# Event detection in connected environments

Elio Mansour



UNIV PAU & PAYS ADOUR

DOCTORAL THESIS

# Event Detection in Connected Environments

**Elio MANSOUR**

| | | |
|---|---|---|
| Advisors: | Pr. Richard CHBEIR | Univ Pau & Pays Adour, France |
| | Dr. Philippe ARNOULD | Univ Pau & Pays Adour, France |
| Reviewers: | Pr. Myriam LAMOLLE | Paris 8 University, France |
| | Dr. Khalil DRIRA | LAAS-CNRS, France |
| Examiners: | Pr. Yannis MANOLOPOULOS | Open University of Cyprus, Cyprus |
| | Pr. Kokou YETONGNON | University of Bourgogne, France |
| | Pr. Philippe ANIORTÉ | Univ Pau & Pays Adour, France |

*A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science*

November 18, 2019

*"Live as if you were to die tomorrow. Learn as if you were to live forever."*

Mahatma Gandhi

*To my loved ones...*
*I dedicate this work to my parents Sylvana & Elias*
*and to my lover and best friend Elsa*

# *Acknowledgements*

Before going through the scientific details of this research, I would like to thank the people and institutions that were pivotal to this work...

I would like to start by thanking Pr. Myriam LAMOLLE and Dr. Khalil DRIRA for the time and effort they devoted to the examination of my dissertation. I also thank them for their comments, suggestions, and feedback. Moreover, I am grateful for the presence of Pr. Yannis MANOLOPOULOS, Pr. Kokou YETONGNON, and Pr. Philippe ANIORTÉ in my thesis evaluation committee. Your comments and input are highly appreciated.

My deepest gratitude goes to my supervisors Pr. Richard CHBEIR and Dr. Philippe ARNOULD. I want to thank Richard for believing in me, helping me make the most of my abilities, and insisting on elevating my skills and overall approach towards research. His guidance, insightful comments, bright ideas, and immense knowledge made this work possible. Aside the scientific aspects, I also want to thank him for understanding my personality, always encouraging me, and all the coffee breaks where we did not talk about work because he knew I was exhausted and needed a break. My sincere heartfelt appreciation also goes to Philippe. He made things simple, easy, and comforting. This journey would not have been that enjoyable without him. He always motivated me, gave me great advice, and on a social level helped me understand and integrate the French culture. It was a pleasure to have known both of them, and I look forward to possibly working with them again in the future.

I am thankful for being part of the LIUPPA laboratory. I specifically appreciate the seminars and scientific gatherings organized by its administration.

I owe my gratitude to the "Conseil Départemantal des Landes". Nothing would have been possible without its financial support and interest in promoting science.

I deeply appreciate the help of Dr. Gilbert Tekli during my first year and even before the start of my Ph.D., when he continuously encouraged me to pursue a career in research. I also benefited a lot from the discussions I had with Dr. Joe Tekli. His ideas and opinions helped me during my final year.

I had the pleasure of working in two campuses during this journey. I thank the "IUT des Pays de l'Adour" in Mont-de-Marsan for their unconditional and massive support throughout the previous years. I cannot name everyone, but I would be remiss not to mention Alain Lespine, Laurent Gallon, Karine Brugirard Ricaud, Jamal El Hachem, Timothée Duron, Thomas Cabaret, Annabelle Tavernier, Juliette Merle, and all my fellow PhD candidates.

I also want to thank the "IUT de Bayonne et du Pays Basque" in Anglet for allowing me to use the materials and facilities. I would like to name Philippe Aniorté, Christophe Marquesuzaa, Patrick Etcheverry, Philippe Lopistéguy, Pantxika Dagorret, and my colleagues in the research open space. I acknowledge the moral and emotional support provided by Lara Kallab, Nathalie Charbel, Khouloud Salameh,

Karam Bou Chaaya, Fawzi Khattar, Rita Zougheib, Jocelyn Habib, and Sabri Allani. These people were at my side every time I needed them.

I shared the longest part of my journey with Nathalie and Lara who became the sisters I never had. I have the outmost love and respect for them.

I do not want to forget my friends in Lebanon. The happiness, support, and laughter brought by Elie Saad, Jad Bou Khalil, and Elie Khoury were of massive importance during these last years.

I sincerely appreciate my volleyball teammates for helping me keep a good level of mental and physical health.

Finally, I have to acknowledge the huge support provided by my family. I felt the presence of my cousins from miles and miles away. Many thanks go to Jules, Léa, Lama, Noelle, Hind, Marly, Roudy, Maribelle, Lello, Elissa, Pierre, Perla, Carole, and Charbel.

I am in huge debt to my parents in every way possible. They loved me, encouraged me, worked hard to give me the opportunity to study abroad, and backed every idea or project I ever had. Words will never tell the entire story about how much they have done, endured, and given up for me...

And last but definitely not least, Elsa. The woman I love and without whom I would have never been able to keep up. She picked me up when I struggled, tolerated me during my bad days, helped me navigate through adversity, and shared my happiest moments. From the bottom of my heart, thank you...

# *Abstract*

The rising interest in smart connected environments (e.g., smart buildings, cities, factories) and the evolution of sensors, data management/communication technologies have paved the way for interesting and useful applications that help users in their every day tasks (e.g. increasing comfort, reducing energy consumption). However, various improvements are still required. For instance, how to enhance the representation of such complex, dynamic, and heterogeneous environments. Moreover, how to facilitate the interaction between users and their connected environments, and how to provide tools for environment monitoring and management.

In this thesis, we focus on four main challenges: (i) representing a diverse set of components and elements related to the environment and its sensor network; (ii) providing a query language that handles user/connected environment interactions (e.g., environment definition, data management, event definition); (iii) coping with the dynamicity of the environment and its evolution over time; and (iv) proposing a generic event detection mechanism for improved environment monitoring.

To do so, we first present an ontology-based data model that represents hybrid environments/sensor networks. Thus covering diverse sensors (e.g., static, mobile), environments (e.g., infrastructures, devices), and data (e.g., scalar, multimedia). Then, we introduce a query language that one might use for various tasks (e.g., defining the connected environment, information retrieval, event definition, data management). Furthermore, to keep up with the environment changes we provide a query optimizer that allows the submitted queries to cope with the dynamicity of the environment prior to their execution. Finally, we propose an event detection core that takes event definitions as input and detects the targeted events.

We group the aforementioned modules in one global framework for event detection in connected environments. Our proposal is generic, extensible, and could be used with different connected environments such as buildings, cities...

# *Résumé*

L'intérêt croissant pour les environnements connectés (bâtiments, villes, usines intelligents) et l'évolution des réseaux de capteurs, technologies de gestion/communication de données ont ouvert la voie à des applications intéressantes et utiles qui aident les utilisateurs dans leurs tâches quotidiennes (augmenter le productivité dans une usine, réduire la consommation d'énergie). Cependant, diverses améliorations sont encore nécessaires. Par exemple, comment améliorer la représentation de ces environnements complexes, dynamiques et hétérogènes. En outre, comment faciliter l'interaction entre les utilisateurs et leurs environnements connectés et comment fournir des outils de surveillance et de gestion de tels environnements.

Dans cette thèse, nous nous concentrons sur quatre défis principaux: (i) représenter un ensemble diversifié de composants et d'éléments liés à l'environnement et à son réseau de capteurs; (ii) fournir un langage de requête qui gère les interactions utilisateur/environnement connecté (pour la définition de l'environnement, la gestion de données, la définition d'événements); (iii) faire face à la dynamique de l'environnement et à son évolution dans le temps; et (iv) proposer un mécanisme générique de détection d'événements pour mieux surveiller l'environnement.

Pour ce faire, nous présentons d'abord un modèle de données basé sur une ontologie qui représente des environnements et réseaux de capteurs hybrides. Couvrant ainsi divers capteurs (statique, mobile), environnements (infrastructures, équippements) et données (scalaires, multimédia). Ensuite, nous introduisons un langage de requête que l'on pourrait utiliser pour diverses tâches (définir l'environnement connecté, la recherche d'informations, la définition d'événements, la gestion de données). De plus, afin de suivre les changements d'environnement, nous fournissons un optimiseur de requêtes qui permet aux requêtes soumises de gérer la dynamique de l'environnement avant leur exécution. Enfin, nous proposons un noyau de détection d'événement qui prend en entrée les définitions d'événement et détecte les événements ciblés.

Nous regroupons les modules susmentionnés dans un framework global pour la détection d'événements dans des environnements connectés. Notre proposition est générique, extensible, et pourrait être utilisée avec différents environnements connectés tels que des bâtiments, des villes...

Le manuscrit est organisé comme suit:

## Chapitre 1

### Introduction

Dans ce chapitre, nous introduisons les facteurs qui ont contribué à la prolifération des environnements connectés pendant nos jours. Nous évoquons des facteurs technologiques (par exemple, les avancées dans le domaine du traiement avancé des données, la modélisation des données, la miniaturization des capteurs), ainsi que

des facteurs économiques et environnementaux. Ensuite, nous nous concentrons sur les objectifs de cette thèse: modéliser et interroger de tels environnements, gérer leur dynamicité et détecter des événements spécifiques se produisant dans leurs locaux. Nous présentons un scénario d'environnement connecté dans un centre commercial intelligent qui illustre la motivation de ce travail et les défis à venir. Ensuite, nous présentons brièvement notre framework proposé pour la détection d'événements dans des environnements connectés (EDCE), dans lequel chaque module répond à un objectif spécifique et répond à un ensemble de besoins et de défis:

- Premier Module: Un modèle ontologique pour enrichir la représentation d'un environnement connecté et son réseau de capteurs. L'ontologie proposée, notée HSSN, propose une description de divers types de capteurs, environnements/-platformes de déploiement, et de données.
- Deuxième Module: Un language de requêtes dédié aux environnements connectés, noté EQL-CE. Ce language permet de considérer tous les composants d'un environnement connecté (c'est à dire, l'environnement, le réseau de capteurs, les événements ciblés, et le domaine d'application). De plus ce language fournit des requêtes pour définir les composant susmentionnés, rechercher des informations, gérer les données, et définir les événements à détecter.
- Troisième Module: Un optimiseur de requêtes qui permet de détecter et réécrire les requêtes qui ne pourront plus fournir des résultats, à cause des changements et de la dynamique de l'environnement.
- Quatrième Module: Un détecteur d'événements, noté eVM, qui prend en entrée les définitions d'événements ciblés et les données fournies par le réseau de capteurs pour détecter les événements.

Finalement, nous répertorions les publications liées à ce rapport avant d'introduire les chapitres suivants.

## Chapitre 2

### Un Modèle de Données pour les Environnements Connectés

Dans ce chapitre, nous décrivons un modèle d'information basé sur une ontologie. Nous présentons une étude comparative des travaux existants sur la modélisation des réseaux/données de capteurs. Ensuite, nous introduisons notre prosition (HSSN [68]) dans laquelle nous enrichissons la représentation de l'environnement et la modélisation du réseau de capteurs avec divers types de capteurs (des nœuds simples, mobiles, statiques, équippements multi-capteurs, capteurs scalaires, multimédia); (ii) plates-formes de déploiement (infrastructures physiques, plates-formes et dispositifs électroniques); et (iii) des données détectées (scalaires, multimédia). Pour se faire, nous étendons l'ontologie SSN [44], qui est largement utilisée dans l'état de l'art, sans compromettre la possibilité de réutilisation du modèle de données dans différents contextes. Enfin, nous évaluons la performance, la clarté, la cohérence et la précision de nos ajouts.

## Chapitre 3

### Un Language de Requêtage pour Environnements Connectés

Dans ce chapitre, nous décrivons le langage de requêtes que les utilisateurs peuvent utiliser comme moyen d'interaction avec leurs environnements connectés. Nous passons en revue les travaux existants sur les langages de requêtes et proposons EQL-CE (un langage de requête d'événement adapté aux environnements connectés [67]). Nous ne traitons pas tous les défis liés à la proposition d'un langage de requêtes pour les environnements connectés, mais nous nous concentrons principalement sur les éléments suivants:

- Couvrir tous les composants de l'environnement connecté et pas seulement les événements.
- Couvrir divers types de requête courantes (par exemple, pour la définition de l'environnement, la recherche d'information, insertion de données, la définition d'événements).
- Couvrir divers types de données (par exemple, des données scalaires, multimédia).
- Consdérer la distribution spatiale des capteurs dans l'espace et les distributions temporelles des observations des capteurs dans le temps.
- Gérer la dynamique de l'environnement connecté, et s'adapter aux changements et évolution de l'environnement dans le temps.
- Fournir une syntaxe réutilisable, qui ne dépend pas des facteurs techniques (par exemple, l'infrastructure de stockage de données).

Nous proposons trois couches pour le langage (c'est-à-dire une couche conceptuelle, logique et physique). Nous détaillons la syntaxe de chaque composant de l'environnement connecté et des différents types de requêtes. Enfin, nous présentons un exemple d'illustration et un protocole expérimental pour évaluer le langage.

## Chapitre 4

### Gérer la Dynamique de l'Environnement

Dans ce chapitre, nous nous concentrons sur un besoin spécifique concernant le language de requêtes: gérer la dynamique de l'environnement. Nous soulignons divers facteurs qui contribuent à la dynamique de l'environnement (la mobilité/fiabilité des capteurs, la compatibilité entre les différentes plate-formes, la volatilité des données, les données/attributs manquants). Ensuite, nous traitons deux défis principaux: (i) la mobilité/la fiabilité des capteurs; et (ii) les données/attributs manquants. Nous proposons un module d'optimisation de requêtes qui complète le langage de requêtes d'événement EQL-CE qui a été détaillé dans le chapitre précédent. Pour se faire, l'optimiseur analyse les requêtes d'événements, avant de les envoyer au détecteur, pour voir si une requête contient des éléments "obsolètes", c'est à dire des éléments qui ne peuvent plus servir leurs fonctions dans la requête (par exemple, un capteur qui n'éxiste plus, qui à bougé, qui est tombé en panne). Par la suite, l'optimiseur réécrit les requêtes qui sont devenues "obsolètes" en raison de la dynamicité de l'environnement (c'est à dire, des requêtes qui ne sont plus

capables de retourner des résultats à cause des changements qui ont eu lieu dans l'environnement). Nous définissons formellement l'envirnnement connecté, les requêtes d'événements, les requêtes obsolètes, et les capteurs. Puis, nous détaillons deux algorithmes pour la détection et la réécriture des requêtes obsolètes. Enfin, nous évaluons les complexités des algorithmes et proposons un protocole expérimental pour mesurer la précision et la performances des deux processus (la détection et la réécriture des requêtes "obsolètes").

## Chapitre 5

### Un Framework Générique pour la Détection des Événements

Dans ce chapitre, nous décrivons le module de détection d'événements, noté eVM, qui prend en entrée des définitions d'événements (requêtes d'événements EQL-CE) et des objets de données (par exemple, des données détectées par les capteurs dans un environnement connecté) afin de détecter les événements ciblés. Tout d'abord, nous passons en revue divers travaux de détection d'événements. Nous présentons aussi une étude comparative de différents techniques de clustering pour pouvoir choisir une technique à utiliser dans le processus de détection d'événements. Suite à l'étude de l'éxistant, nous choisissons de baser le détecteur sur une technique conceptuelle de clustering: "Formal Concept Analysis" (FCA), et détaillons les principes et le fonctionnement global de cette technique. Ensuite, nous présentons un framework de détection d'événements générique reposant sur FCA. Nous détaillons le processus de détection d'événements de l'entrée (requête d'événement EQL-CE) à la sortie (les événements détectés). Finalement, nous présentons l'expérimentation et les résultats dans différents domaines d'application.

## Chapitre 6

### Conclusion et Travaux Futurs

Ce chapitre conclut le rapport en récapitulant tous les chapitres susmentionnés et en détaillant les prochaines étapes, extensions futures, et de nouvelles orientations possibles pour la suite de ce travail de recherche.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **MEMS** | **M**icro **E**lectro **M**echanical **S**ystems |
| **CRE** | **C**ommercial **R**eal **E**state |
| **CAGR** | **C**ompound **A**nnual **G**rowth **R**ate |
| **GHG** | **G**reen **H**ouse **G**ases |
| **GPS** | **G**lobal **P**ositioning **S**ystem |
| **SN** | **S**ensor **N**etwork |
| **WSN** | **W**ireless **S**ensor **N**etwork |
| **SSN** | **S**emantic **S**ensor **N**etwork |
| **SOSA** | **S**ensors **O**bservations **S**amples **A**ctuators |
| **HSSN** | **H**ybrid **S**emantic **S**ensor **N**etwork |
| **HVAC** | **H**eating **V**entilation **A**ir **C**onditioning |
| **CPU** | **C**entral **P**rocessing **U**nit |
| **IEEE** | **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers |
| **SUMO** | **S**uggested **U**pper **M**erged **O**ntology |
| **SDO** | **S**ensor **D**ata **O**ntology |
| **SHO** | **S**ensor **H**ierarchy **O**ntology |
| **EPO** | **E**xtension **P**lug-ins **O**ntologies |
| **W3C** | **W**orld **W**ide **w**eb **C**onsortium |
| **OGC** | **O**pen **G**eospatial **C**onsortium |
| **MPEG-7** | **M**oving **P**icture **E**xperts **G**roup |
| **EXIF** | **E**xchangeable **I**mage **F**ile |
| **MSB** | **M**obile **S**ensing **B**ox |
| **PSD** | **P**ersonal **S**ensing **D**evice |
| **WSMO** | **W**eb **S**ervice **M**odeling **O**ntology |
| **SPARQL** | **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage |
| **ICT** | **I**nformation and **C**ommunication **T**echnologies |
| **IoT** | **I**nternet **o**f **T**hings |
| **EQL** | **E**vent **Q**uery **L**anguage |
| **CeDR** | **C**omplex **e**vent **D**etection and **R**esponse |
| **CQL** | **C**ontinuous **Q**uery **L**anguage |
| **C-SPARQL** | **C**ontinuous **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage |
| **SPARQL**$_{Stream}$ | **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage **S**tream |
| **T-SPARQL** | **T**emporal **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage |
| **SPARQL-ST** | **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage - **Spatio T**emporal |
| **EP-SPARQL** | **E**vent **P**rocessing **S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage |
| **EQL-CE** | **E**vent **Q**uery **L**anguage for **C**onnected **E**nvironments |
| **SQL** | **S**tructured **Q**uery **L**anguage |
| **EBNF** | **E**xtended **B**ackus-**N**aur **F**orm |
| **ECA** | **E**vent **C**ondition **A**ction |
| **RDF** | **R**esource **D**escription **F**ramework |
| **ISO** | **I**nternational **O**rganization for **S**tandardization |
| **BNF** | **B**ackus-**N**aur **F**orm |
| **TF-IDF** | **T**erm **F**requency **I**nverse **D**ocument **F**requency |
| **RAM** | **R**andom **A**ccess **M**emory |

| | |
|---|---|
| **NMI** | **N**ormalized **M**utual **I**nformation |
| **eVM** | **e**vent **V**irtual **M**achine |
| **API** | **A**pplication **P**rogram **I**nterface |
| **FCA** | **F**ormal **C**oncept **A**nalysis |
| **ECG** | **E**lectro**C**ardio**G**ram |
| **ACLED** | **A**rmed **C**onflict **L**ocation & **E**vent **D**ata |
| **RCA** | **R**elational **C**oncept **A**nalysis |

# Chapter 1

# Introduction

*"Faith is taking the first step even when you don't see the whole staircase."*

*— Martin Luther King, Jr.*

## 1.1 Connected Environments

Recent years have witnessed a widespread interest in smart connected environments. Typically defined as infrastructures that host sensor networks capable of providing valuable data for various applications (e.g., home automation, energy management), the connected objects and environments are impacting numerous application domains. From smart homes and buildings to cities, vehicle networks, and electrical grids, they have become a novel trend that is revolutionizing how people interact with their personal surroundings, how they accomplish their daily tasks in the workplace, and how they handle their health, security, and safety. Connected environment markets are currently booming and projected to continue their growth for the years to come. Figure 1.1 shows the distribution of investments in smart buildings by region. The investment growth in such connected environments is significant (from 7.42 billion dollars in 2017 to a projected 31.74 billion dollars in 2022).[1]

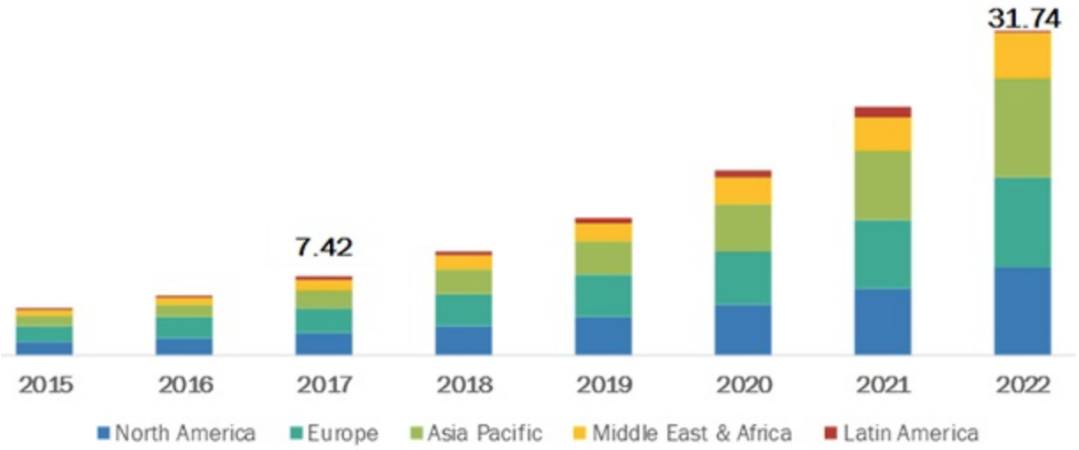


FIGURE 1.1: Smart Buildings Market, By Region (USD Billion)

[1]Source: MarketsandMarkets Analysis - Link: https://www.marketsandmarkets.com/Market-Reports/smart-building-market-1169.html

Moreover, Figure 1.2 highlights the increasing investments in smart cities from 2016 to 2023. An estimated total of 79.5 billion dollars were invested in smart cities in 2018. The chart[2] shows that the continuous growth in smart city investments is reaching a projected total of 219.6 billion dollars in 2023. Finally, both figures show that the increasing interest in the aforementioned connected environments is global since North America, Europe, Asian Pacific (APAC), the Middle East (MEA), Africa, and Latin America are all contributors to the investments.

