statique** : exprimée en général sur la syntaxe abstraite à l'aide d'un langage de contrainte comme OCL, afin de définir les règles et les contraintes que doivent respecter les concepts et leurs relations lors d'une instanciation, par exemple, la multiplicité, les invariants, les pré- ou post-conditions.

- **La sémantique dynamique** : exprimée sous forme de modèle, elle permet de décrire le comportement du programme.

Pour la sémantique comportementale de nos DSL, nous avons utilisé Acceleo pour spécifier les sémantiques de nos DSML afin de générer du code ou des modèles que nous souhaitons.

4.5) Génération du code source

Généralement l'étape qui suit la manipulation du modèle est la génération du code. Cette génération renferme des templates permettant de générer du texte à partir d'un modèle EMF. Pour ce faire, nous nous appuyons sur Acceleo afin de réaliser une génération automatique à partir de modélisation haut-niveau. Fondé sur la démarche MDA, Acceleo offre un générateur de code compatible avec les standards UML, EMF, XMI, MOF permet de

transformer des modèles vers du code. Le principe général de fonctionnement d'Acceleo consiste à adopter un template de génération à un modèle de départ pour obtenir en sortie un code.

Comme tous les outils basés sur la plateforme Eclipse, Sirius bénéficie de toutes les possibilités d'extension et d'intégration offerte par Eclipse. Cependant, il y'a des améliorations à apporter au niveau du concepteur de l'éditeur. Nous souhaitons intégrer des algorithmes de ML dans la syntaxe concrète graphique. La spécification de la syntaxe concrète est liée étroitement à la syntaxe abstraite, ce qui affaiblit les possibilités de réutilisation des langages visuels.

5. APPLICATION (CASE STUDY : COVID19)

5.1 Présentation de l'étude de cas :

Notre travail consiste à générer le modèle simulable en se basant sur les concepts de l'ingénierie dirigée par les modèles, Afin d'illustrer notre approche, nous allons prendre un exemple d'application d'actualité qui pourrait limiter la diffusion du virus en identifiant des chaînes de transmission. L'idée serait de prévenir les personnes qui ont été en contact avec un malade testé positif, afin de pouvoir se faire tester soi-même et si besoin, d'être pris en charge très tôt ou bien de se confiner"..

L'usage des données et de l'IoT pour surveiller la population n'est pas une nouveauté, mais le phénomène s'est considérablement accéléré avec la crise sanitaire de ce début d'année. En effet, afin d'endiguer l'épidémie du nouveau coronavirus Covid-19.

5.2 Identification des besoins :

Dans cette étape on identifie les besoins fonctionnels, techniques et ubiquitaires, on procède à une identification globale des différents cas d'utilisations décrits dans les besoins du système :
• **User** : est le consommateur de l'application téléchargée. Elle doit prendre en compte les attentes des utilisateurs en particulier des fonctionnalités ubiquitaires à réaliser et des technologies ubiquitaires que les clients souhaitent utiliser. Aussi, ajouter des besoins non-fonctionnels comme la sécurité.

• **L'application TraceCovid-19** : est l'application proposée au consommateur. Elle doit fournir des besoins fonctionnels et définir certains besoins ubiquitaires s tels que Personnaliser les informations fournies ou Adapter l'affichage.

Le résultat de cette étape est un diagramme de cas d'utilisation, les liens entre eux et avec les acteurs et les besoins ubiquitaires qui leurs sont associés.

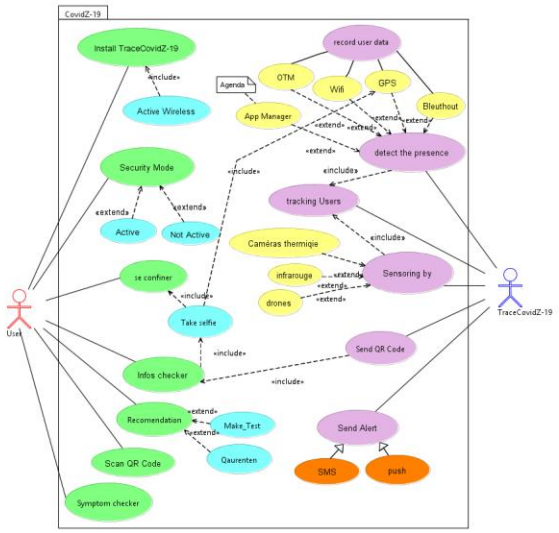

fig 6 : Diagramme de cas d'utilisation correspondant aux besoins ubiquitaires.