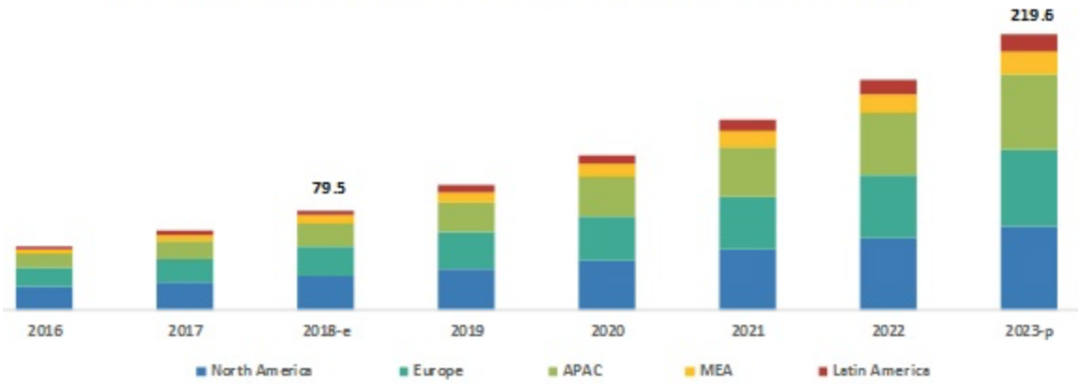


FIGURE 1.2: Smart Cities Market, By Region (USD Billion)

### 1.1.1 Driving Factors

The wide-spreading investments in connected environments and their integration in different fields have been driven and motivated by various factors. We present next the different categories of driving factors.

#### 1.1.1.1 Technological Factors

**Data Processing & Modeling Techniques.** Recent advances in Information and Communication Technologies (ICT), Big Data, and Data Mining techniques have made it easier to tackle challenges related to managing (i) big data volumes; (ii) heterogeneous data (e.g., such data is collected/sensed in connected environments); (iii) continuous data streams; (iv) data pre-processing (e.g., cleaning, normalization); and (v) knowledge extraction from raw data (e.g., sensor data).
Moreover, advances in data modeling have enriched the descriptions of connected environments, sensor networks, different types of sensors, sensed data/observations, communication protocols, and platforms for sensor deployment. This provided an expressive, semantic, and rich representation of connected environments that could benefit high level applications.
All of the above, allowed the aforementioned environments to impact different application domains (e.g., energy management, home automation) and provide various applications for users (e.g., energy consumption predictions in smart buildings, increasing production efficiency in factories). These applications require advanced data processing and modeling techniques (e.g., clustering, classification, anomaly detection, semantic models, ontologies) to provide their intended services for every day users.

[2]Source: MarketsandMarkets Analysis - Link: https://www.marketsandmarkets.com/Market-Reports/iot-smart-cities-market-215714954.html

**Sensing & Miniaturization.** Recent advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multi-functional sensor nodes that are small in size [3]. One could deploy such sensors in different environments (e.g., buildings, cities) and/or embed them on various machines, devices, and electronic platforms (e.g., mobile phones). The advanced capabilities of new sensors (e.g., sensing various properties, transmitting data, storing observations), their increased autonomy (e.g., longer life cycles, more battery power, more fault/breakdown resistance), and their miniaturization have allowed sensor networks to be widely adopted for environment monitoring. This has greatly benefited the proliferation of connected environments. Figure 1.3 shows the increasing number of sensors deployed in Commercial Real Estate (CRE) over the past years. The study projects a Compound Annual Growth Rate (CAGR) of 78.8% from 2015 to 2020.

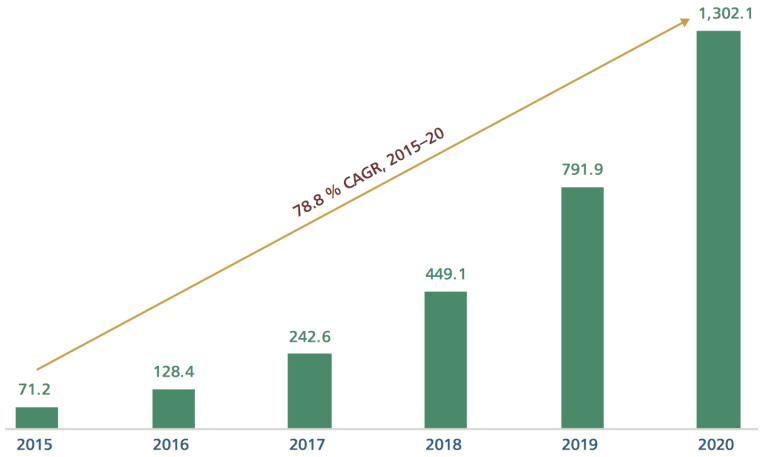


FIGURE 1.3: Sensor Usage in Commercial Real Estate (CRE), millions

#### 1.1.1.2 Other Factors

Besides the technological advances, other factors have also helped the proliferation of connected environments (e.g., environmental, economic, political). For instance, due to the high impact that energy consumption by buildings has at the global scale, energy-efficient buildings (i.e., that reduce CO2 emissions and energy consumption) are now needed more than ever. By 2020, there will be 7.5 billion people in the world and consumption will increase by 75% compared to the year 2000, equally split between developing and developed countries. This means an increase of 37.5% in energy consumption every 10 years. These factors have driven research on sustainability in energy production, distribution, storage, and consumption [72, 90]. From a political standpoint, Europe has set the European 20-20-20 objectives. This entails (i) decreasing gas emissions with greenhouse effect (GHG) by 20%; (ii) decreasing the energy consumption by 20%; and (iii) increasing the production of renewable energy by 20% . These objectives have been set since studies [76] show that buildings are responsible for 40% of total European Union energy consumption and generate 36% of GHG. This highlights the need to achieve energy-efficient buildings (and thereafter cities) to reduce their CO2 emissions and their energy consumption. Moreover, this affects the quality of life and work of all citizens/building occupants. Thus, there is

a need to address and balance environmental (i.e., eco-friendly and green environments), economic (i.e., lowering energy costs), and occupant comfort requirements (i.e., healthy, comfortable, and safe working/living environments).

## 1.2 Thesis Context

This thesis considers connected environments from a modeling and management perspective. We are interested in aiding connected environment users (e.g., smart home owner, smart building manager, smart city occupants) in setting up and managing their own environments. The general, or global, goal is to provide a framework that allows the user the definition and management of a connected environment without having to deal with the low-level technical aspects (e.g., data storage, infrastructure evolution, changing technical constraints/needs). Thus, allowing users to focus on high level applications related to defining his/her needs and requirements, interacting with the system, defining and managing data, and event detection. Moreover, since the environment is dynamic and could change over time, we need to ensure that the provided tools evolve and keep up with the changes. Finally, we aim to provide the user with means that allow him/her the definition of specific happenings, events, or patterns that he/she would like to track, detect, and find within the premises of his environment.

### 1.2.1 Thesis Objectives

Specifically, the objectives of this thesis can be summarized as follows:

- Designing a generic data model that could be reused to describe various connected environments (e.g., buildings, homes, cities). This entails covering the environment and its sensor network. In addition, the data model should consider event modeling in various application domains.
- Providing users with a means for interaction with their connected environments. This entails having one interface from which the user can formulate his/her demands and requirements (e.g., defining components, managing data, detecting events).
- Ensuring that the interaction between the user and environment is not static and is capable of coping with the dynamicity and evolution of the connected environment over time.
- Providing the user with a generic way of defining events based on his/her specifications, and a common mechanism for event detection that could be reused for different events and in various environments/contexts.

### 1.2.2 Motivating Scenario

To illustrate the motivations behind the objectives of this thesis, we provide here a connected environment example. Figure 1.4 shows a smart mall (shopping center) where clients spend time shopping, eating, watching movies and so on. The figure details the infrastructure of the environment, the location map, locations, and the spatial setup. From a sensor network standpoint, the figure shows various sensors (e.g., noise, temperature) deployed in the entire mall. These sensors can provide valuable data that could be exploited for high level applications such as reducing

energy consumption, increasing client comfort/safety, and improving the overall shopping experience in the mall. For the aforementioned purposes, a mall manager is interested in managing the environment and detecting specific events that occur within its premises. To achieve this, various needs should be considered:

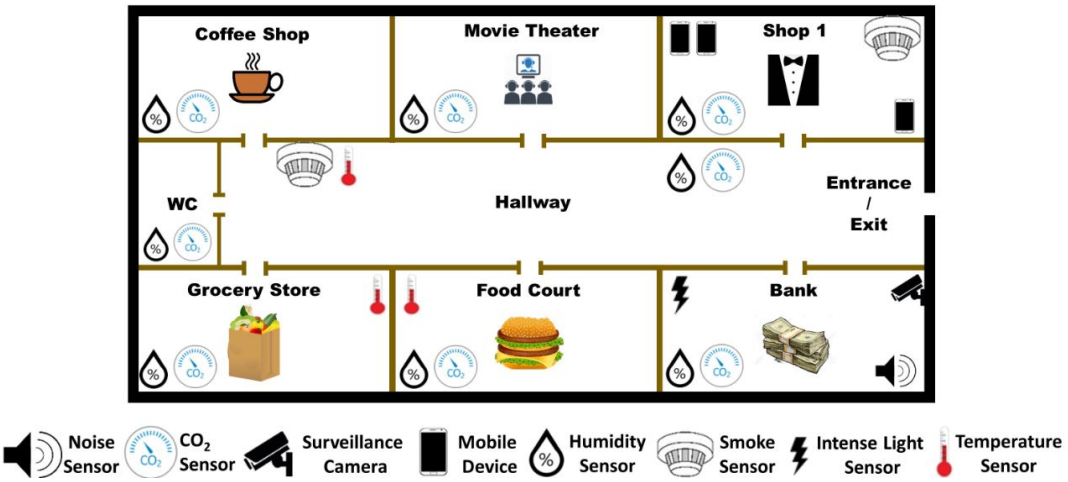


FIGURE 1.4: Motivating Scenario Overview

**Need 1.** Retrieve diverse, rich, and precise observations/events from the mall.

**Need 2.** Define and manage all components (i.e., objects or entities), data, and datatypes in the mall.

**Need 3.** Cope with the dynamicity of the mall and its evolution over time.

**Need 4.** Define and detect events of interest that happen in the mall.

To address the aforementioned needs, the mall manager must have tools that allow him/her to (i) model the entire connected environment (e.g., the mall); (ii) query the environment for definition, data retrieval, and management purposes while considering constraints related to dynamicity and evolution; and (iii) define/detect events of interest. However, when considering all of the above, several challenges emerge:

**Challenge 1.** How to represent a diverse set of elements related to the environment and sensor network (e.g., sensors, platforms, data)?

**Challenge 2.** How to provide one reusable query language that addresses various tasks (e.g., component definition, data retrieval, event definition, data management) for all connected environment components (i.e., related to the environment, sensor network, events, and application domain)?

**Challenge 3.** How to allow the query language to cope with the dynamicity of the connected environment and its evolution over time?

**Challenge 4.** How to provide a mechanism for event detection that could be common for the detection of different events?

Several other challenges exist when considering the topic of connected environments (e.g., stream data processing, data volatility, real-time event detection). However, we focus here on the aforementioned needs and challenges (which will be further detailed separately in the following chapters). We present next our proposed framework and detail how each module addresses a specific challenge.

## 1.3 Proposal

We present here a brief overview of our proposal for Event Detection in Connected Environments, denoted EDCE. The framework addresses the needs and challenges in Section 1.2. Figure 1.5 shows an overview of its main modules. Briefly, the system interrogation module represents the interface (a query language) that one could use to interact with the connected environment. The data model module describes the connected environment components and their ties/relations. The event Virtual Machine module is a pluggable event detector that takes event definitions and sensed data as input, and outputs the targeted events. Finally, the query optimizer aids the query language (system interrogation module) in coping with the dynamicity and evolution of the environment.

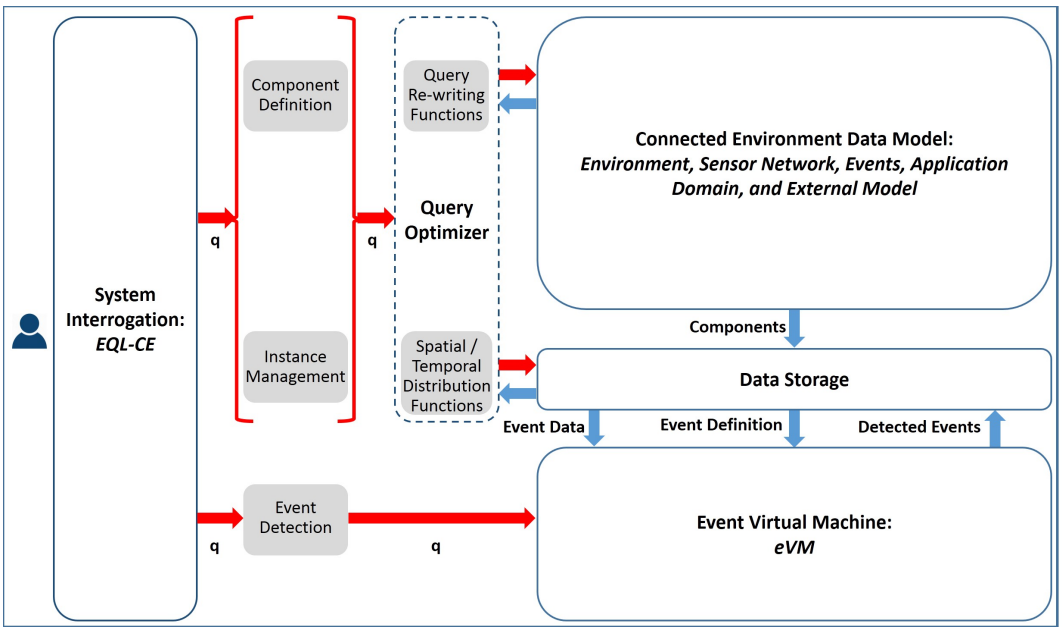


FIGURE 1.5: EDCE Global Framework

### 1.3.1 Connected Environment Data Model

This module proposes a representation of the entire connected environment by considering four main parts (cf. Figure 1.6).

- Part 1 - The environment modeling: is an essential part of the global connected environment model. It covers the representation of the physical real world infrastructure (e.g., a building, city, grid, home, factory) and all its underlying components (e.g., location map, individual locations, embedded/nested environments). Moreover, the environment could contain devices, machines, and equipment that host sensors. The description of the environment components as well as their ties and interactions are considered in this part of the connected environment data model.

- Part 2 - The sensor network modeling: is the second part of the data model. Each connected environment hosts one or more sensor networks. The purpose of sensors is the gathering of useful data by monitoring the environment

(where the sensor network is deployed) for high level applications. We represent various sensor types (e.g., nodes, devices, mobile, static) and sensed data (e.g., scalar, multimedia) in this part of the connected environment data model.

- Part 3 - The event modeling: is dedicated to the representation of event definitions. The latter are crucial for the detection of occurring events in the connected environment.

- Part 4 - The application domain modeling: is a pluggable part in the connected environment data model. We use it to enrich the description of the environment (e.g., a smart hospital differs from a smart mall in terms of its constituent components and the buildings' constraints/configurations). Similarly, the application domain also affects the events (e.g., a body temperature overheating event for a patient is defined differently than a room overheating event in a building).

For parts 1 and 2, we propose an extension of several ontologies and mainly the SSN ontology, denoted HSSN (Hybrid Semantic Sensor Network). We discuss our proposal in Chapter 2. The interconnection of the four parts and the overall data model for connected environments is presented in Chapter 3. Finally, we detail furthermore the event modeling in Chapter 5 (cf. Need 1). This study is published [68] in the proceedings of the $23^{rd}$ International Database Applications & Engineering Symposium (IDEAS 2019):

- *Elio Mansour, Richard Chbeir, Philippe Arnould: HSSN: an ontology for hybrid semantic sensor networks. IDEAS 2019: 8:1-8:10*

### 1.3.2 System Interrogation

The user creates and interacts with the connected environment through the system interrogation interface. In this module, we propose an Event Query Language specifically designed for connected environments, denoted EQL-CE. One uses the language to define a connected environment and all its components (e.g., environment, sensor network, events, and application domain). Then, EQL-CE queries could be used to generate instances of each component and manage the data. Finally, the user could compose event queries to define the targeted events of interest that he/she would like to detect. This entails detailing the event defining features and the sensors that could provide data for the detection of the aforementioned events. The proposed language operates at three different levels. At the top layer (i.e., conceptual level), the connected environment entities and relations are organized in the form of a graph. At the middle layer (i.e., logical level), the user composes queries that are written in a re-usable syntax. The bottom layer (i.e., physical level), handles the parsing of the logical queries into domain specific languages (e.g., SQL, SPARQL) and executes them. Every created component, or component instances as well as all the modifications are saved in the storage space. The details of the query language are presented in Chapter 3 (cf. Need 2). The EQL-CE framework is published [67] in the proceedings of the $23^{rd}$ International Database Applications & Engineering Symposium (IDEAS 2019). Moreover, another paper concerning the syntax and queries of EQL-CE is accepted (to appear) in the proceedings of $15^{th}$ ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '19):

- *Elio Mansour, Richard Chbeir, Philippe Arnould: EQL-CE: an event query language for connected environments. IDEAS 2019: 7:1-7:10*

- *Elio Mansour, Richard Chbeir, and Philippe Arnould. 2019. EQL-CE: An Event Query Language for Connected Environment Management. In 15th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '19) - Accepted*

### 1.3.3 Query Optimizer

Connected environments rely on sensor network data. The latter is provided by sensors (i.e., data sources). However, various issues might emerge due to the dynamicity of the environment. Sensors could breakdown, mobile sensors could leave or change locations, new sensors could enter the network, and the data required for specific event definitions might be lost. Therefore, we propose the query optimizer module to manage queries that became obsolete in time due the aforementioned issues. Chapter 4 details the proposed query optimizer module and focuses on addressing obsolete queries via query rewriting (cf. Need 3). We propose two main algorithms: the first automatically detects obsolete queries (i.e., queries that need rewriting) and the second performs the rewriting by replacing missing/unavailable query elements (e.g., sensors, data) by adequate successors. To do so, we detail how one can measure the similarity between sensors, and sensed data/event features.

### 1.3.4 Event Virtual Machine

This module represents a pluggable event detector that takes data objects (e.g., sensor observations) and event definitions (e.g., event queries) as input and detects what we called feature-centric events at the output. Feature-centric events, as the label indicates, are events that focus on one or more key features. For instance, time-centric events are any event that happened around a specific time interval, geo-centric events are events that occur in a specific location (or set of targeted locations), and temperature-centric events refer to any event related to temperature that occurred at any time and location. This pluggable module was designed to be reused in different application domains (i.e., with various data objects and event definitions at the input end). Chapter 5 details the entire event detection process (cf. Need 4), and how eVM is applied for sensor event detection in connected environments. Furthermore, the conducted experiments highlight the re-usability of eVM in other application domains (e.g. social event detection, conflict event detection), with different data objects (e.g., images, videos on social networks, conflict stories from news channels and papers) and event definitions (e.g., social, conflict events) as input. A detailed view of the framework is presented in Figure 1.6 to illustrate the inner composition of each module. The eVM framework is published [66] in the Transactions on Large Scale Data and Knowledge-Centered Systems XXXIX (special issue on Database and Expert Systems Applications). Moreover, the re-usability of the detector was demonstrated in another paper where the detector was used to detect social events. This paper is published [69] in the proceedings of $28^{th}$ International Conference on Database and Expert Systems Applications (DEXA 2017):

- *Elio Mansour, Richard Chbeir, Philippe Arnould: eVM: An Event Virtual Machine Framework. T. Large-Scale Data- and Knowledge-Centered Systems 39: 130-168 (2018)*

- *Elio Mansour, Gilbert Tekli, Philippe Arnould, Richard Chbeir, Yudith Cardinale: F-SED: Feature-Centric Social Event Detection. DEXA (2) 2017: 409-426*

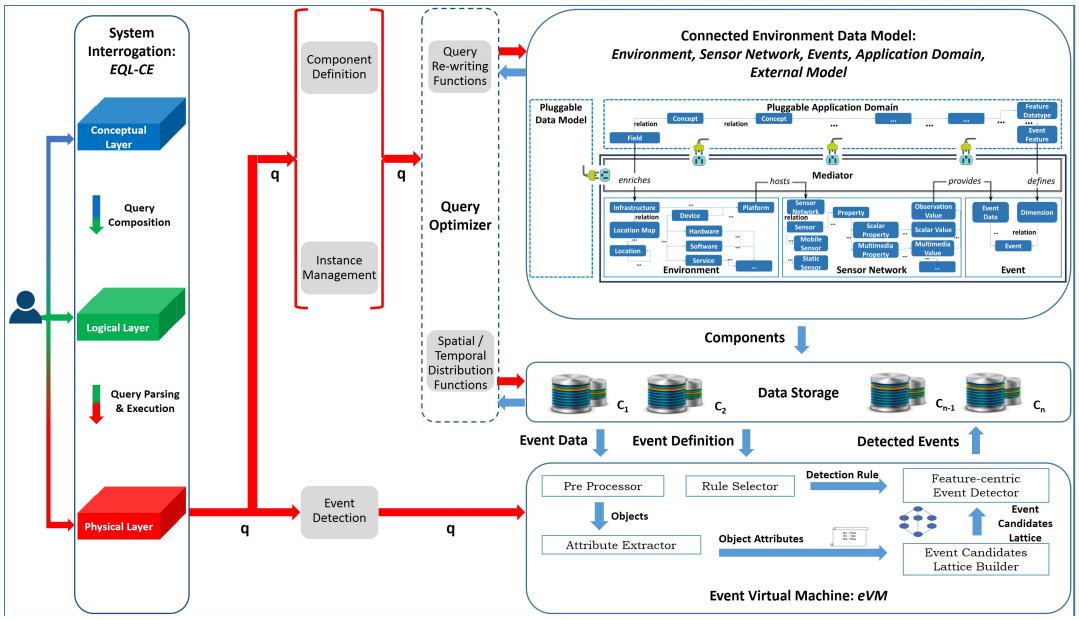


FIGURE 1.6: Detailed EDCE Framework View

## 1.4 Report Organization

The remainder of the thesis is organized as follows:

**Chapter 2** describes our ontology-based information model. We review related work on sensor network/data modeling. Then, we introduce the Hybrid Semantic Sensor Network (HSSN [68]) ontology where we enrich the environment and sensor network modeling with diverse types of (i) sensors (e.g., static, mobile, simple nodes, multi-sensor devices, scalar, multimedia); (ii) deployment platforms (e.g., physical infrastructures, electronic platforms and devices); and (iii) sensed data (e.g., scalar, multimedia sensor observation). We do so by extending the widely used SSN [44] ontology without compromising the re-usability of the data model in different contexts. Finally, we evaluate the performance, clarity, consistency, and accuracy of our additions.

**Chapter 3** describes the event query language that one uses to interact with the entire framework. We review existing works on query languages and propose EQL-CE (an event query language adapted to connected environments [67]). We do not address all challenges related to proposing a query language for connected environments but mainly focus on the following: (i) covering all connected environment components and not only events; (ii) covering common query types; (iii) covering various datatypes; (iv) considering spatial distribution of sensors over the space, and temporal distributions of sensor observations over time; (v) handling the connected environment dynamicity; and (vi) providing a re-usable syntax. We propose a three layered framework for the language (i.e., conceptual, logical, and physical layers). We detail the syntax of each connected environment component and the various query types. Finally, we detail an illustration example and the experimental protocol for the evaluation of the language.

**Chapter 4** focuses on one specific need regarding the language: coping with the dynamicity of the environment. We point out various factors that contribute to the dynamicity of the environment (e.g., sensor mobility/reliability, platform compatibility, data volatility, missing data/features). Then, we focus on two main challenges: (i) sensor mobility & reliability; and (ii) missing data/features. We propose a query optimizer module that complements the previously detailed event query language EQL-CE. The optimizer rewrites event queries that became obsolete due to the dynamicity of the environment. We detail two algorithms for obsolete query detection and rewriting. Finally, we evaluate the algorithms' complexities and propose an experimental protocol to assess the accuracy, and performance of the both processes (obsolete query detection and rewriting).

**Chapter 5** describes an event detector module that takes event definitions (EQL-CE event queries) and data objects (e.g., sensed data) as input in order to detect the targeted events. First, we review various event detection works. Then, we present a generic event detection framework that relies on Formal Concept Analysis, a clustering technique denoted FCA. We detail the process of event detection from the incoming input query to the detected event at the output. Finally, we present the experimentation and results.

**Chapter 6** concludes the report with a recap of all the aforementioned chapters and discusses in details the next steps and potential future research directions.

# Chapter 2

# A Data Model For Hybrid Connected Environments

*"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time."*

*— Thomas A. Edison*

Recent advances in sensor technology, have allowed sensor networks to impact a large spectrum of domains (e.g., health, environment). These networks generate heterogeneous data, that is hard to represent, share, and integrate. Therefore, semantic web techniques, such as ontology-based data models, have been widely adopted for information representation in sensor network modeling.

However, some existing works do not fully address the following challenges: (i) representing different sensor types (e.g., mobile sensors), in order to enrich the network with different data and ensure better coverage; (ii) representing a variety of platforms (e.g., environments, devices) for sensor deployment, therefore, integrating new components (e.g., mobile phones); (iii) representing the diverse data (i.e., scalar, multimedia) needed for various applications (e.g., event detection); and (iv) proposing a generic model to allow re-usability in various application domains.

In this chapter, we propose the Hybrid Semantic Sensor Network ontology. HSSN extends the Semantic Sensor Network ontology. SSN is already re-usable (widely used in different contexts), and considers various platforms. Our proposal extends the representation of sensors, the sensed data/properties, and deployment environments. We evaluate the consistency of our ontology structure, the clarity of the used nomenclature, the accuracy of our additions, and their impact on performance.

## 2.1 Introduction

The convergence of Internet, communications, and information technologies coupled with recent advances in engineering have paved the way for Wireless Sensor Networks (WSNs) to impact more and more application domains [92] such as environmental sensing, inventory monitoring, habitat management, military, and medical fields. Also, the evolution of sensor technology has impacted mobile phones and other devices. Various sensors (e.g., gyroscope, camera, microphone, ambient light, GPS, compass) are nowadays embedded in smart phones. Useful information can be inferred from mobile phone sensor data for various purposes (e.g., detecting traffic congestion through GPS data, monitoring pollution levels in a city). Therefore, allowing mobile sensors or any mobile device with sensing capabilities to seamlessly integrate wireless sensor networks is very beneficial from a knowledge extraction point of view.

Nonetheless, these networks would produce huge amounts of heterogeneous data, that have to be collected, processed, analyzed, and visualized in order to provide various services and overall decision making aid for network managers. Representing, sharing, and integrating the aforementioned data, which lack semantics [103], is a challenging task. In order to address this challenge, semantic web techniques, such as ontology-based data models, have been adopted for their information representation. However, the existing works on sensor network modeling [6, 10, 15, 32, 40, 79, 84] are restrictive due to the following issues:

- *Lack of platform diversity:* existing approaches [10, 15, 32, 40] do not consider equipment with embedded sensors (e.g., smart phones, drones, machines) as platforms, in addition to traditional platforms (e.g., buildings, cities, offices) where sensors are deployed. The platform representation is limited to simple platforms, and does not support complex/nested platforms. Extending the representation, by both considering and detailing the representation of various types of platforms, allows better expressiveness of components in the network, nested platforms, and dynamic, collaborative sensing activities (e.g., crowdsensing).

- *Lack of sensor diversity:* these works [32, 79] do not represent different sensor types. This entails modeling (i) mobile sensors capable of moving in the environment as well as static (immobile) ones; (ii) simple sensor nodes that make observations and send them to a base station as well as multi-sensor devices capable of sensing, storing, managing, and communication data; and (iii) sensors capable of sensing scalar as well as multimedia properties. Providing a more detailed and diverse sensor representation that considers various attributes (e.g., mobility) improves network coverage, and allows sensor tracking and dynamic sensing.

- *Lack of data diversity:* most works [10, 15, 40] cover scalar environment properties (i.e., mainly focus on scalar data such as temperature, motion, and neglecting multimedia data such as sounds, images, and videos). Since several devices are capable of sensing both types, and data diversity is required for different application purposes (e.g., event detection), it is important to cover scalar and multimedia data in the representation. This enriches the representation of the environment and the descriptions of the events that are detected within its premises.

- *Lack of re-usability:* these approaches [40, 79] are heavily linked to a specific application domain. The sensor network modeling should remain generic and re-usable in different contexts. Moreover, having domain specific knowledge could increase the semantic complexity/computation costs of the data model.

Then, there is a need for an appropriate semantic representation of sensor networks that adds meaning to the heterogeneous sensor data, and ensures data linkability. Also, such a representation should be generic and diverse (concerning platforms, sensors, and data). Finally, the proposal should be light, performance-wise, and able to avoid the semantic complexity that could heavily impact performance in some cases. This benefits specific applications (e.g., critical event detection) where responsiveness and light processing costs are critical.

To answer these challenges, we present here an extension of the widely used Semantic Sensor Network ontology (SOSA/SSN) [44] called HSSN. It allows the representation of hybrid sensor networks, i.e., networks containing mobile/static sensors, scalar/multimedia properties, and infrastructures/devices as platforms where sensors are deployed. We chose to extend SSN since it is already re-usable in various contexts and allows the representation of different platforms. Nonetheless, sensor and data diversity are not fully developed. Our proposal adds diverse data, sensors, and details the description of various platform types. In addition, HSSN does not contain domain specific knowledge and can be easily aligned with other ontology models (e.g., a mobile phone [47], smart building ontology [94]).

The rest of this chapter is organized as follows. Section 2.2 illustrates a scenario that motivates our proposal. Section 2.3 reviews related work regarding mobility, platforms, and sensed data. Section 2.4 details the HSSN ontology. Section 2.5 describes the implementation and an illustration example. Section 2.6 details the experimental setup and results. Finally, Section 2.7 concludes the chapter and discusses future research directions.

## 2.2 Motivating Scenario

To highlight the utility of our proposal, we choose the following scenario. Consider a smart mall/shopping center, where clients shop, eat, watch movies, and interact with others through various activities (cf. Figure 2.1). In order to optimize client comfort, health, security, and overall visiting experience, the smart mall relies on a set of sensors ($s_1$-$s_9$) to monitor the environment. Video surveillance cameras ($s_1$-$s_6$) monitor security related events. Humidity and $CO_2$ sensors ($s_7$ and $s_8$ respectively) make observations that help the HVAC system (Heating, Ventilation, and Air Conditioning) regulate the indoor air quality. Finally, $s_9$ is an indoor temperature sensor. The data produced by this sensor is used to keep the mall cool/warm for optimal comfort. Although the mall's sensor network generates the required data for these applications, many improvements still need to be integrated:

**Need 1.** Monitor temperature/air quality readings by zone: since areas might have different requirements (e.g., cool temperature for food storage in the grocery store, normal temperature in Shop 1). This requires temperature (using $s_9$) and air quality readings (using $s_7, s_8$ combined) from each zone. The current mall

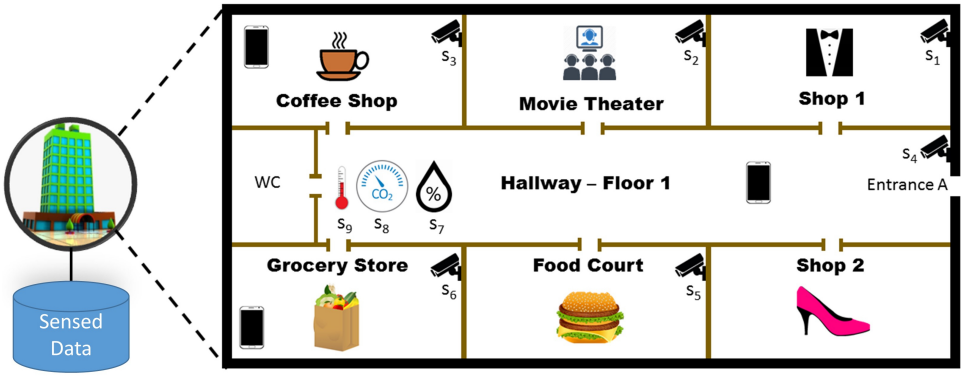


FIGURE 2.1: Smart Mall Example

setup does not allow this since only the hallway is monitored by temperature and air quality sensors.

**Need 2.** Provide better temperature/air quality readings: relying on measures from a multitude of sensors (instead of only one) allows a more precise monitoring of the environment (e.g., calculate average values). This requires a collaboration between multiple temperature/air quality sensors (similar to $s_9$, $s_8$, and $s_7$), each providing measures. In the current setup, this is not possible since there is only one temperature and one air quality sensor in the mall.

**Need 3.** Keep track of client density in the mall: it is useful to know the number of occupants in each zone since this affects heating and ventilation for instance. Also, clients are useful for tracking suspicious/interesting behaviours in the mall (e.g., groups of people moving together, same person visiting the same places multiple times). The cameras ($s_1$-$s_6$) are used by security agents to monitor security related events. They cannot be used to track each client's location. Therefore, a more advanced solution is needed to achieve this need.

**Need 4.** Cover all areas of the mall: this is critical for the security and safety of the clients, since some zones are unmonitored (e.g., Shop 2). This means that any event that happens in these areas is not detected. In the current setup, the 9 deployed sensors cannot cover the entire mall space. Many uncovered areas exist (e.g., no temperature monitoring in the movie theater, no video surveillance in Shop 2).

**Need 5.** Provide a rich documentation of every critical event: the events that occur in the mall are detected based on the sensed data. Therefore, in order to increase the understanding of these events (e.g., when reporting incidents, or providing evidences), event descriptions should be enriched by a variety of sensed multimedia and scalar properties (e.g., video, audio, image, temperature, humidity). Currently, if the mall managers wanted to report an attack incident (e.g., gunshot) they cannot provide a rich description of the event. They can only rely on video surveillance footage (from $s_1 - s_6$). There is no variety of sensed properties that could enrich the description of such an event (e.g., noise levels to confirm the gunshot, motion data to describe how people ran away). In order to answer this need, a bigger variety of data should be sensed.

**Need 6.** Adapt to changing event detection needs over time (i.e., detect various new events in the future): sometimes new/spontaneous events need to be detected, the mall should be able to sense the required data and detect these events.

However, the current setup is static. The sensor configuration/deployment and sensed data cannot be easily modified. This doesn't allow the detection of new events.

In order to address these issues, the mall managers would need to add more sensors to cover all zones. This ensures full coverage of the mall (Need 4), and allows multiple observations from each zone for aggregation (Needs 1-2). In addition, they could replace the cameras with more advanced ones that enable image processing for tracking purposes (Need 3). However, this increases the equipment, maintenance, and implementation costs without addressing Needs 5 and 6. A more appropriate solution would be to allow integrating visitors' mobile phones (since they embed sensors) as mobile sensors in the mall's network, while avoiding excessive resource consumption from the devices (e.g., draining a phone's battery). This provides the following benefits:

**Benefit 1.** Sensor mobility provides observations from different areas of the mall (based on visitor movements). Multiple sensors can therefore collaborate in order to calculate more reliable air quality/temperature measures by zones (Needs 1 and 2).

**Benefit 2.** Mall visitors can easily be tracked using their connected mobile phones (Need 3). Location information can also be used to calculate the number of occupants of each zone. This helps discovering crowded areas (where air quality/temperature monitoring is most critical) and uncovered areas (Need 4).

**Benefit 3.** The network becomes hybrid with the usage of various sensors from different devices in addition to the static mall sensors. This helps cover a wider array of observed properties (Need 5), i.e., multimedia properties (e.g., audio, video, images) and scalar properties (e.g., temperature, movement, humidity).

**Benefit 4.** These devices allow more adaptability and flexibility when it comes to the changing event detection needs (Need 6). They provide a diversity of hardware (e.g., sensors) and software (e.g., applications, services) that can be adapted to the detection needs.

However, when adding mobility, diverse data, and devices to the network, the following challenges emerge:

**Challenge 1.** How to expressively describe locations in the mall?

**Challenge 2.** How to consider ad-hoc devices in the network?
How to query them based on their capabilities (e.g., without draining their batteries, based on the services they provide or their processing power)?
How to represent the services that they provide?

**Challenge 3.** How to track locations and coverage areas of mobile sensors?

**Challenge 4.** How to collect scalar/multimedia observations from sensors?

Other challenges also exist when modeling sensor networks (e.g., how to represent temporal data, how to address data volatility, how to model inter-platform ties/interactions). However, we address here the aforementioned four challenges from a

data modeling perspective by proposing an extension of the semantic sensor network ontology that includes mobility, platform, and data related concepts. More precisely, we model mobile sensors, devices, their coverage areas, and locations. We also detail the description of various platform types such as infrastructures where sensors are deployed, and the various (scalar/multimedia) data they produce.

## 2.3 Related Work

In this section, we study the features of existing sensor and semantic-based sensor network modeling. We also review some works about sensor mobility, mobile phone sensing, crowd-sensing, deployment platforms, and semantic representation of multimedia data. We compare these works based on the following criteria:

**Criterion 1.** *Sensor Diversity:* Integrating mobile sensing devices in the sensor network is beneficial for coverage of large areas, and giving users an active role in monitoring their own environments (crowd-sensing). Also, in addition to simple sensor nodes, it is beneficial to have multi-sensor devices capable of sensing, processing, communicating, and storing data. Moreover, these devices are capable of providing various services to the users. Finally, the network is enriched with more data and datatypes when equipped with sensors capable of sensing scalar and/or multimedia data. Therefore, for the aforementioned reasons, we propose sensor diversity as a comparison criterion for existing works. This criterion indicates $\{YES, NO, PARTIAL\}$ if different types of sensors exist in the sensor network (e.g., mobile/static sensors, simple sensor nodes/multi-sensor devices, scalar/-multimedia sensors).

**Criterion 2.** *Platform Diversity:* Allowing different platforms enables deploying sensors in various environments/infrastructures, embedding them in various devices, or even having nested platforms (e.g., devices in buildings). It is also beneficial to detail the description of the different platforms. When modeling physical world environments (i.e., infrastructures) such as cities and buildings, it would be interesting to model spatial elements and locations since this enables location-based tasks (e.g., monitoring zones of interest, the impact of neighbouring areas). However, when modeling devices other elements are more interesting. For instance modeling the hardware, software, and the provided services helps query devices based on their capabilities (e.g., querying mobile phones without draining their batteries). This criterion states $\{YES, NO, PARTIAL\}$ if the approach allows sensor deployments on different platforms and if the description of the latter is detailed.

**Criterion 3.** *Data Diversity:* This diversity enriches the representation of the network, and benefits various application purposes such as event detection where a combination of scalar (e.g., temperature, humidity) and multimedia (e.g., audio, video) data might be necessary for the detection of specific events. Therefore, we consider data diversity as an important criterion when modeling sensor networks. This criterion denotes $\{YES, NO, PARTIAL\}$ the approach's ability to handle various data/properties (e.g., scalar, multimedia, both).

**Criterion 4.** *Re-usability:* A re-usable approach does not contain domain specific knowledge and therefore is compatible with different application purposes. Therefore, we consider the importance of re-usability when comparing existing works.

This criterion indicates $\{YES, NO, PARTIAL\}$ if the approach is re-usable in various contexts.

Many works [6, 10, 15, 19, 31, 32, 33, 34, 40, 44, 65, 70, 71, 79, 85] have evolved around knowledge representation for sensors and sensor networks. In the following, we detail each work separately (the purpose or name of the model is highlighted in bold font). Finally, we evaluate them based on the aforementioned criteria.