Nous considérer le diagramme des cas d'utilisation comme notre modèle CIM de départ à partir duquel un concepteur spécifie un modèle PIM en utilisant le langage UML. La Figure représente le PIM de notre exemple d'application COVID19, y compris certains services avec leurs opérations.

La figure 7 illustre un diagramme de sequence pour TraceCovid-19, l'activité principale qui est « Tracking User » implique l'invocation de deux services :

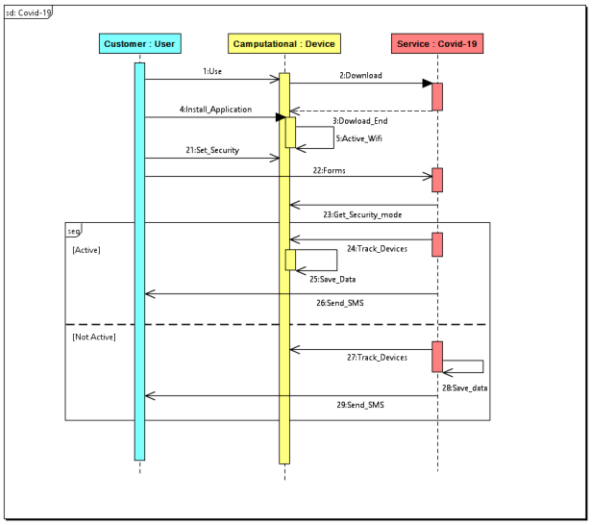

fig 7 : Le PIM de notre exemple d'application.

5.3 Modélisation de notre cas d'étude COVID-19 avec l'éditeur graphique

La figure 4.4 montre l'environnement de modélisation dans lequel nous avons créé au moyen de Sirius. A gauche, nous visualisons un extrait de modèle conçu et édité avec Sirius. Nous visualisons également à droite

un extrait de notre palette d'outils d'éléments graphiques facilitant la création des éléments de notre diagramme. Dans ce qui suit, nous détaillons les éléments du modèle affiché dans cette figure.

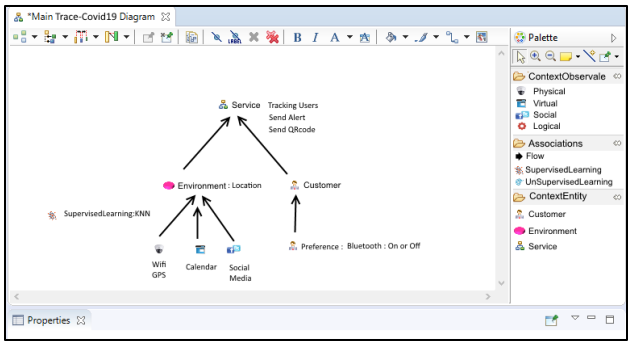

fig 8 : Extrait de l'éditeur réalisé avec Sirius.

5 CONCLUSION

La complexité de conception des systèmes ubiquitaires augmente parallèlement, avec les besoins des utilisateurs qui ne cessent de s'accroître. Dans ce contexte, l'Ingénierie Dirigée par les Modèles, se présente comme une réponse pour relever ce défi. Cet article propose l'utilisation d'une approche dirigée par les modèles pour concevoir des applications ubiquitaires.

Pour cela, nous nous sommes situés dans le contexte technique standard du domaine MDA et nous avons employés ainsi une chaîne d'outils de ce domaine : EMF pour la définition du métamodèle, Sirius pour la création de modèles d'une manière graphique concrète et Acceleo pour la génération du code.