### 2.3.1 Sensor Diversity

**Adaptiveness In WSN.** In [10], the authors tackle the wireless sensor network adaptivity problem by proposing a two-phase solution. In the first phase, nodes in the network are organized as clusters and execute an algorithm in order to calibrate the sensed data. The data and state of each sensor in the cluster are reported to the cluster head node. Then, the latter executes an ontology-driven algorithm to determine the future state of the network. The sensor node ontology designed by the authors focuses mainly on features that describe the sensor nodes, their functionality (for sensed data calibration), and their state such as CPU, memory, and power supply current states (in order to determine the future state of the WSN).

**Sensor Network Data.** In [34], the authors propose an ontology with the main focus of searching distributed and heterogeneous sensor data. They provide a two-layer ontology that uses the IEEE Suggested Upper Merged Ontology (SUMO) aligned with two sub-ontologies: (i) the sensor data ontology (SDO); and (ii) the sensor hierarchy ontology (SHO). To enable interoperability, the authors propose the EPO (Extension Plug-ins Ontologies) module. It allows developers to integrate domain-specific ontologies with the universal ontology. Each plug-in ontology should implement the knowledge representation for a particular domain of sensor data/networks and establish the connection with the SUMO ontology.

**Sensor-Mission Assignment.** In [40], the authors emphasize on sensor to task assignments in sensor networks. Their work approaches the sensor-mission assignment problem from a Semantic Web perspective. The core of their contribution is a set of ontologies describing missions, tasks, sensors, and deployment platforms. Semantic reasoning is then applied on used ontologies to recommend the deployment of specific sensors over recommended platforms in order to better execute the required tasks.

**SSN/SOSA.** In [44], the authors propose SOSA/SSN[3], a set of ontologies both published as a W3C (World Wide Web Consortium) recommendation and as an OGC (Open Geospatial Consortium) implementation standard. This set includes a lightweight core module called SOSA (Sensor, Observation, Sampler, and Actuator) and a more expressive extension module called SSN (Semantic Sensor Network). Together they describe systems of sensors and actuators, observations, the used procedures, the subjects and their properties being observed or acted upon, samples and the process of sampling, and so forth. A sensing overview of the ontologies is presented in Figure 2.2. The authors define sensors as physical objects that observe, transform incoming stimuli (related to certain properties) into observations, thus producing an output (a digital value of the observation). To do so, sensors implement sensing

[3] https://www.w3.org/TR/vocab-ssn/

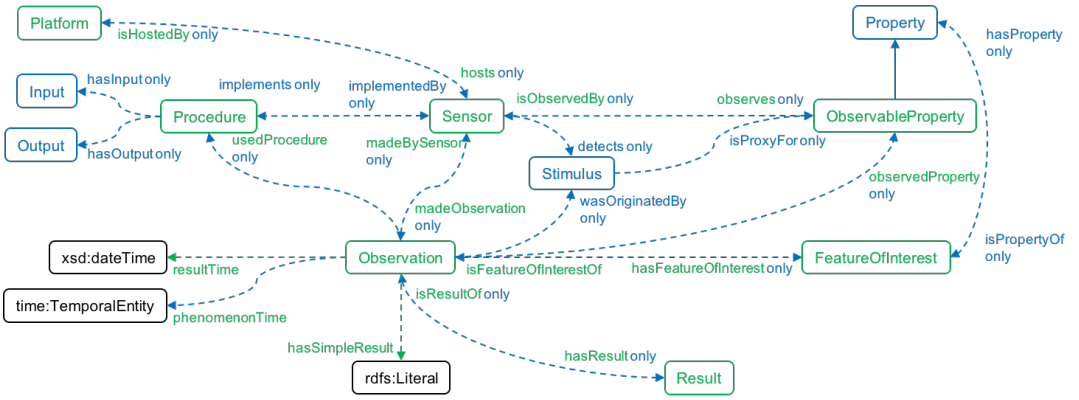


FIGURE 2.2: The SOSA/SSN ontologies: A sensing overview

methods that describe how the sensor observes. Sensors are deployed on a platform via a certain deployment process. Platforms could be real world infrastructures (e.g., building) or electronic devices (e.g., mobile phones). However, the representation of the different platform types is not detailed. Finally, SOSA/SSN propose simple sensor node representation, as well as (sensing) systems/devices. However, the authors do not propose any mobility-related concepts, to represent mobile sensors/devices, nor multimedia data/properties.

**MSSN-Onto.** In [6], the authors propose an extension of SSN, denoted MSSN (Multimedia SSN), where they detail the technical aspects of multimedia data (e.g., video, audio segments, frequencies) and introduce domain specific knowledge related to event detection. They introduce the following new key concepts: (i) multimedia sensor; (ii) event; (iii) atomic event; and (iv) complex event. They consider that a multimedia sensor is a sensor capable of producing audio, video, or/and image data/observations. The authors also provide technical descriptions related to the multimedia data based on various metadata standards (e.g., MPEG-7, EXIF). However, the relation between sensors, observable properties, produced scalar or multimedia observations, and their respective metadata lacks clarity. Moreover, the authors do not detail the representation of different types of platforms. Finally, MSSN does not consider sensor mobility, i.e., they authors consider that sensors do not change locations.

**SensorML.** In [19], the authors propose an XML schema for defining the geometric, dynamic, and observational characteristics of a sensor. The purpose of the sensor description is to (i) provide general sensor information in support of data discovery; (ii) support the processing and analysis of the sensor measurements; (iii) support the geo-location of the measured data; (iv) provide performance characteristics (e.g. accuracy, threshold, etc.); and (v) archive fundamental properties and assumptions regarding sensors. SensorML provides a functional model for sensors, does not detail the description of hardware, and separates the sensor from its associated platform(s) and target(s). Although, SensorML could be used to represent different sensors, it does not detail hardware related concepts, nor the impact of different sensors (e.g., mobile sensors) on the environment (e.g., impact of mobility on locations and coverage areas).

### 2.3.2 Platform Diversity

**SOSA/SSN.** In the SSN ontology [44], the authors state that sensors are deployed on platforms. The concept `sosa:Platform` defines the latter as entities that could host other entities, particularly sensors, actuators, samplers, and other platforms. The authors provide some platform examples (vehicle, ship, satellite, aircraft, cellphone, human). SSN also introduces the concept `System`, that can integrate various sensors, actuators, and samplers. Therefore, it provides a foundation for sensor deployment on various platforms (e.g., traditional deployment on platforms, embedding sensors in systems and devices).

**MSSN-Onto.** In [6], the authors extend the SOSA/SSN ontologies in order to enrich the data description by integrating multimedia sensor observations. Therefore, MSSN benefits from the already existent `sosa:Platform` concept. The authors do not present any extensions regarding platforms, and limited their additions to data and event related concepts/properties.

**Noise Pollution Monitoring.** In [65], the authors use mobile phones as sensors in order to monitor noise pollution in cities. They do not represent various sensor types, but only consider embedded sensors on mobile phones. In addition, the authors mainly focus on cities as platforms where sensors are deployed. They also describe some spatial constraints of the environment (i.e., the city) in order to map each noise pollution observation to a specific location.

**Wildfire Monitoring.** In [32], the authors rely on traditional deployment of sensor nodes in the wilderness to detect fire events. They use temperature, relative humidity, and barometric pressure observations from the wilderness (the platform where the sensors were deployed). They combine the sensed data with GPS information to localize the detected fire events.

**S3N.** In [85], the authors propose the Semantic Smart Sensor Ontology to represent smart sensors, their different computation and communication profiles, and how different algorithms may be selected and loaded, potentially at run-time. This work focus on the sensing process and different functionality of the smart sensors. The latter have not been detailed nor exploited as platforms that could potentially host other platforms (nested platforms, different types of platforms).

### 2.3.3 Data Diversity

**Ear-Phone.** In [79], the authors monitor noise pollution in an urban, by sensing noise levels using occupants' mobile phones. The noise data is combined with geolocations in order to generate a noise pollution map of the area. Since this approach's purpose is to monitor noise pollution, it integrates audio data without considering other multimedia data (e.g., video, image). Also, the authors focus more on generating the noise level map instead of representing the multimedia contents.

**Sensor Image Interpretation.** In [33], the authors represent images for object recognition purposes. They rely on images observed by sensors and propose a recognition method based on an ontology which has been developed by experts of the domain. In order to give objects a semantic meaning, the authors develop a matching process between an object and the concepts of the aforementioned ontology.

**Object-based Image Retrieval.** In [71], the authors only represent images, since they propose an approach for object-based image retrieval. The images they wish to represent do not come from sensors. However, they propose an ontology where the detail image describing features (e.g., size, shape, position, colors). The authors do not address other scalar or multimedia data.

**Sense & Sens'ability.** In [15], the authors address the issue of processing the huge amounts of sensed data that originate from sensor networks. More specifically, they focus on the heterogeneity of the data. Hence, they provide a semantic model for heterogeneous sensor data representation. Even though they target the representation of various types of sensor data, the authors do not detail the description of scalar/multimedia data. Their contribution evolves around adding semantics (i.e., by creating an ontology) to the description instead of using non semantic data models (e.g., they compare their work with sensor data representation using XML).

**MSSN-Onto/SOSA/SSN.** The authors in [6] represent multimedia data in sensor networks. For each multimedia observation value, the authors associate data descriptors (denoted media descriptors), and data segments (denoted media segments). Their proposed ontology, MSSN, complements the SOSA/SSN ontology [44] since the latter does not cover multimedia contents nor multimedia sensors.

### 2.3.4 Re-usability

**Noise Pollution Monitoring.** In [65], the authors propose a noise pollution monitoring solution in a city using mobile phones to sense noise. The authors enrich the sensed information by allowing users to add contextual information to their sensor observations. The approach is task-centric and needs to take into consideration additional concepts/properties for it to be re-used in different contexts.

**P-Sense.** In [70], the authors present P-Sense (Pollution-Sense): a system for air pollution monitoring and control. The latter combines the use of everyday mobile devices, such as smart phones, GPS technology and location-based services, and sensors to collect air pollution data at various granularities of a city. Unfortunately, P-Sense is limited to observations that contribute in detecting pollution levels.

**Air Quality Monitoring.** In [31], the authors propose a device, the Mobile Sensing Box (MSB), that can be mounted on public transportation means such as buses, and another device, the Personal Sensing Device (PSD), that can be mounted in people's cars. These devices only monitor scalar environmental properties related to air pollution in a city.

**MSSN-Onto.** In [6], the authors propose a multimedia wireless sensor network ontology for event detection purposes (the authors include concepts related to atomic, complex events, and event detection/composition).

**SOSA/SSN.** The SSN ontology [44] remains generic and re-usable in various contexts since it is extensible and does not contain any concepts that link it to any specific application.

### 2.3.5 Discussion

Table 2.1 shows that none of the aforementioned works fully considers the entire list of criteria.

**Sensor Diversity.** These models do not integrate sensor diversity in their representation of sensor networks. Considering mobility adds challenges related to properly locating/tracking mobile sensors, and updating their coverage areas when they move. Nonetheless, integrating mobility improves coverage and dynamic sensing. SSN considers simple sensor nodes and multi-sensor devices. MSSN adds to that scalar and multimedia sensors. However, none of them considers mobile sensors. Moreover, the other works either rely on static or mobile sensors. None of the compared studies fully integrates (i.e., representing scalar/multimedia, simple nodes/multi-sensor devices, and static/mobile sensors).

**Platform Diversity.** SSN and MSSN (since it extends SSN), have the ability to consider various types of platforms and nested platforms at once. However, they do not provide additional representation of each platform type. A physical, real world environment, is not distinguished from a machine or device that acts as a platform (by hosting sensors). This denies the ability to query an infrastructure based on spatial constraints, or an electronic platform based on its hardware, software, or services. Other works, do not consider different types of platforms.

**Data Diversity.** SSN lacks multimedia data in its representation of the sensor network. This lead to the proposal of the MSSN ontology which integrates multimedia and scalar data. This is very useful for a variety of applications (e.g., complex event detection) where both data types are needed. Although this provides data diversity, we do not choose to extend MSSN for the following reasons: (i) the mapping between multimedia properties, sensors, and observation values lacks clarity; (ii) MSSN does not consider mobility, i.e., sensor locations do not change and the authors state that the location of a sensor is equivalent to its coverage area (this statement is no longer true when considering mobile sensors); and (iii) MSSN is heavily linked to one application domain (event detection) and contains event related knowledge. Therefore, key concepts from MSSN need to be integrated in order to model multimedia data in our proposal. The other compared works only represent the data that is required for the objectives of their study.

**Re-usability.** The SSN ontology [44] is a culmination of much of the related work on semantic sensor networks and is the most widely used [87]. In addition, SSN is extensible, facilitates alignments with other standards, and allows the integration of new concepts. Therefore, we propose to extend SSN (since it is already re-usable), in order represent in details diverse sensors (e.g., static, mobile), platforms (e.g., infrastructures, devices), and sensed data / properties (e.g., scalar, multimedia).

TABLE 2.1: Sensor Network Modeling - Related Work Comparison

| Criteria | SSN | MSSN | Others | |
|---|---|---|---|---|
| | [44] | [6] | [33, 65, 70, 71, 79] | [10, 15, 19, 31, 32, 34, 40, 85] |
| *Sensor Diversity* | *PARTIAL* | *PARTIAL* | *NO* | *NO* |
| *Platform Diversity* | *YES* | *YES* | *NO* | *NO* |
| *Data Diversity* | *NO* | *YES* | *PARTIAL* (audio or image) | *NO* |
| *Re-usability* | *YES* | *PARTIAL* | *NO* | *NO* |

## 2.4 HSSN Ontology

In this section, we detail our proposed extension of the SSN ontology, and mainly our additions related to: (i) sensor diversity; (ii) platform diversity; and (iii) data diversity. The following prefixes `sosa:`, `ssn:`, `mssn:`, `time:`, and `hssn:` refer to the SOSA [44], SSN [44], MSSN [6], TIME [51], and HSSN [68] ontologies respectively. We extend SSN/SOSA, integrate multimedia related concepts and properties from MSSN, and enrich some concepts with temporal concepts from TIME. We begin first by describing sensor-related concepts.

### 2.4.1 Sensor Diversity

#### 2.4.1.1 Sensor Mobility

Figure 2.3 illustrates the sensor types added in HSSN. The concept `Sensor` already exists in SOSA/SSN, where mobility is not extensively developed. Therefore, we add two child concepts of `sosa:Sensor`: (i) `hssn:MobileSensor`, describing any sensor that has the ability to move or change location; and (ii) `hssn:StaticSensor`, a sensor that does not change location in time. Mobile sensors are basically sensors that are embedded on a mobile platform (e.g., smart phone sensors, sensors deployed on drones, vehicules, or any mobile equipment/machine). This allows the sensor network to have diverse sensor types (cf. Criterion 1 - Section 2.3).

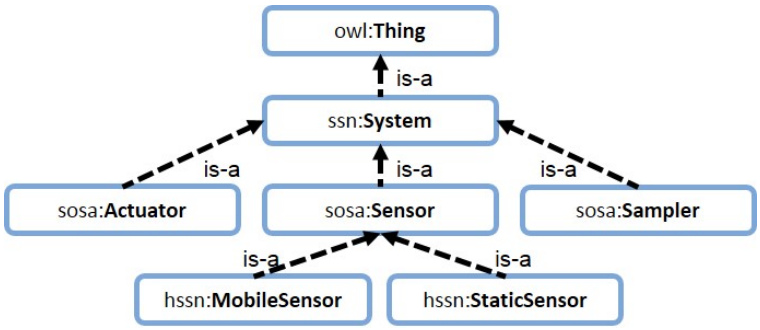


FIGURE 2.3: HSSN Sensor View

#### 2.4.1.2 Sensor Tracking

Every sensor has a `mssn:Location`. To consider mobility, one should be able to locate any sensor at all times. The object property `hssn:isCurrentlyLocatedAt` maps each sensor to its current `mssn:Location` (cf. Challenge 3 in Section 2.2). This is specifically important for tracking mobile sensors, since static sensors do not change locations (cf. Figure 2.4). A `hssn:hasPastLocation` property is added to retrieve the previous positions of any sensor, and also a `hssn:hasLocationTime` (cf. Figure 2.5) property is added to map these positions to time instants or intervals in order to track sensors (temporal entities are extracted from TIME ontology [51]).

#### 2.4.1.3 Coverage Area

Each `sosa:Sensor`, mobile or static, has a `hssn:CoverageArea` (cf. Figure 2.6), a geographical zone described by a specific geometric shape. The sensing activity of a `sosa:Sensor` is operational within the premise of its coverage area (i.e., any happening outside of this zone is not detected by the `sosa:Sensor`). In order to

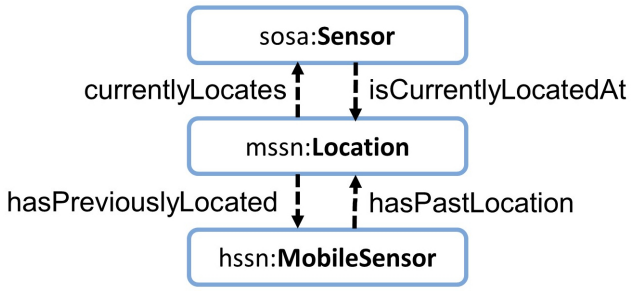


FIGURE 2.4: Sensor/Location Mapping

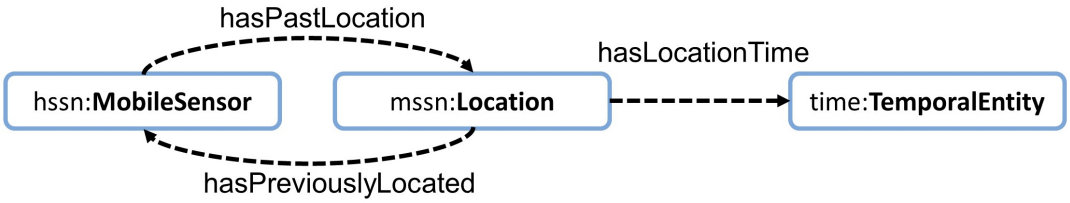


FIGURE 2.5: Previous Location/Time Mapping

represent coverage areas, we consider the following: (i) a `hssn:CoverageArea` is bound to the sensor's current `mssn:Location`; and (ii) the geographical spread of a `hssn:CoverageArea` is affected by the sensing range and sensing angles (horizontal and vertical orientation) of the concerned `sosa:Sensor`. We represent the coverage area as a sector of space (Figure 2.7 and 2.8 show the horizontal/vertical slices of the space respectively) where S is the focal point (the sensor's current `mssn:Location`), $\alpha, \beta \in [0; 2\pi]$ are the angles that define the horizontal/vertical rotational spread of the coverage area respectively, and the distance $SA = SB$ is the sensing range that defines the extent of the coverage area. The angles and range depend of the sensor's capability properties. For instance, a temperature sensor has $\alpha = \beta = 2\pi$, but a surveillance camera has $\alpha = \frac{\pi}{4}$, $\beta = \frac{\pi}{6}$ if the camera lens is limited to a 45° horizontal angle, and a 30° vertical angle. Similarly, the sensing range varies from one sensor to another (e.g., 10, 20, 50 meters).

The composition of a `hssn:CoverageArea` is explained in Figure 2.9. The `hssn:SensingLocation` is equivalent to the sensor's `mssn:Location`, and the angles and range of the `hssn:CoverageArea` are equivalent to the sensor's angles and range (i.e., `hssn:HorizontalAngle`, `hssn:VerticalAngle`, and `hssn:Range` respectively) properties that we added in HSSN as part of a system's properties. Since static sensors are immobile, it is easy to know their coverage areas using the sensor's location, and its sensing range and angles. In contrast, knowing the coverage areas of mobile sensors is more challenging, since these areas move when the sensors move. In order to keep track of these changes, the object property `hssn:currentlyCovers` maps each `sosa:Sensor` to its current `hssn:CoverageArea` (cf. Figure 2.10). Also, the property `hssn:hasPastCoverageArea` maps mobile sensors to their respective sets of previous coverage areas (cf. Challenge 3 in Section 2.2). Finally, `hssn:hasCoverageTime` is the property that maps previous coverage areas to temporal entities (i.e., time instant or interval from TIME ontology [51]) for tracking purposes (cf. Figure 2.11).

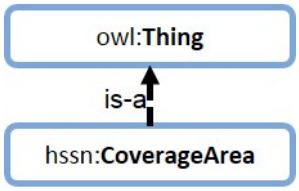


FIGURE 2.6: Coverage Area

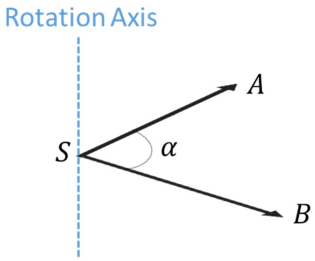


FIGURE 2.7: Horizontal Spread

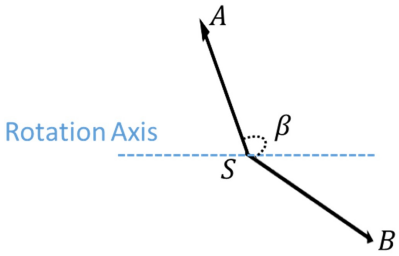


FIGURE 2.8: Vertical Spread

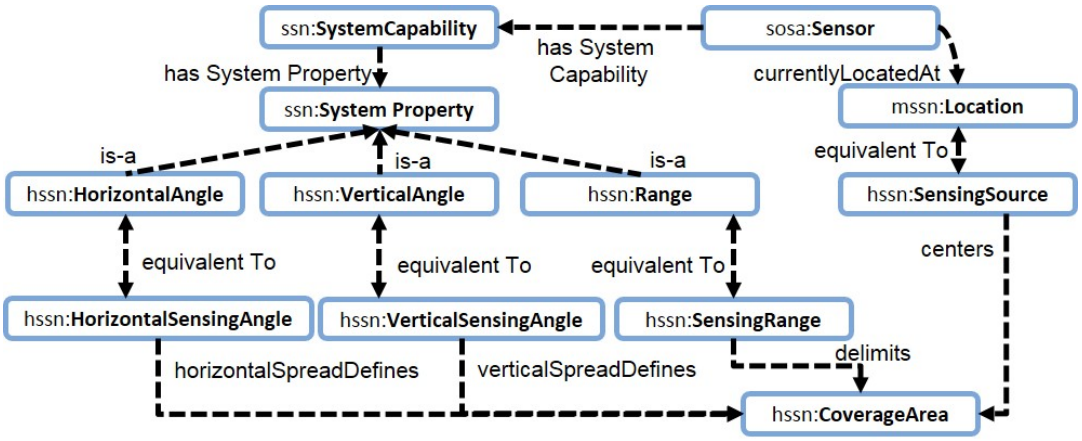


FIGURE 2.9: Coverage Area Composition

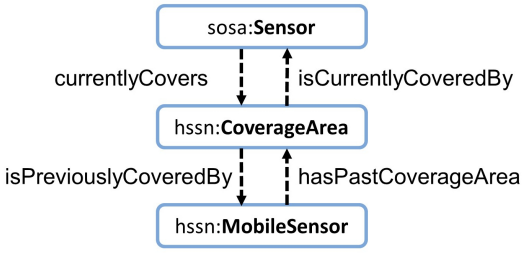


FIGURE 2.10: Sensor/Coverage Area

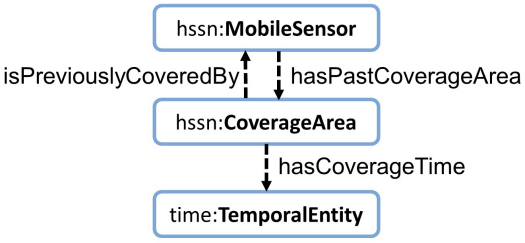


FIGURE 2.11: Coverage Area/Time

### 2.4.2 Platform Diversity

#### 2.4.2.1 Infrastructure Representation

In SSN [44], sensors are deployed on platforms. In Figure 2.12, we define the following child concepts of `sosa:Platform`: (i) `hssn:Infrastructure`, a physical environment having locations where sensors could be deployed (cf. Challenge 1 in Section 2.2); and (ii) `hssn:Device`, an electronic equipment where sensors could be embedded (cf. Challenge 2 in Section 2.2). This allows different types of deployments such as the traditional deployment in environments (e.g., buildings, malls) or nested deployment of multi-purpose devices that in turn embed sensors (e.g., mobile phones). This provides platform diversity (Criterion 2 cf. Section 2.3). Every `hssn:Infrastructure` describes a specific physical environment where sensors are deployed. Therefore, infrastructures can host platforms such as other infrastructures (e.g., cities host buildings) and devices (e.g., buildings host mobile phones). However, devices can embed systems of sensors, actuators, and samplers but cannot host infrastructures (e.g., buildings). Each `hssn:Infrastructure` is described by a `mssn:Location Map` which contains (`hssn:isComposedOf` property) a set of `mssn:Location` (cf. Figure 2.13). For example, a building is an `hssn:Infrastructure` that has a `mssn:LocationMap`. The latter describes the spatial relations between individual `mssn:Locations` in the building such as floors, offices, etc. HSSN uses topological, distance, and directional relations to describe the spatial ties that exist between individual `mssn:Locations`. We integrate the aforementioned location-related concepts in order to locate sensors, and better understand the spatial constraints/setup of the `hssn:Infrastructure`.

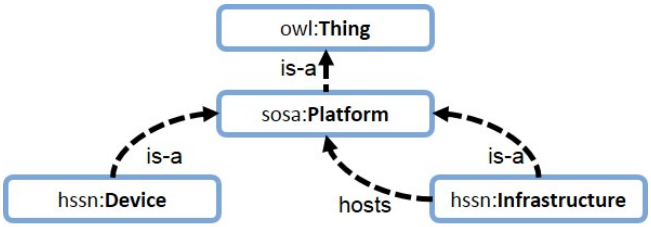


FIGURE 2.12: Platform Representation

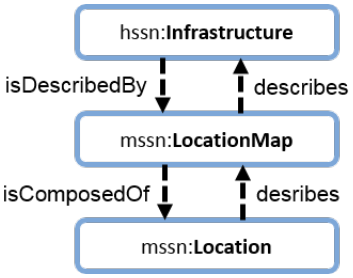


FIGURE 2.13: Infrastructures

#### 2.4.2.2 Device Representation

A `hssn:Device` is another type of `sosa:Platform` where sensors are deployed. It is introduced in HSSN to represent mobile phones and other sensing equipment. A `hssn:Device` has sub-concepts for storage, communication, processing, and power supply, in addition to the ability of embedding sensors (using the `hssn:deployEntity`

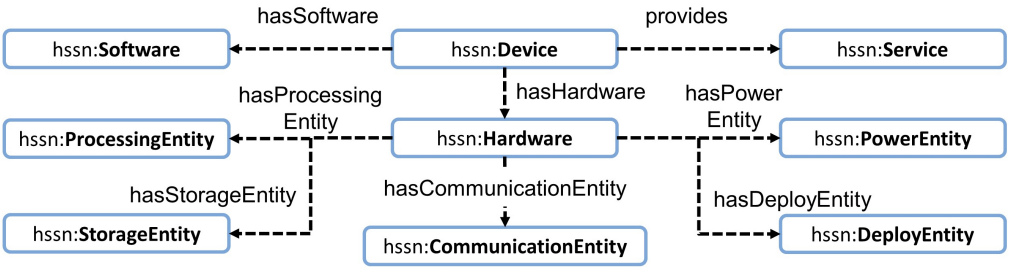


FIGURE 2.14: Device Components

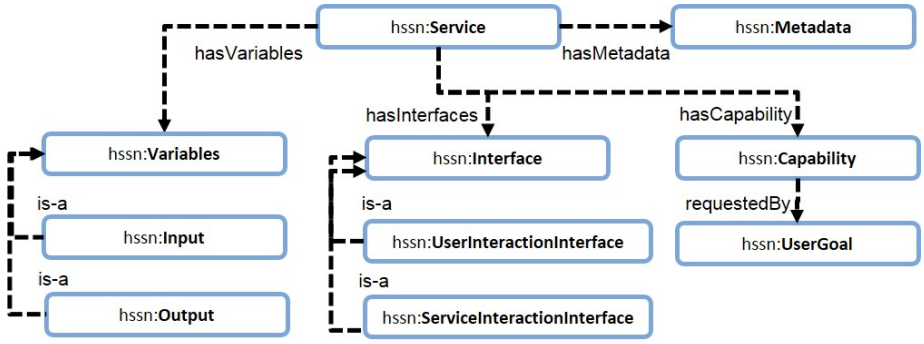


FIGURE 2.15: Service Components

concept cf. Figure 2.14). These concepts describe the `hssn:Hardware` of a device (represented by the `hssn:Device` concept). The `hssn:Software` is also represented. A `hssn:Device` could be used for various purposes (e.g., representing mobile phones for mobile phone sensing, machines with mounted sensors for fault detection in an Industry 4.0 scenario). The hardware and software representation allows complex queries such as assigning sensing tasks to devices based on their processing capabilities, or battery status (cf. Challenge 2 in Section 2.2). Finally, each `hssn:Device` can provide a set of services. Figure 2.15 illustrates our service modeling, inspired by the Web Service Modeling Ontology (WSMO) [82]. We created generic concepts that can be aligned with WSMO. We do not aim to detail the service description to allow alignments with any other service ontology. We limit the service modeling to the following concepts: Service `hssn:Metadata` describes the properties of a `hssn:Service`. The `hssn:Input` represents the set of variables and constraints required for correct service execution, while the `hssn:Output` is the set of generated results. The functionality of a `hssn:Service` is described by the `hssn:Capability` concept which is mapped to a specific `hssn:UserGoal` or objective (i.e., a user desire satisfied by the service). Users communicate with a service through `hssn:UserInteractionInterfaces` (similar to the idea of choreography in **WSMO**). Finally, services communicate with each other via the `hssn:ServiceInteractionInterface` (similar to service orchestration in WSMO). Finally, the infrastructure and device detailing also improves sensor diversity by allowing the representation of simple sensor nodes in infrastructures, multi-sensor systems, and multi-sensor devices.

### 2.4.3 Data Diversity

Audio, image, and video data can be sensed by mobile or static sensors (e.g., surveillance cameras, mobile phones). Also, in order to detect complex events (e.g., gunshot) a combination of multimedia and scalar observations is needed. Therefore, we aim to integrate concepts related to multimedia properties (cf. Criterion 3 in Section

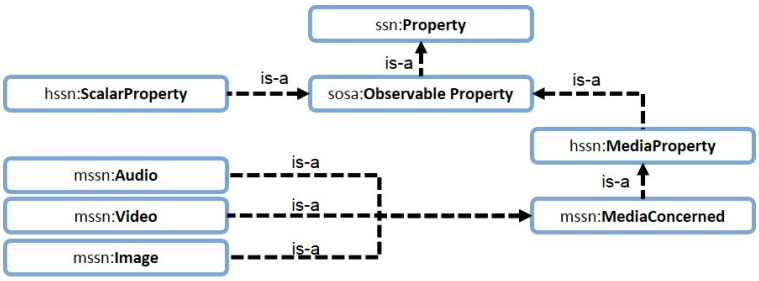


FIGURE 2.16: Observable Properties

2.3). In MSSN [6], multimedia data/properties are integrated in SSN. We distinguish MSSN scalar (e.g., temperature, motion) from multimedia (e.g., noise, video) concepts. Moreover, we group them into two categories as illustrated in Figure 2.16. Also, we introduce in Figure 2.17 the `hssn:mediaSenses` and `hssn:scalarSenses` relationships to map sensors to their corresponding scalar and/or multimedia observable properties (cf. Challenge 4 in Section 2.2). This highlights the sensor diversity in HSSN since static/mobile sensors can detect scalar and/or multimedia properties. The authors in [6] also describe technical aspects/metadata of multimedia objects such as annotations, audio (e.g., frequencies), motion (e.g., trajectories), visual (e.g., color histograms). However, these concepts are not clearly mapped in MSSN. We use and organize these concepts in HSSN to describe sensor observation values in the following way. A `hssn:MediaValue` is composed of the `mssn:MultimediaData` concept, referring to the audio, video, or image objects/files and the `mssn:MediaDescriptor` concepts, describing the metadata of the multimedia objects (e.g., frequencies, colors). *hssn:ScalarValues* are textual (e.g., temperatures, humidity levels). Finally, we map observation values to their related properties using the `hssn:hasMediaValue` and `hssn:hasScalarValue` relationships. Sensors can now be correctly mapped to observable properties and observation values (cf. Challenge 4 in Section 2.2) and each observation can also be linked to its corresponding metadata.

In conclusion, new concepts and properties are introduced in HSSN in order to address the challenges presented in Section 2.2. Our proposal details the representation of infrastructures (a type of platforms) by adding location maps, individual locations, and spatial relations. This allows to expressively describe locations (cf. Challenge 1). In HSSN we describe devices as platforms that host sensors. We detail device hardware, software, and provided services. In addition, we add properties that help locate, track, and query these devices (cf. Challenge 2). HSSN also provides a description of sensor coverage areas and properties that map both locations and coverage areas to mobile/static sensors at any time (cf. Challenge 3). Finally, we address data heterogeneity by detailing multimedia data objects, their metadata, and scalar data. We also map them to their respective sensors (cf. Challenge 4).

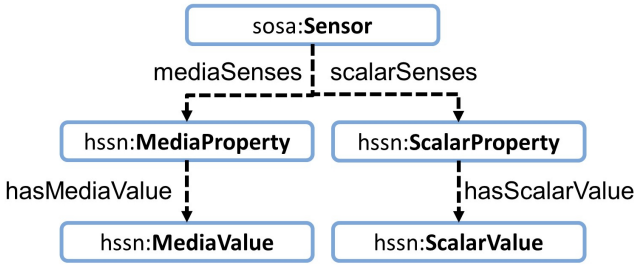


FIGURE 2.17: Sensors/Properties

## 2.5 Implementation & Illustration Example

### 2.5.1 HSSN Implementation

We implemented the HSSN ontology using Protege 5.2.0[4]. Appendix A presents implementation details regarding the added concepts (Table A.3), the added object properties (Table A.2), and the contributing ontologies (Table A.1). We focus here on some interesting HSSN assertions using Description Logic language (cf. Table 2.2). These assertions formalize key constraints in the ontology. For instance, assertion 3 highlights the fact that a sensor can cover different areas but can only be located in one location at any point in time. Assertions 11 and 12 show that Infrastructures can host other platforms but devices cannot (devices can only host sensors, and other hardware/software). Finally, assertions 13-17 distinguish multimedia properties from scalar properties. Further details regarding each concept and property can be found on the HSSN documentation web page[5]. Also, the ontology files are available online[6] for download.

TABLE 2.2: Interesting HSSN Assertions

| # | Assertion |
|---|---|
| 1 | $MobileSensor \sqcap StaticSensor \sqsubseteq \bot$ |
| 2 | $MobileSensor \sqcup StaticSensor \sqsubseteq Sensor$ |
| 3 | $Sensor \sqsubseteq (=1\ isCurrentlyLocatedAt.Location) \sqcap (\exists\ currentlyCovers.CoverageArea)$ |
| 4 | $CoverageArea \sqsubseteq (\exists\ includes.Location) \sqcap (\exists\ isCurrentlyCoveredBy.Sensor) \sqcap (isPreviouslyCoveredBy.MobileSensor) \sqcap (=1\ isCenteredAround.SensingLocation) \sqcap (=1\ isDelimitedBy.SensingRange) \sqcap (=1\ hasHorizontalSpreadDefinedBy.HorizontalSensingAngle) \sqcap (=1\ hasVerticalSpreadDefinedBy.VerticalSensingAngle) \sqcap (\exists\ hasCoverageTime.TemporalEntity)$ |
| 5 | $Location \equiv SensingLocation$ |
| 6 | $Range \equiv SensingRange$ |
| 7 | $HorizontalAngle \equiv HorizontalSensingAngle$ |
| 9 | $VerticalAngle \equiv VerticalSensingAngle$ |
| 10 | $Infrastructure \sqcup Device \sqsubseteq Platform$ |
| 11 | $Infrastructure \sqsubseteq Platform \sqcap hosts.Platform$ |
| 12 | $Device \sqsubseteq Platform \sqcap (\neg\ hosts.Platform)$ |
| 13 | $MultimediaProperty \sqcup ScalarProperty \sqsubseteq ObservableProperty$ |
| 14 | $MultimediaProperty \sqsubseteq (hasMultimediaValue.MultimediaValue) \sqcap (isMultimediaSensedBy.Sensor)$ |
| 15 | $MediaConcerned \sqsubseteq MultimediaProperty$ |
| 16 | $Audio \sqcup Video \sqcup Image \sqsubseteq MediaConcerned$ |
| 17 | $ScalarProperty \sqsubseteq (hasScalarValue.ScalarValue) \sqcap (isScalarSensedBy.Sensor)$ |

[4] https://protege.stanford.edu/
[5] http://spider.sigappfr.org/HSSNdoc/index-en.html
[6] http://spider.sigappfr.org/research-projects/hybrid-ssn-ontology/ (External Links)

In the following, we detail the SPARQL queries used during the experimentation. Then, we describe the experimental setup, before discussing the obtained results from an accuracy, clarity, performance, and consistency standpoint.

### 2.5.2 Illustration Example

The challenges mentioned in Section 2.2 can be addressed by answering SPARQL queries related to platforms, sensors, and data in HSSN.

**Platform Diversity.** In order to expressively describe locations (Challenge 1) in the mall infrastructure, a detailed representation of location maps and locations is needed (`Query 1`). Also, covered and uncovered areas should be easily found (`Query 2`). In order to consider ad-hoc devices in the network (Challenge 2), one should be able to query devices, their hardware (e.g., embedded sensors), software, and services. `Query 3` shows how to locate a mobile device by querying its embedded sensor. Similarly, one could query a device based on other characteristics (e.g., battery status, processing power).

**Query 1: Knowing the spatial description of infrastructures**

```
SELECT distinct ?infrastructure ?locationmap ?location
    WHERE
    {
        ?infrastructure isDescribedBy ?locationmap.
        ?locationmap isComposedOf ?location.
    }
```

**Query 2: Knowing covered locations**

```
SELECT distinct ?location ?coveragearea
    WHERE
    {
        ?location isIncludedIn ?coveragearea.
    }
```

**Query 3: Locating mobile devices**

```
SELECT distinct ?location ?dev
    WHERE
    {
        ?location currentlyLocates ?sensor.
        ?sensor isEmbeddedOn ?du.
        ?du hasExpansionCard ?hd.
        ?hd isRelatedToDevice ?dev.
    }
```

**Sensor Diversity.** Knowing where each sensor is currently located is important for various reasons (e.g., assigning tasks to specific sensors, monitoring certain areas). Also, one might need to track one or more sensors at all times (Challenge 3).

Therefore, it is pivotal to know the current locations for all sensors, as well as previous ones. Similarly, the sensors' coverage areas should be easily retrievable (e.g., in order to discover unmonitored areas in the environment). The following queries retrieve the current (`Query 4`), and previous (`Query 5`) locations/coverage areas of the available sensors.

**Query 4: Finding current sensor locations/coverage areas**

```
SELECT distinct ?location ?sensor ?coveragearea
    WHERE
    {
        ?location currentlyLocates ?sensor.
        ?sensor currentlyCovers ?coveragearea.
    }
```

**Query 5: Finding previous sensor locations**

```
SELECT distinct ?location ?sensor
    WHERE
    {
        ?location hasPreviouslyLocated ?sensor
    }
```

**Data Diversity.** In order to consider data diversity (Challenge 4), on should be able to distinguish scalar/multimedia data and correctly map them to sensors. `Query 6` selects all scalar properties that are observable by any sensor. In addition, the query retrieves all made observations regarding each scalar property. Similarly, `Query 7` has the same functionality but targets multimedia instead of scalar data.

**Query 6: Mapping sensors to their scalar properties and observations**

```
SELECT distinct ?sensor ?property ?observation
    WHERE
    {
        ?sensor scalarSenses ?property.
        ?property isScalarValueOf ?observation.
    }
```

**Query 7: Mapping sensors to their multimedia properties and observations**

```
SELECT distinct ?sensor ?property ?observation
    WHERE
    {
        ?sensor mediaSenses ?property.
        ?property isMediaValueOf ?observation.
    }
```

## 2.6 HSSN Experimental Setup

Here, we did not aim to experiment SOSA/SSN [44] concepts and properties. We evaluated the impact of our newly added HSSN concepts and properties (e.g., concerning static/mobile sensors, infrastructures/devices, multimedia/scalar data).
Our objectives were the following:

1. *Accuracy Evaluation:* Checks if the added concepts and properties answer the challenges mentioned in Section 2.2. This is a query-based evaluation that highlights the impact of our extensions and their contribution towards overcoming the aforementioned challenges.

2. *Clarity Evaluation:* Checks if the labels used to describe the added concepts and properties are clear and unambiguous to domain stakeholders. The aim is to evaluate the compatibility and clarity of our provided description with respect to the sensor network domain.

3. *Performance Evaluation:* Measures the impact of our additions on performance (i.e., query run time). The aim is to evaluate the feasibility, performance-wise, of integrating HSSN in sensor network applications.

4. *Consistency Evaluation:* Checks if the added concepts and properties generate inconsistencies (e.g., anti-patterns) within the structure of the ontology. The aim is to evaluate the soundness of the ontology graph.

### 2.6.1 Accuracy Evaluation

We created a population of individuals and ran the aforementioned queries (described in the illustration example). Then, we compared the obtained and expected results. We created two infrastructures, each described by a location map containing 500 locations. Then, 1000 sensors were deployed (500 mobile, 500 static, 500 scalar, 500 media). Each sensor is located in one location, covers one coverage area, observes one property, and produces one observation value.