Pour atteindre cet objectif, on a commencé par donner un état de l'art qui comporte un aperçu de l'Ingénierie Dirigée par les Modèles, de la notion de système ubiquitaire et les outils de de modélisation, méta-modélisation, puis l'accent a été mis sur un modèle contextuel conforme au méta-modèle proposé et en utilisant le langage Ecore standard intégrant les principales entités utilisées dans l'état de l'art des applications ubiquitaires.

Notre approche est implémentée dans un framework (atelier graphique). Cet éditeur a été conçu pour les modèles PIM à l'aide de l'outil Eclipse Sirius, cet éditeur est basé sur notre méta-modèle et facilite la spécification des modèles PIM d'une façon totalement graphique, enfin Nous avons démontré nos contributions sur l'exemple d'une application d'actualité Covid-19. Ceci permet de mettre en place des scenarios applicatifs très riches impliquant un grand nombre d'objet contextuel.

Actuellement nous travaux utilisons l'environnement de développement Xtext (Xtext) pour créer un éditeur textuel, l'objectif est de fournir une syntaxe concrète hybride : une partie graphique et une partie textuelle.

Bibliographie

[1] Mark Weiser, The computer for the 21st century, Sci-entic American, tome 265, n° 3, pages 66-75, septem-bre 1991.

[2] Mohammed Fethi Khalfi and Sidi Mohamed Benslimane, Toward A Generic Infrastructure for Ubiquitous Computing, Journal: International Journal of Advanced Pervasive and Ubiquitous Computing, 2013, Volume 5, Number 1, Page 66.

[3] M. Khalfi and S. M. Benslimane, Systèmes D'information Pervasifs : Architecture et Challenges. Presented at the UbiMob 14, 2014.

[4] Li, X., Tao, X., Song, W. et al. AocML: A Domain-Specific Language for Model-Driven Development of Activity-Oriented Context-Aware Applications. J. Comput. Sci. Technol. 33, 900–917 (2018).

[5] Aicha Azoui and Djilali Idoughi, Towards Ubiquitous Services Design and Development Approach. HCI (19) 2016: 3-14.

[6] A. M. Baddour et al., "CIM-CSS: A Formal Modeling Approach to Context Identification and Management for Intelligent Context-Sensitive Systems," in IEEE Access, vol. 7, pp. 116056-116077, 2019.

[7] Dörndorfer J., Hopfensperger F., Seel C. (2019). The SenSoMod-Modeler – A Model-Driven Architecture Approach for Mobile Context-Aware Business Appli-cations.. In: Cappiello C., Ruiz M. (eds) Information Systems Engineering in Responsible Information Sys-tems. vol350. Springer, Cham.

[8] Christos Mettouris1(&), Achilleas Achilleos2, Georgia Kapitsaki1, The UbiCARS Model-Driven Framework: Automating Development of Recommender Systems for Commerce.

[9] hristos Mettouris and George A. Papadopoulos, 'UbiRS4Tourism: Design and Development of Point-of-Interest Recommender Systems Made Easy', 26th Annual eTourism Conference (ENTER 2019), Nicosia, Cyprus, 29 January-01 February 2019, e-Review of Tourism Research (eRTR 2019), Volume 16 Issue 2-3 2019, Applied Research Notes 11.

[10] Jaouadi, I., Ben Djemaa, R. & Ben-Abdallah, H. A model-driven development approach for context-aware systems. Softw Syst Model 17, 1169–1195 (2018).

[11] Boudjemaa Boudaa, Slimane Hammoudi, Leila A. Mebarki, Abdelkader Bouguessa, Mohammed Amine Chikh: An aspect-oriented model-driven approach for building adaptable context-aware service-based applications. Sci. Comput. Program. 136: 17-42 (2017).

[12] A Model-Driven Approach to Generate Context-Aware Applications.. the 20th Brazilian Symposium. 14, p. 99.

[13] Boudjemaa BOUDAA, Olivier CAMP, Slimane HAMMOUDI, Mohammed Amine CHIKH Model-Driven Development of Context-Aware Services: Issues, Techniques and Review.