#### 2.6.1.1 Platform Results

We ran queries 1, 2, and 3. The returned results match perfectly the expected ones. Infrastructures were correctly assigned to their location maps and included locations. This allowed the identification of distinct spaces/areas. `Query 2` correctly returned the set of distinct locations included in each coverage area. This allowed the identification of non covered locations. `Query 3` allowed the identification of device hardware related to the embedded sensors. Also, the mobile devices were correctly located in the location map.

#### 2.6.1.2 Mobility Results

We ran queries 4 and 5 on the population of individuals and for each case the returned results matched exactly the expected ones. Sensors were correctly assigned to their current/previous locations and coverage areas.

#### 2.6.1.3 Data Results

We ran queries 6 and 7 and obtained an exact matching between the actual and expected results. Thus, scalar/multimedia properties were correctly distinguished. Also, sensors were correctly assigned to the scalar or multimedia observations that they produced.

**Discussion.** The test results showed that locating any type of sensor (i.e., simple node/multi-sensor device, static/mobile sensors, and scalar/multimedia sensors), and knowing their coverage areas is possible at any point in time. Hence, allowing tasks such as tracking mobile sensors, and detecting uncovered areas. Also, the results showed that the detailing of infrastructure and device descriptions (platform diversity) allowed a better knowledge of the environment space (also important for locating sensors). Multi-sensor devices were also detailed by describing their hardware and software which proved useful when querying devices based on their capabilities (e.g., we ran an additional query that returns sensors/devices with good battery status). From a data diversity standpoint, the results showed that sensors that sense multimedia/scalar properties were correctly distinguished and their observations were accurately retrieved. To conclude, the query results confirmed that the added extensions (i.e., regarding sensor, platform, and data diversity) accurately answer the challenges mentioned in Section 2.2.

### 2.6.2 Clarity Evaluation

We created two evaluation forms: the first[7] for evaluating the ambiguity of the labels used to describe the HSSN concepts, and the second[8] for evaluating the ambiguity of the labels used to describe inter-concept relations (i.e., the object properties). We sent the two forms to 50 sensor network and ontology experts (25 networking experts, and 25 computer scientists).

#### 2.6.2.1 Clarity Results

Results in Figure 2.18 and 2.19 show that terms considered clear by computer scientists are sometimes found ambiguous by network experts and vice-versa. Figure 2.18 shows that a few terms do not meet the acceptable ambiguity level (e.g., ComUnit, DeployUnit), while others (e.g., MediaProperty, MediaValue) need some clarification. Therefore, we considered the experts' suggestions in the final version of the ontology and made the adjustments described in Table 2.3. Moreover, we added synonyms for some labels to increase the clarity by considering nomenclatures from different domains. We also adjusted several property labels in order to be coherent with the new concept labels. Finally, Figure 2.19 shows that in most cases, both categories of experts assigned correctly the inter-concept relationships. Networking experts have low success on the first two questions since the latter are outside of their domain of expertise (regarding inheritance between concepts).

**Discussion.** The clarity evaluation allowed the identification and correction of ambiguous/unclear labels that we used to describe our added concepts/properties. In the version currently available online, all labels achieve an acceptable level of clarity

[7]Link: https://goo.gl/forms/blc8pKLLqtNtjXHI2
[8]Link: https://goo.gl/forms/KNNY3XsmGp0ptM2N2

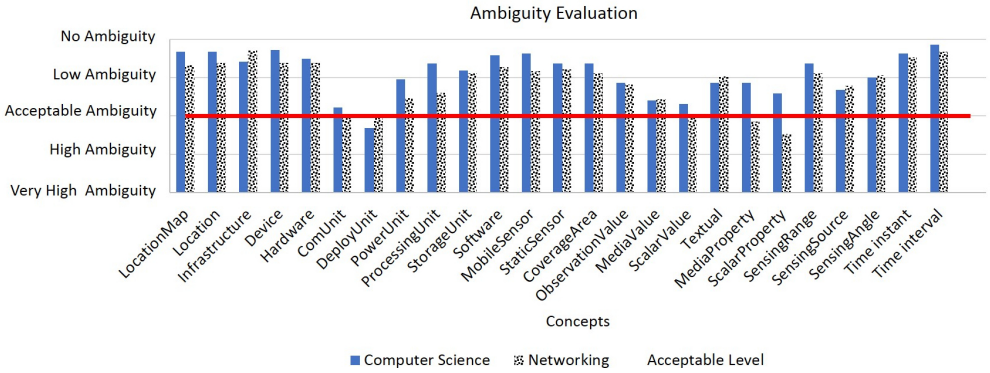


FIGURE 2.18: Concept Evaluation

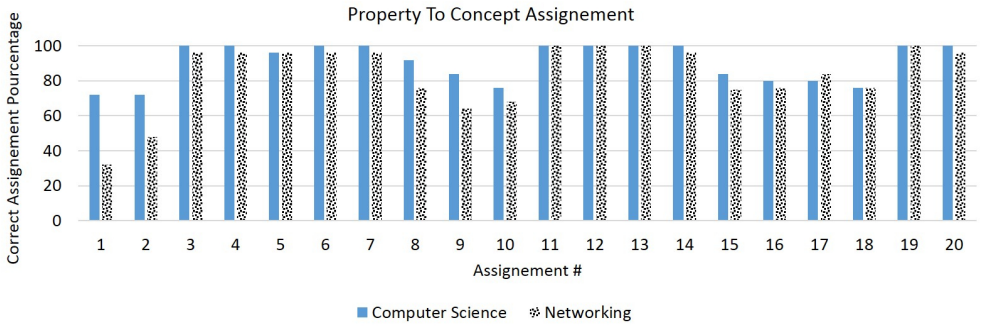


FIGURE 2.19: Property Evaluation

TABLE 2.3: HSSN Concept Label Modifications

| New Label | Initial Label |
|---|---|
| ExpansionCard | DeployUnit |
| PowerSupply | PowerUnit |
| NetworkInterface | ComUnit |
| Memory | StorageUnit |
| Processor | ProcessingUnit |
| Multimedia | Media |

(based on the stakeholders' feedback). This reinforces the re-usability of HSSN since it is unambiguous and easily understood.

### 2.6.3 Performance Evaluation

In order to evaluate the performance of HSSN, we measured the query run-time by running each of the previously mentioned queries 10 times and calculating the average execution time. We varied the size of the population (100 sensors, 1000 sensors, and 10000 sensors) in order to test various scenarios related to mobility, platforms, and data. The tests were conducted on a machine equipped with an Intel i7 - 2.6 GHz processor, and 16 GB of RAM.

#### 2.6.3.1 Mobility Impact

In this test, we varied the percentage of mobile sensors in the network (0, 30, 50, 70, and 100 %). Then, we retrieved the current/previous sensor locations (cf. Figure 2.20 and 2.21). We measured the run-time for queries 4 and 5. In Figure 2.20, we noticed that increasing the number of mobile devices increases the time required to

retrieve current sensor locations. This is due to the fact that locating a device (`Query 3`) was a more complex task than locating a static sensor since we needed to locate the sensor, the card where it is deployed, the related hardware, and then the device that has this specific hardware. We noticed the same pattern for all three cases (100, 1000, 10000 sensors). Finally, the progression from 0% to 100% mobile devices had a quasi-linear impact on query run-time. Similarly, Figure 2.21 details the query run-time for retrieving previous different sensor locations. Since mobile sensors have a larger list of previous locations in comparison with static sensors, increasing the mobility percentage (0, 50, 100 %) increases the query run-time. This progression was also quasi-linear for all three cases (100, 1000, 10000 sensors).

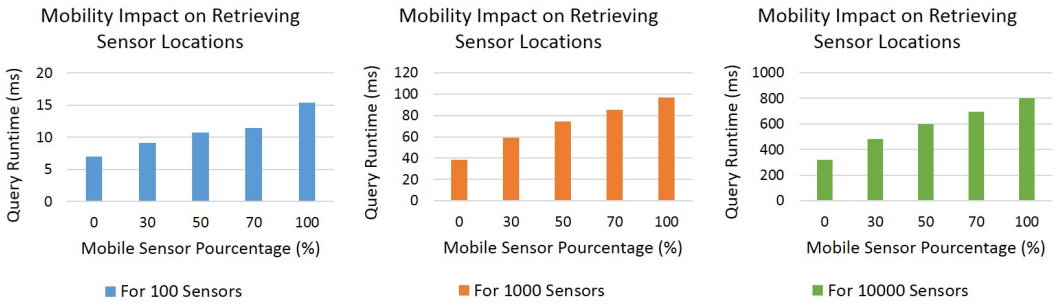


FIGURE 2.20: Mobility impact on current location retrieval

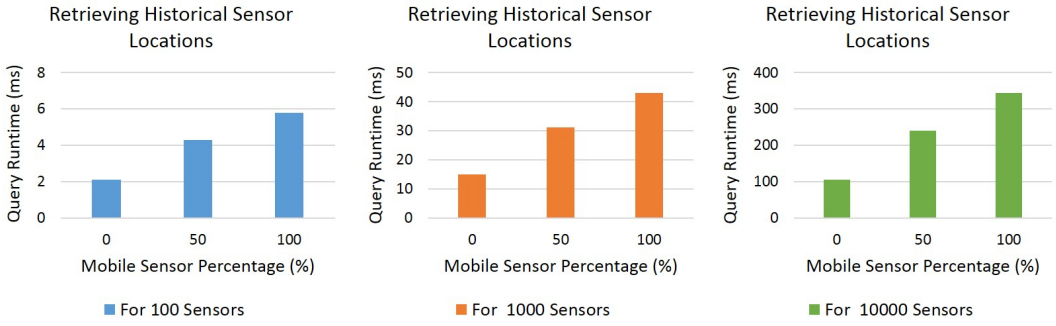


FIGURE 2.21: Mobility impact on previous location retrieval

### 2.6.3.2 Platform Impact

In this test, we varied the sensor distribution on the platform locations. We tested three different scenarios (i) each sensor is located in one location; (ii) all sensors are located in one location; and (iii) half of the sensors are located in a location and the other half in another. We measured the run-time of the query that retrieves sensor locations. Figure 2.22 shows how sensor distribution on locations affected the time needed to map sensors to their current locations. When all sensors were located in one location, the required time to perform this task was minimal. Then, as we began to decrease sensor densities, the query took more time. Finally, the worst case was when every location contained only one sensor.

### 2.6.3.3 Data Impact

Here, we checked the impact of scalar/multimedia data on the run-time of queries 6 and 7 (cf. Figure 2.23). For data diversity impact on performance (cf. Figure 2.23),

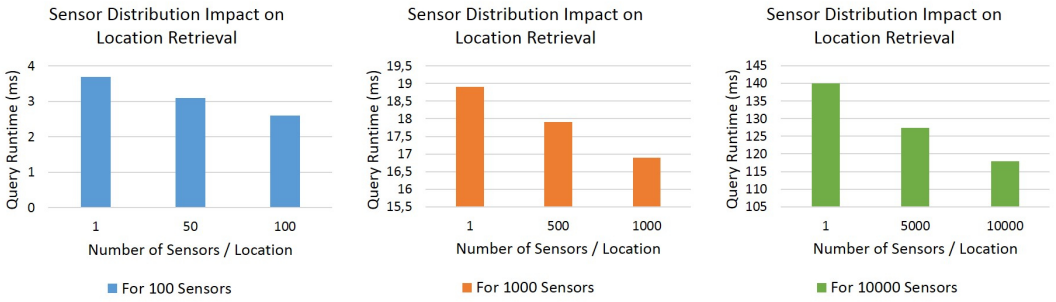


FIGURE 2.22: Platform impact on current location retrieval

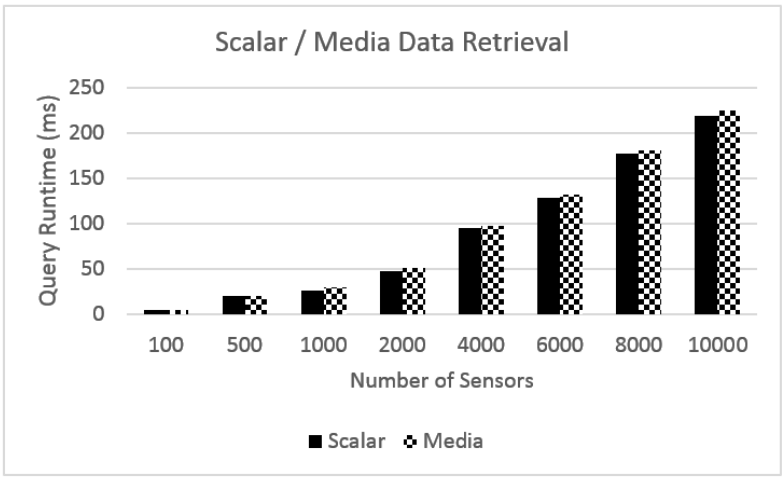


FIGURE 2.23: Data impact on observation retrieval

we noticed that the evolution of the graph is slightly exponential (which was not the case for the previously discussed performance results). This is due to the fact that sensor observations have complex structures that hold not only the observation values but also a set of metadata describing each sensed data. Moreover, in all cases (100, 1000, 10000 sensors) the query run-time was similar when considering scalar and multimedia data. This is due to the fact that we were measuring the time required to retrieve the data and not the time needed to capture/sense it. Multimedia observations (data objects and metadata) require more time to retrieve than scalar observations (textual data and metadata).

**Discussion.** The performance evaluation showed that the added concepts and properties do not heavily impact the query run time, which remains quasi-linear in most cases. This highlights the feasibility of using of HSSN in sensor applications (from a performance point of view).

### 2.6.4 Consistency Evaluation

In [93], consistency is defined as a criterion that verifies if the ontology allows contradictions. The descriptions in the ontology should be consistent.

**Consistency Results.** To evaluate consistency, we adopted the following SPARQL queries that search for anti-patterns, a strong indicator of inconsistencies, in the ontology. `Query 8` detects concepts with no parent, and `Query 9` detects abnormally

disjointed concepts in the ontology. Finally, to conclude the inconsistency evaluation, we ran Protege's HermiT 1.3.8.413 reasoner, and found no inconsistencies between the asserted class hierarchy and inferred one.

**Query 8: Searching for concepts with no parent**

```
SELECT ?a WHERE
{
    ?a subClassOf owl:Nothing.
}
```

**Query 9: Searching for abnormally disjointed concepts**

```
SELECT distinct ?A ?B1 ?B2 ?C1
    WHERE
    {
        ?B1 subClassOf ?A.
        ?B2 subClassOf ?A.
        ?C1 subClassOf ?B1.
        ?C1 disjointWith ?B2.
    }
```

**Discussion.** We found no inconsistencies in the HSSN ontology structure. The only concept subsuming nothing is owl:Nothing (`Query 8`). `Query 9` results indicate that there are no concepts that have abnormal disjoint relations with their relatives. This denotes the soundness of the integration of newly added concepts mainly with the SOSA/SSN core. This highlights the soundness of the graph structure, which proves critical when considering future alignments between HSSN and other ontologies (e.g., that describe smart buildings, events).

## 2.7 Summary

Many works adopted ontologies for better semantic representation of sensor networks. These approaches do not fully consider diversity in terms of sensors, data, platforms. Moreover, some works contain domain specific knowledge and are not re-usable for application purposes. In this chapter, we propose an extension of the Semantic Sensor Network ontology (SSN) [44], since it is already re-usable in various contexts. Our proposed ontology, denoted HSSN [44], adds to SOSA/SSN sensor mobility, and multimedia data related concepts in order to have a representation of hybrid sensor networks. HSSN also extends the platform representation of SOSA in order to fully consider platform diversity. We implemented HSSN, evaluated the consistency, accuracy of our additions, and their impact on performance.

# Chapter 3

# An Event Query Language For Connected Environments

*"The chief virtue that language can have is clearness, and nothing detracts from it so much as the use of unfamiliar words."*

*— Hippocrates*

Nowadays, connected environments impact various application domains (e.g., energy management, environment monitoring) by offering users a wide array of applications that help them in their every day lives (e.g., reducing energy wastes in buildings, monitoring air or noise pollution levels in a city). Although these applications seem different, they all rely on an environment, its sensor network, and sensed data in order to detect and handle specific events (e.g., energy waste, traffic congestion, high level of noise, bad air quality). The differences lie in the definition of the targeted events (e.g., high noise different from traffic congestion event), the application domain (e.g., environmental, energy), the sensors/data required for the detection of the events, and the chosen technique for event detection.

Event Query Languages (EQL) have been proposed in connected environments to allow users the definition of targeted events. However, existing languages are limited to the definition of event patterns and suffer from the following drawbacks: (i) no consideration of environment, sensor network, and application domain related components; (ii) lack of provided query types (functionality) required for the definition/management of the entire connected environment; (iii) lack of considered data and datatypes (e.g., scalar, multimedia) needed for the definition of specific events; (iv) lack of considered functionality when expressing spatial/temporal constraints; and (v) difficulty in coping with the dynamicity of the environments.

To address the aforementioned limitations, we propose here an EQL specifically designed for connected environments, denoted EQL-CE. We detail its framework, the used language, syntax, and queries. EQL-CE is re-usable and generic. It allows the definition of various connected environment components, offers various query types for data management, and considers various datatypes. We also introduce a query optimizer that handles the dynamicity of the environment and spatial/temporal constraints. We finally illustrate the EQL and conclude the chapter.

## 3.1 Introduction

Recent advances in the fields of Information & Communication Technologies (ICT), Big Data, Sensing Technologies, and the Internet of Things (IoT) have paved the way for the rise of smart connected environments. These environments are defined as infrastructures that host a network of sensors capable of providing data that can be later mined and processed using advanced techniques, for high level applications. Hence, Sensor Networks (SN) are currently impacting numerous domains (e.g., medical, industrial, environmental, cities, buildings). This allowed a plethora of sensor-based applications such as monitoring a patient's health [104], improving manufacturing processes in smart factories [63, 102], detecting fires in the wilderness [110], monitoring pollution levels or helping drivers avoid traffic congestion in a city [60], and reducing the energy footprint or optimizing occupants' comfort in buildings [1, 22, 59, 101, 108, 111].

The aforementioned applications have different objectives. However, in order to achieve their goals, they all need to define and detect specific key events while relying on the sensed data (e.g., abnormal heart rates from wearable sensors, machinery faults from sensors mounted on machines, fires from fire sensors, bad air quality from $CO_2$ sensors, traffic congestion from GPS enabled sensors mounted on vehicles, energy wastes or room overheating from indoor sensors deployed in buildings). Therefore, these applications share the following needs: (i) representing the infrastructure and the sensor network of the connected environment; (ii) defining and detecting the targeted events; and (iii) protecting the security of the sensed data and the privacy of the users in the environment (e.g., protecting patients' medical records). In the aforementioned works, the authors do not emphasize on the environment's representation and define events statically. They also propose event detection mechanisms that perfectly fit the description of the targeted events. This is constraining since these works are not re-usable in different contexts.

To overcome this issue, Event Query Languages (EQL) have been proposed in many works [5, 88] as a means for event definition prior to detection. EQL allow users to express how the targeted events are defined (i.e., event describing features, patterns). However, existing languages [7, 8, 14, 21, 26, 35, 42] focus mainly on the event descriptions and do not consider other environment components (e.g., infrastructure, sensor network, application domain). They share the following limitations:

- *Lack of considered components:* existing works heavily focus on events. It is important that the EQL allows the definition/management of the entire connected environment. This includes components related to the environment itself, its sensor network, the targeted events, and the application domain. For instance, one might need to manage the infrastructure (e.g., locations, spatial ties), the sensor network (e.g., sensors, observations, properties), and additional descriptions related to the application domain (e.g., industrial, environmental).
- *Lack of considered functionality:* existing languages offer few query type, i.e., mainly queries that define event patterns. It is important that the EQL (i) allows the definition of components (e.g., buildings, sensors, data, events); (ii) allows the manipulation of component instances (e.g., inserting new instances, updating, deleting, selecting them); and (iii) protects the security/privacy of data/users.

- *Lack of considered datatypes:* some events (e.g., intrusion in a building) require a combination of scalar (e.g., motion) and multimedia (video, noise) data. It is important to integrate the diverse data and datatypes needed for the definition of such specific events (e.g., for scalar, multimedia sensor observations).

- *Lack of spatial/temporal distribution functions:* since the sensors' locations impact event detection, the EQL should allow users to define spatial distributions of the sensors over the environment in order to better detect the targeted events. This entails specifying where each sensor should be located or how they should be distributed over the space (e.g., nearest sensors to a point of interest, sensors within a range of a point of interest, sensors that fit a mathematical distribution around a point of interest). Similarly, since sensors provide observations at specific rates, one could end up with either: (i) big volumes of unnecessary data (if the rate is too quick); or (ii) undetected events (if the rate is too slow). Therefore, it is important that the EQL allows the adjustment of the temporal distribution of sensor observations based on events' needs/requirements. This entails specifying which sensor observations/sensing rates are considered for a specific event, or selecting a temporal distribution of these observations (e.g., the closest observations to a certain point in time, all observations within a temporal range, distributed sensing rates).

- *Difficulty in coping with the dynamicity of the environment:* in a dynamic environment, sensors might breakdown, mobile sensors could enter/leave the network or change locations at any time. Since events rely on sensors and their observations, event queries need to cope with such changes. The EQL should be able to keep track of environment changes/states, in order to address obsolete queries. This entails replacing missing sensors by others capable of providing the required data or replacing missing observations with others that fit the event definition in the queries.

- *Lack of re-usability:* some works rely on data model based syntax (e.g., SQL-based, SPARQL-based languages). It is important that the EQL remains generic and independent from any technological constraints or underlying infrastructure to ensure re-usability in different contexts.

Many other challenges emerge when considering an EQL for connected environments (e.g., handling big volumes of data, continuous heterogeneous data streams). However, in this work, we focus mainly on the aforementioned limitations. Hence, we propose here an EQL specifically designed for connected environments and partitioned into three layers: (i) the conceptual layer where one could represent the connected environment in the form of an entity/relation graph; (ii) the logical layer where high level generic queries are composed using the Extended Backus-Naur Form (EBNF) syntax; and (iii) the physical layer where the queries will be parsed and executed. Our proposal, denoted EQL-CE, covers various connected environment elements (i.e., related to the environment, sensor network, events, and application domain). It also provides common query types for the definition of components and the management of their instances. and integrates various datatypes. Finally, we propose a query optimizer module that will handle spatial/temporal distributions and query re-writing in order to redefine components that need to evolve when handling the environment's dynamicity (the optimizer will be fully detailed in the next chapter).

The remainder of this chapter is organized as follows. Section 3.2 presents a scenario that motivates our proposal. Section 3.3 evaluates existing approaches. EBNF preliminaries and background are presented in Section 3.4. Section 3.5 presents the EQL-CE framework and details its syntax. An illustration example and experimental protocol are presented in Section 3.6. Finally, Section 3.7 concludes this chapter and discusses future research directions.

## 3.2 Motivating Scenario

In order to motivate our proposal, consider the following scenario that illustrates a smart mall. This is a simplified example that illustrates the setup, the needs, and motivations behind our proposal. Of course, it does not summarize all needs found in a connected environment/event detection application scenario. Figure 3.1 details the infrastructure's location map, and individual locations (i.e., shops and open areas). The mall is equipped with a hybrid sensor network having static/mobile sensors, single sensor nodes/multi-sensor devices capable of monitoring the environment and producing scalar/multimedia observations (e.g., temperature, video). A manager uses an Event Query Language (EQL) in order to define/detect interesting events that occur within the mall's premises. Although this seems enough to manage the smart mall, many improvements can still be integrated:

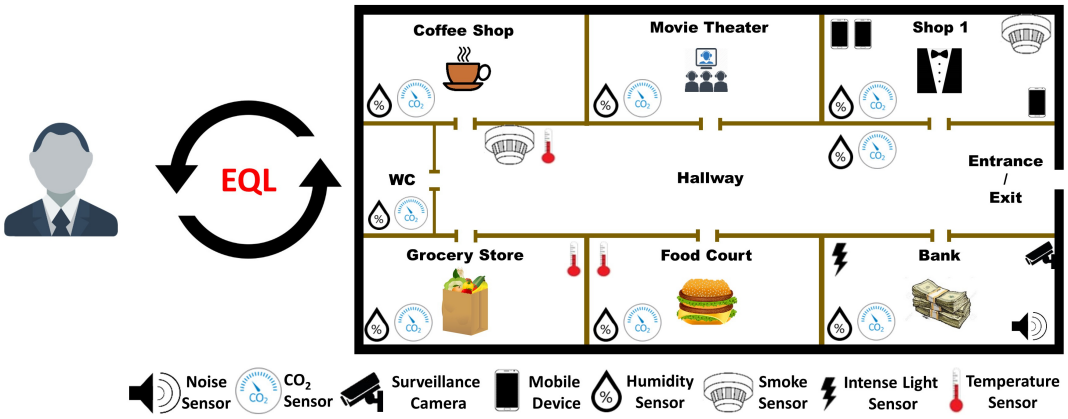


FIGURE 3.1: The Smart Mall

**Need 1.** Modeling the environment and its sensor network. Before defining and detecting events, a mall manager needs to represent the smart mall using the EQL. This includes defining the infrastructure (i.e., the mall), the locations (e.g., shops), and their spatial relations. Then, the manager needs to define the sensor network that is hosted in the mall. This entails modeling the available sensors (e.g., temperature, humidity), their deployment locations, the data they sense and so on. Once all component structures are defined, the mall manager needs to use the EQL to create instances of each component (e.g., temperature sensor in food court). This is currently not possible since the EQL used in the example only defines events.

**Need 2.** Querying the connected environment. In addition to event definition and detection, the mall manager might need to query the connected environment for other data management purposes (e.g., retrieving specific sensor readings, locating mobile devices). The EQL should provide common/basic querying functionality. This entails queries for defining the connected environment components

(e.g., environment, sensor network, events, and application domain). Moreover, one might need other query types for data manipulation (e.g., selecting sensors, retrieving sensor observations, detecting events).

**Need 3.** Handling various datatypes. This entails covering scalar and multimedia sensor observations (e.g., textual temperature values, video surveillance footage). The mall manager needs to define different events (e.g., intrusion detection, indoor temperature overheating). To do so, the EQL should be able to integrate different formats of data and their respective datatypes.

**Need 4.** Measuring the average temperature in the grocery store (for food storage concerns). The mall manager uses the existing EQL to define the targeted event (i.e., the average temperature in the grocery store). The EQL allows the manager to consider all sensors within the area of interest. However, Figure 3.2 shows that the sensors are not evenly distributed in the store (most are located in the upper left corner). Hence, considering all sensors and calculating the average temperature will produce a biased measure that does not reflect the reality of the situation. This can be solved by allowing the manager to define a specific distribution of sensors over the space (e.g., even distribution, only considering sensors within a range of the center of the store). The current setup is limited since it does not allow the definition of spatial distributions of sensors.

**Need 5.** Minimize data overload/missed events. Currently, the manager can use the EQL to define one sensing rate for all sensors or sensor types (e.g., temperature, humidity). This is constraining since (i) a quick sensing rate overloads the system with big volumes of unwanted/unnecessary data; and (ii) a slow sensing rate could lead to missing events that began and ended in a short time lapse. Therefore, the temporal distribution of sensor observations (i.e., a start time, a specific rate, a stop time) should be based on the event definition and therefore considered/handled in the event queries (e.g., selecting the closest observations to a time of interest, considering different sensing rates from various sensors at once). The EQL used by the mall manager does not allow such customization of temporal constraints (cf. Figure 3.3).

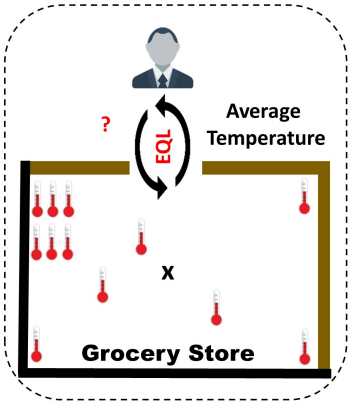


FIGURE 3.2: Spatial Distribution

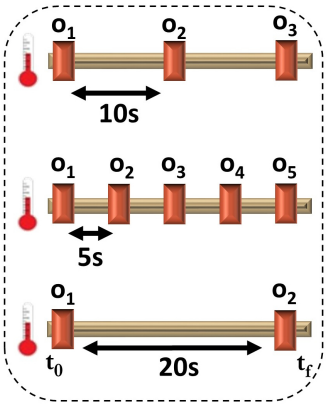


FIGURE 3.3: Temporal Distribution

**Need 6.** Detecting a fire event in Shop 1. The mall manager defines this fire event using the EQL. His/Her definition relies on the smoke, humidity, and $CO_2$ sensors located in Shop 1. However, what if the smoke sensor broke down ? Or what

if the mobile device that the query depends on, left the shop ? Then, the previously defined event query will become obsolete since there are no more smoke observations coming from shop 1, and there is no way of changing the event definition. Hence, query re-writing is necessary in order to update the definition: (i) by replacing the smoke sensor by another capable of providing the same data (e.g., mobile device 1 - cf. Figure 3.4 - left); or (ii) by replacing the event describing feature smoke by another (e.g., temperature from mobile device 1 if no other sensors can provide smoke observations - cf. Figure 3.4 - right). The current EQL is limited since it does not allow such re-writing.

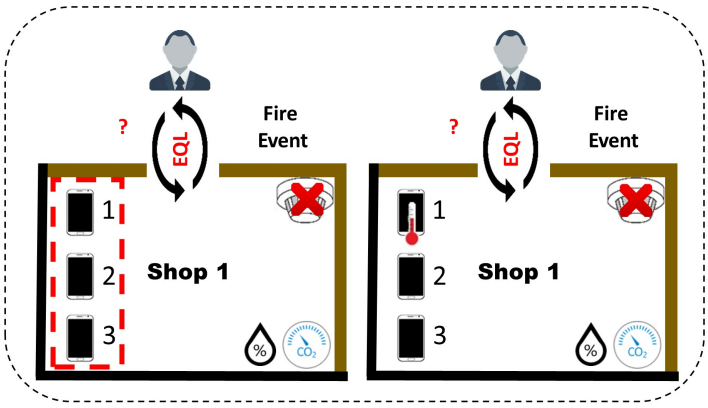


FIGURE 3.4: Need For Query Re-writing

Existing EQL mainly focus on event definition, and do not handle other connected environment components. To address needs 1 and 2, one might use another language that integrates different functionality and handles the environment/sensor network changes. However, in this case, the manager will have to use various languages with different syntax. A more appropriate solution might be to extend the capabilities of the EQL to provide a means for defining the structure of various components related to the environment, sensor network, targeted events, and application domain (cf. Need 1). Moreover, the EQL should not be limited to defining components. Its functionality should extend to managing instances of these components, ensuring basic/common querying tasks, and protecting the security/privacy of the data/users (cf. Need 2). Also, the language should be capable of integrating a variety of data and datatypes that allow the management of both scalar and multimedia data (cf. Need 3). In addition, customizing the sensors' spatial distribution over the infrastructure/environment based on event requirements is required (cf. Need 4). This benefits the event detection since it provides the user with the ability to customize the setup in the way that he/she believes is optimal. The same is also applied for temporal distribution of sensor observations. The EQL should allow the user to select specific observations, or a set of distributed observations in time (e.g., considering different sensing rates, temporal distance to a point in time) when defining the event (cf. Need 5). The EQL should also allow re-writing queries (cf. Need 6) to handle the dynamicity of the connected environment. This is especially beneficial when faults or sensor breakdowns/mobility can render some event definitions obsolete. Finally, this should be done using a generic, technology independent syntax that could be parsed into various data model-based languages to ensure re-usability.

However, when considering various components, functionality, datatypes (e.g., scalar, multimedia), data distributions (e.g., spatial, temporal), and query re-writing the following challenges emerge:

**Challenge 1.** How to model components and inter-component relations? How to establish ties between the different connected environment elements (i.e., environment, sensor network, events, and application domain)?

**Challenge 2.** How to define different query types to cover all the required functionality?

**Challenge 3.** How to define the structure of both scalar and multimedia data?

**Challenge 4.** How to integrate variables that specify spatial/temporal distributions in the query syntax ? How to propose different distribution queries types?

**Challenge 5.** How to detect obsolete event definitions? How to redefine these events by replacing missing sensors/data? How to measure similarities between sensors and data in order to replace missing query elements by others that are similar?

**Challenge 6.** How to establish a generic/re-usable syntax that is independent from the underlying infrastructure?

Therefore, we propose here a high-level generic event query language, denoted EQL-CE, that covers all connected environment components, provides basic, common query types to consider various functionality, and integrates different datatypes. We also propose a query optimizer that handles query re-writing and spatial, temporal distribution functions. In this chapter, we present the framework and syntax of our proposed language. We detail query re-writing in the following chapter.

## 3.3 Related Work

In this section, we review existing works on Event Query Languages. We propose the following criteria based on the challenges and needs discussed in Section 3.2:

**Criterion 1.** *Component Coverage:* This criterion denotes $\{YES, NO\}$ if the EQL is capable of covering the entire connected environment. This includes environment, sensor network, application domain, and event related components (cf. Need 1).

**Criterion 2.** *Basic Querying:* This criterion states $\{YES, NO\}$ if the EQL allows common query types for component definition, and component instance manipulation (cf. Need 2).

**Criterion 3.** *Data Diversity:* This criterion specifies $\{YES, NO\}$ if the EQL is capable of integrating various datatypes related to the scalar/multimedia sensed properties/sensor observations (cf. Need 3).

**Criterion 4.** *Spatial/Temporal Distributions:* This criterion indicates $\{YES, NO\}$ if the EQL allows (i) spatial distribution queries (e.g., selecting sensors that are distributed based on a mathematical law, within a specific range, or near a point of interest); and (ii) temporal distribution queries (e.g., selecting sensor observations that are closest to a point in time that have various sensing rates). This is important for the definition of specific events where such level of detail is required (cf. Needs 4 and 5).

**Criterion 5.** *Handing Environment Dynamicity:* This criterion points out $\{YES, NO\}$ if the EQL provides the means to modify the structure of previously defined components (e.g., events) in order to cope with the environment's dynamicity. This is useful in a dynamic setup, where sensor mobility causes gain/loss of data in certain areas of the environment (cf. Need 6).

**Criterion 6.** *Re-usability:* This criterion designates $\{YES, NO\}$ if the EQL is generic and technology independent in order to re-use it in various setups with different underlying infrastructures (e.g., traditional database, ontology). It is beneficial to have a high level, generic, and declarative EQL that can be parsed into data-model specific languages (instances). This facilitates its integration in various contexts, therefore we consider re-usability as an additional need.

We group the existing works into three main categories: (i) conceptual languages (e.g., Event-Condition-Action languages) ; (ii) logical languages; and physical languages (e.g., SQL/SPARQL-based languages). We compare in the following some works [7, 8, 9, 13, 14, 21, 24, 26, 35, 41, 42, 58, 75] from each category (we do not detail here every existing event query language for the sake of brevity).

### 3.3.1 Conceptual Languages

This category of languages includes Event Condition Action (ECA) languages that allow the declaration of three event attributes: (i) an event name or label; (ii) a set of conditions (i.e., the pattern) that best define the event; and (iii) the set of actions that should be triggered once the event is detected.

**SNOOP.** In [26], the authors propose an intuitive event query language denoted SNOOP. They follow the ECA model when defining event structures. They integrate operators for inter-condition relations (e.g., conjunction, dis-junction, and sequence) and represent repetitive events through the usage of the periodic and aperiodic operators. The syntax and an example are described below. In this example, we define an event, its pattern, and the triggered action once the event is detected.

**SNOOP Syntax**

```
ON (<event_label>)
CONDITION <constraints>
ACTION <action_label>
```

**SNOOP Query**

```
ON (empty_room)
CONDITION no_face_detected
ACTION turn_off_lights
```

**CeDR.** In [14], the authors propose a language denoted CeDR. In comparison with SNOOP, CeDR adds a WHERE clause for filtering statements and has a wider range of operators. Therefore, CeDR is considered more expressive in terms of event pattern description. CeDR also includes an event lifetime operator and a detection window operator. The syntax and an example are illustrated below.

**SaSE.** In [42], the authors propose an event query language for data streams called SaSE. They include the WITHIN and RETURN statements to respectively declare sliding time windows and the required output. SaSE also allows event pattern operators (similar to CeDR) in a WHERE clause. The syntax consists of an event pattern,

**CeDR Syntax**

```
EVENT <event_label>
EVENT_PATTERN <pattern>
WHERE <constraints>
```

**CeDR Query**

```
EVENT empty_room
EVENT_PATTERN face AND vibration
WHERE no_face AND low_vibration
```

a set of conditions for result filtering, a specific time window to detect events based on temporal constraints, and finally the expected result. We also provide an example of an empty room event where no faces are detected during a time window of 30 minutes.

**SaSE Syntax**

```
EVENT <Pattern>
WHERE <Constraints>
WITHIN <Time_Sliding_Window>
RETURN <Output_Data>
```

**SaSE Query**

```
EVENT face
WHERE no_face_detected
WITHIN last_30_min
RETURN empty_room
```

### 3.3.2 Logical Languages

This category of languages includes works that define events in logic style formulas (math-like notations).

**ETALIS.** In [8], the authors propose an EQL that describes events as rules. It provides a set of temporal relations and composition operators to define the event patterns. The syntax of the rules is independent of any underlying data model. The syntax of ETALIS is written in a fortran style function. The following operators are proposed for event pattern definition: SEQ, AND, PAR, OR, DURING, STARTS, EQUALS, FINISHES and MEET. The event pattern is defined as a set of triples (i.e., operand, operator, operand). The following illustrates a fire event as a sequence of three other events.

**ETALIS Syntax**

```
<Event_Name> ← {(CONDITION | EVENT , OPERATOR, CONDITION | EVENT)}
```

**ETALIS Query**

fire ← {(*temperature* > 30°) SEQ ($CO_2$ > 30%) SEQ (smoke)}

**XChangeEQ.** In [21], another logical language is provided. The authors allow the following features in its queries: (i) data-related operations such as variable bindings and conditions containing arithmetic operations; (ii) event composition operators such as conjunction, dis-junction, and order; (iii) temporal and causal relations between events in the queries; and (iv) event accumulation, for instance aggregating

data from previous events to discover new ones. The XChangeEQ syntax contains a DETECT statement which points the the targeted event and precises a function (and a variable) to be used in order to detect the aforementioned event. The ON part defines the event pattern, the where clause filters the results based on specific constraints. Finally, the END keyword terminates the statement. In the following, we illustrate again the empty room event in XChangeEQ.

**XChangeEQ Syntax**

```
DETECT <event_label>
ON <event_Structure>
WHERE <constraints>
END
```

**XChangeEQ Query**

```
DETECT empty_room
ON face
WHERE no_face_detected
END
```

### 3.3.3 Physical Languages

This category of languages includes data model specific works. We detail here languages that were specifically designed for either relational database or linked data management systems. Therefore, the following EQL are either inspired from or directly extend SQL/SPARQL.

**ESPER.** In [35], an implementation for event detection in database systems is presented. The authors proposed an SQL-like syntax for event processing. Therefore, known operators such as CREATE, SELECT, INSERT, UPDATE, and DELETE are available for event definition and detection. ESPER also includes temporal operators and a specific statement for event definition (i.e., the pattern). In addition to the aforementioned advantages, this language has a fast learning curve since it is highly similar to traditional SQL. The following describes the syntax for the SELECT statement. The pattern keyword defines the event and the right arrow denotes the 'followed by' operator.

**ESPER Syntax**

```
SELECT *
FROM PATTERN [
<event_pattern>]
```

**ESPER Query**

```
SELECT *
FROM PATTERN [
face_detected → no_face_detected]
```

**CQL.** In [9], the authors propose a language that can be used for event definition/retrieval. CQL extends SQL by emphasizing on continuous data streams and queries. The authors add temporal operators, sliding windows, and window parameters to better handle continuous data. They do so by providing mappings among streams and relations (i.e., from relations to relations, from streams to relations, and from relations to streams). From these mappings a precise and general interpretation for continuous queries is defined. The following describes the syntax for defining a new event (i.e., by creating its respective table).

**CQL Syntax**

```
CREATE TABLE <event_label> (
<property1> <datatype> [PRIMARY KEY],
<property2> <datatype>,
<property3> <datatype> ) ;
```

**CQL Query**

```
CREATE TABLE fire (
id int PRIMARY KEY,
temperature float,
smoke boolean ) ;
```

**C-SPARQL.** In [13], the authors extend SPARQL to consider stream data in the queries. To do so, they integrate sliding time windows. Supporting streams in RDF format guarantees interoperability and opens up important applications, in which reasoners can deal with knowledge that evolves over time. The following describes the syntax, and an example of a C-SPARQL event definition as a stream. The Range and Step values are needed to configure the sliding window.

**C-SPARQL Syntax**

```
REGISTER STREAM <event_label> AS
CONSTRUCT {
( ?<property> <ontology_IRI>:<object_property> ?<property> )}
FROM STREAM <URI>
[RANGE <value1> STEP <value2>]
WHERE {<condition>}
```

**C-SPARQL Query**

```
REGISTER STREAM empty_Shop1 AS
CONSTRUCT {
( ?face <ontology_IRI>:not_Detected_In ?shop )}
FROM STREAM <www.uri/of/ontology/sensornetwork.trdf>
[RANGE 30 MIN STEP 5 MIN]
WHERE ?shop ontology:label "Shop1"
```

**SPARQL$_{Stream}$.** In [24], the authors extend SPARQL to consider data streams. They introduce temporal windows that could be configured with a start and end parameters, as well as an optional step parameter that represents a number of temporal units (e.g., hours, minutes, days). In comparison with C-SPARQL, SPARQL$_{Stream}$ only focuses on time-based windows. The following describes the syntax, and an example of a SPARQL$_{Stream}$ query that returns windspeed every minute for the last ten minutes.

**SPARQL$_{Stream}$ Syntax**

```
conceptmap-def <Concept_Name> uri-as <URI>
described-by attributemap-def <Attributes_Definitions>
```

**SPARQL$_{Stream}$ Query**

```
SELECT RSTREAM ?windspeed
FROM STREAM <windStream>
[FROM NOW - 10 MINUTES TO NOW STEP 1 MINUTE]
```

**T-SPARQL.** In [41], the authors present an extension of SPARQL to integrate temporal features (e.g., annotating triples with time stamps) for better querying of RDF triples over time. The language is equipped with the basic temporal constructs and works with an extended set of the temporal datatypes, functions and operators already present in SPARQL. The following provides the syntax, and an example of a SELECT statement in T-SPARQL.

**T-SPARQL Syntax**

```
SELECT ?<property>
FROM <event_label>
WHERE {<condition>.
FILTER (<time>).}
```

**T-SPARQL Query**

```
SELECT ?temperature
FROM overheating_event
WHERE { ?temperature > 30°.
FILTER ("2010-01-01"^^xs:date).}
```

**SPARQL-ST.** In [75], the authors extend SPARQL by adding operators for spatial/temporal queries. This covers the definition and manipulation of spatial shapes and temporal entities. In addition to normal SPARQL variables (denoted with a ? prefix), SPARQL-ST introduces a spatial variable type (denoted with a % prefix) and a temporal variable type (denoted with a # prefix). The following illustrates the usage of the spatial and temporal variable types.

**SPARQL-ST Syntax**

```
SELECT ?<p>, %<s>, #<t>
FROM <event_label>
WHERE {
?<p> <condition> #<t>.
SPATIAL FILTER
(<function>(%<s>))}
```

**SPARQL-ST Query**

```
SELECT ?sensor, ?o, %s, #t
FROM empty_room
WHERE {
?sensor ontology:produced ?o #t.
SPATIAL FILTER
(?o inside(%s))}
```

**SPARQL-MM.** In [58], the authors also extend SPARQL to consider multimedia data, and media fragments. This language aims to improve semantic multimedia data retrieval. The extension includes media specific concepts and functions in order to unify the access to Linked Media. The following illustrates the syntax for retrieving multimedia data segments and ordering them based on a custom multimedia function (e.g., their duration). The data discussed in this example can be used in specific event definitions (e.g., the empty room event could rely on video footage to detect faces).

**SPARQL-MM Syntax**

```
SELECT ?<mm>
WHERE {
?<mm> <condition>.}
ORDER BY <function>
```

**SPARQL-MM Query**

```
SELECT ?video
WHERE {
?video ontology:is_A "Video".}
ORDER BY ontology:duration(?video)
```

**EP-SPARQL.** In [7], the authors integrate event processing operators, most notably the sequence operator, into the SPARQL syntax. This work allows the definition of simple and complex event patterns in a linked data management system. The work provides the syntax and formal semantics of the language and devises an effective execution model for the proposed formalism. To illustrate this language's capability of defining composite events, we provide the following example which shows how a fire event can be defined as the sequence of two events: high temperature, and appearance of smoke.

**EP-SPARQL Syntax**

```
CONSTRUCT {?<event>} WHERE
{<event_1> <operator> <event_2>}
```

**EP-SPARQL Query**

```
CONSTRUCT {?fire} WHERE
{overheating SEQ smoke}
```

### 3.3.4 Discussion

Conceptual languages are intuitive, practical, and allow various composition operators for event definition. Their syntax is also independent from specific data models (e.g., SQL or SPARQL). However, they all suffer from the same limitations. None of them covers the environment or sensor network definition in their queries (cf. Criterion 1). They mainly focus on the definition and retrieval of events while neglecting other tasks such as updating definitions or inserting data (cf. Criterion 2). They also do not consider spatial/temporal distributions (cf. Criterion 4). Logical languages are re-usable in different contexts since their syntax, a logical rule-based notation, is independent of specific data models (cf. Criterion 6). They also cover the majority of temporal and composition operators. However, they do not cover spatial/temporal distributions (cf. Criterion 4). These languages have not fully detailed query re-writing (cf. Criterion 5), and they mainly focus on the events. They cannot be used to define and manage the environment and sensor network components (cf. Criteria 1-2). Physical languages are all user friendly since they extend known languages. They cover definition and manipulation queries for various components or entities (cf. Criterion 2). They also provide a basis for spatial/temporal operators and query re-writing (cf. Criterion 5). However, distribution queries are not considered (cf. Criterion 4) and their high reliance on a specific data model syntax (SQL or SPARQL) limits their re-usability in different systems (cf. Criterion 6). For instance, EP-SPARQL cannot be used in a relational database infrastructure. Table 3.1 recapitulates the comparison between the aforementioned languages.

TABLE 3.1: Event Query Languages - Related Work Comparison

| Criteria | | *Component Coverage* | *Basic Querying* | *Data Diversity* | *Spatial/ Temporal Distributions* | *Handling Environment Dynamicity* | *Re-usability* |
|---|---|---|---|---|---|---|---|
| **Conceptual Languages** | SNOOP [26] | *NO* | *NO* | *NO* | *NO* | *NO* | *YES* |
| | CeDR [14] | *NO* | *NO* | *NO* | *NO* | *NO* | *YES* |
| | SaSE [42] | *NO* | *NO* | *NO* | *NO* | *NO* | *YES* |
| **Logical Languages** | ETALIS [8] | *NO* | *NO* | *NO* | *NO* | *NO* | *YES* |
| | XChangeEQ [21] | *NO* | *NO* | *NO* | *NO* | *NO* | *YES* |
| **Physical Languages** | ESPER [35] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | CQL [9] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | C-SPARQL [13] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | SPARQL$_{Stream}$ [24] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | T-SPARQL [41] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | SPARQL-ST [75] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |
| | SPARQL-MM [58] | *YES* | *YES* | *YES* | *NO* | *NO* | *NO* |
| | EP-SPARQL [7] | *YES* | *YES* | *NO* | *NO* | *NO* | *NO* |

## 3.4 Background & Preliminaries

A syntactic metalanguage is useful whenever a clear formal description and definition is required. EBNF is defined by the International Organization for Standardization (ISO 14977[9]). It proposes a notation for defining the syntax of a language using rules. Each rule names part of the language (called a non-terminal symbol) and then defines its possible forms. A terminal symbol is an atom that cannot be split into smaller components of the language. EBNF extends the original BNF to avoid lengthier rules by adding notations for options and repetitions. Furthermore, EBNF includes mechanisms for enhancements, defining the number of repetitions, excluding alternatives, and adding comments. The following resumes the main characteristics of EBNF:

- Terminal symbols of the language are quoted so that any character, including one used in EBNF, can be defined as a terminal symbol of the language being defined.
- Each rule has an explicit final character so that there is never any ambiguity about where a rule ends.
- The [ ] symbols indicate optional rules/statements.
- The { } symbols indicate repetition.
- The ( ) symbols group items together. It is an obvious convenience to use the brackets symbols in their ordinary mathematical sense.

Table 3.2 details the main EBNF notations and their usage.

[9] https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

TABLE 3.2: EBNF Notations

| Usage | Notation |
|---|---|
| Definition | = |
| Concatenation | , |
| Termination | ; |
| Alternation | \| |
| Option | [ ... ] |
| Repetition | { ... } |
| Grouping | ( ... ) |
| Terminal String | ... |
| Terminal String | ' ... ' |
| Comment | (* ... *) |
| Special Sequence | ? ... ? |
| Exception | - |

EBNF provides all the required tools for our syntax. Moreover, it allows the conception of technology independent queries (i.e., queries that do not depend from any data model specific syntax). This highlights the ability to re-use (cf. Criterion 6) the language since EBNF can be parsed into various data model specific instances, such as SQL or SPARQL, depending on the underlying infrastructure [56, 106]. Finally, any component from the connected environment (i.e., related to the environment, sensor network, event, and application domain modeling) can be expressed/defined using EBNF.

To conclude, we propose in the following section the Event Query Language for Connected Environments (EQL-CE). Our proposal has three layers (conceptual, logical, and physical). It uses EBNF to allow re-usability, and covers various components, query types, and datatypes. Finally, we introduce our proposed query optimizer that addresses spatial/temporal distributions, and the dynamicity of the environment.

## 3.5 EQL-CE Proposal

In this section, we detail our proposed language. We start by presenting the EQL-CE framework [67]. Then, we focus on the syntax, and how to define various connected environment components.

### 3.5.1 The EQL-CE Framework

We structure our proposal into three layers: (i) the conceptual layer provides an overview of the connected environment's components and their relations in the form of a graph; (ii) the logical layer allows the construction of generic queries written in EBNF (Extended Backus-Naur Form) syntax; and (iii) the physical layer parses the EBNF queries into a data model-specific language (e.g., SQL, SPARQL) and executes the parsed queries. A simplified overview of EQL-CE is presented in Figure 3.5. In the following we detail each layer separately.

#### 3.5.1.1 Conceptual Layer

Here, we detail the top layer of EQL-CE. The conceptual layer provides a clear and easy to exploit conceptual view of the connected environment. Therefore, we use a

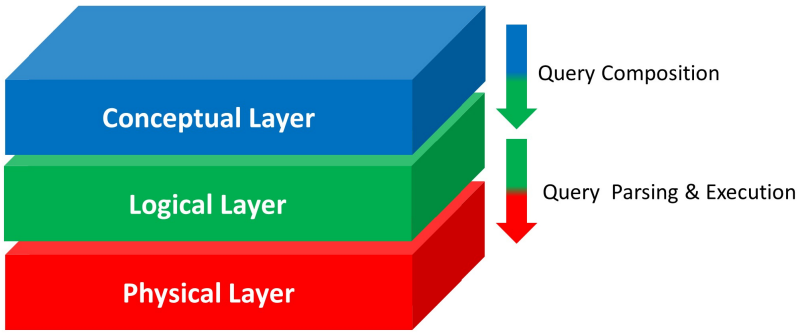


FIGURE 3.5: EQL-CE Overview

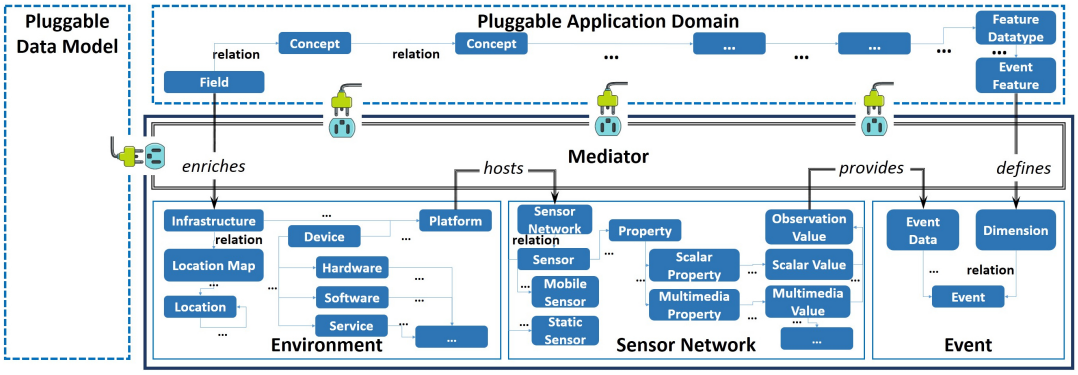


FIGURE 3.6: EQL-CE Conceptual Layer

graph to represent the various elements (i.e., components and properties). The latter are split into the following categories (cf. Figure 3.6):

**Core Modeling.** This part contains the basic elements that always exist in a connected environment. For a clear organization, we group the elements into the following two parts:

- Sensor Network modeling, where we represent (i) sensor networks; (ii) various sensor types (e.g., static, mobile); (iii) the different types of properties (i.e., scalar, multimedia) observed by sensors; and (iv) the observation values produced by sensors (i.e., textual values, multimedia objects and their respective metadata).
- Environment modeling, where we represent (i) platforms (i.e., infrastructure, devices) that host sensors or sensor networks; (ii) physical infrastructures, such as buildings, and their detailed description (i.e., location maps, spatial relations); (iii) devices, such as mobile phones, and their detailed description (i.e., hardware, software, provided services).

Many other components can still be added to the core part. The full description of the environment and sensor network can be inspired from the HSSN[10] ontology [68].

**Event Modeling.** This part contains the representation of events that one might wish to detect in a connected environment. Here, the application domain should also be considered since it affects the definition of specific events. For instance a

[10] http://spider.sigappfr.org/research-projects/hybrid-ssn-ontology/

body overheating (medical) event cannot be defined the same way as a room overheating (environmental) event. Hence, the application domain dictates the type of an event, its describing features, its pattern, and the required data for its detection. Therefore, we do not detail the event modeling, we keep it generic and restrict it to the following components: (i) event that defines an event and its type; (ii) dimensions to mathematically represent the event features (provided by the Application Domain) in a n-dimensional space (we formalize and detail the generic event definition in Chapter 5); and (iii) event data to represent sensor observations that contributed in each event. This allows us to have a generic event definition that applies to various events from different application domains. All context specific details are defined in the application domain and then imported in the event definition via the mediator.

**Application Domain Modeling.** This part represents the application domain (e.g., medical, energy, military). Since these elements differ from one field to another, this part is pluggable into the conceptual model. It contains basic components/properties denoted concepts and relations respectively. Instances of the concept component can be used to define any domain specific entities, and instances of the relation property can be used to interconnect the concepts (e.g., Figure 3.6 shows an Event Feature concept that helps define event dimensions). This allows the customization of environment descriptions and event definitions based on specific contexts. For instance, one might wish to represent medical equipment and health related constraints when modeling a hospital environment. These elements are not the same when describing a shopping center. Similarly, what describes medical related events is different from normal every day events that happen in a mall. To conclude, this part of the data model complements the event description on one side, and enriches the environment representation on the other.

**The Mediator.** This part of the conceptual model only contains properties that ensure the interconnection of the previously mentioned parts (i.e., the core, event, and application domain). For instance, a platform hosts a sensor network, the observation values produced by the sensors provide event data, the event dimensions are defined by event features, and the concept field enriches the description of an infrastructure based on the application domain. In addition, the mediator can also be used to plug in an external data model and align it with the existing elements.

#### 3.5.1.2 Logical Layer

The middle layer of EQL-CE, denoted the logical layer, allows users to compose/design their queries. The process starts by choosing a specific query type. To cover a wider set of functionality (cf. Criterion 2), we provide three main groups of queries:

- The Component Definition Language defines the structure of components. Various query types are included in this group (e.g., CREATE, ALTER, RENAME, DROP).
- The Component Manipulation Language handles component instances. Here we propose the various query types that allow data management in a connected environment (e.g., SELECT, INSERT, UPDATE, and DELETE).

- Component Access Control (e.g., GRANT, REVOKE). These queries manage access rights to component data. We detail access control tasks in a dedicated future work.

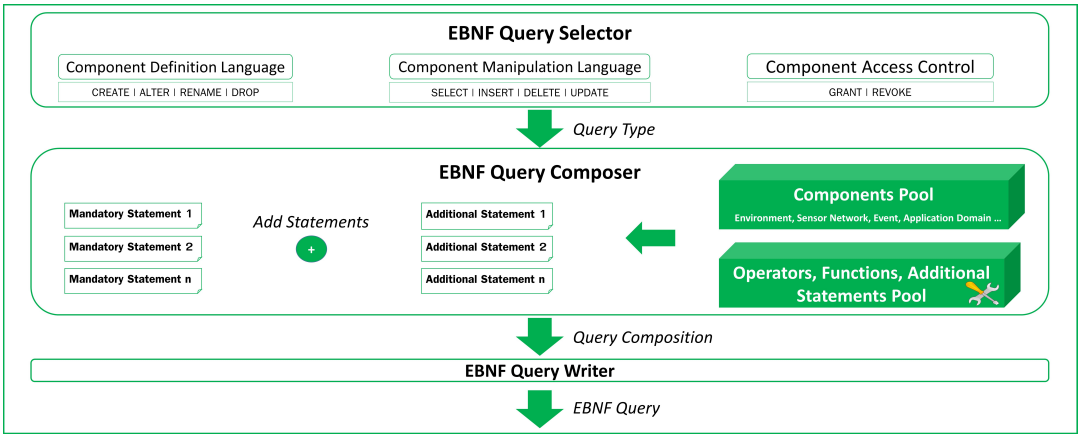


FIGURE 3.7: EQL-CE Logical Layer

The process of composing a query is described in Figure 3.7. First, the user chooses the query type (e.g., CREATE, INSERT, DELETE). Then, the user starts filling the mandatory statements (e.g., what to CREATE, what to SELECT, from which component). Once this is done, the user can add optional statements for filtering, ordering, calling external functions. Finally, the query is written using an Extended Backus-Naur Form syntax, denoted EBNF [107]. We use EBNF since it allows the conception of technology independent queries (i.e., queries that do not depend from any data model specific syntax). This highlights the ability to re-use (cf. Criterion 6) EQL-CE in different setups, since EBNF can later be parsed, in the physical layer, to a specific data model instance, such as SQL or SPARQL, depending on the underlying infrastructure [56, 89, 106]. Any component from the conceptual model (i.e., related to the environment, sensor network, event, and application domain modeling) can be defined, manipulated, and protected using these queries (cf. Criterion 1). Finally, the EBNF query is sent to the physical layer.

#### 3.5.1.3 Physical Layer & Query Optimizer

The bottom layer of EQL-CE (cf. Figure 3.8) saves the received EBNF queries in a dedicated storage unit for future use. Then, it parses the aforementioned queries into a specific syntax depending on the underlying data model (e.g., SQL, SPARQL). Finally, the parsed query is saved and sent to the query run engine where it is executed. If needed, external functions, methods, or even algorithms are called (e.g., string comparison functions, mathematical libraries). All the above describes how EQL-CE can be re-used in various contexts, since it is independent from any technological infrastructure (cf. Challenge 6). Using the EBNF queries, one can define the data model and all its various related components (cf. Challenge 1). In addition, EQL-CE allows users to handle instances of each component for data retrieval, modification, deletion, security/privacy, and event detection by providing a plethora of functionality (cf. Challenge 2). However, when defining specific events, one might need to manage the spatial distribution of sensors over a location (cf. Need 4). For instance, consider k-nearest sensors to a specific location, or all sensors within a range R of a point in space. Also, one might consider mathematical distributions

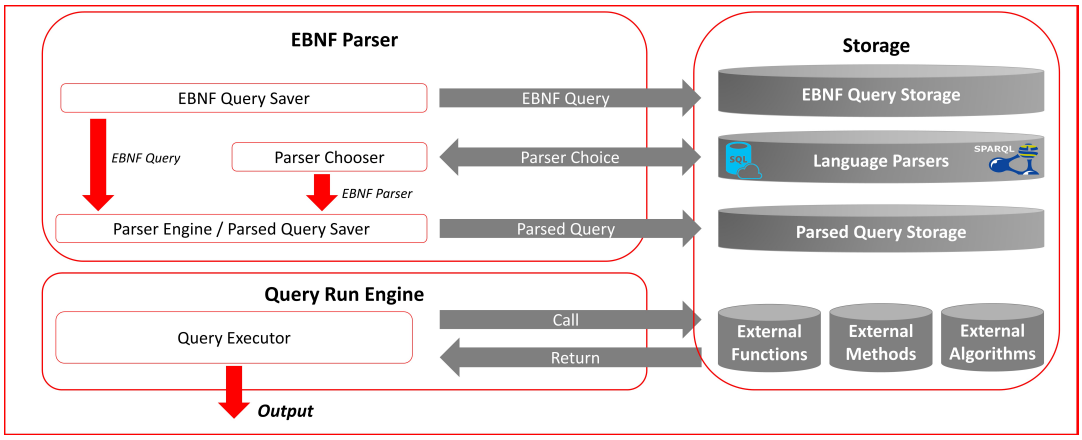


FIGURE 3.8: EQL-CE Physical Layer

of sensors over a zone (e.g., even distribution). Similarly, one might need to manage the temporal distribution of sensor observations for specific events (cf. Need 5). For example, selecting the k-most recent sensor observations, or all observations that were produced during a specific time lapse. Also, one might need to select observations based on specific sensing rates. To do so, the query optimizer allows the integration of spatial/temporal distribution functions in the queries (cf. Challenge 4). Finally, in dynamic connected environments sensors might suffer from breakdowns, mobile sensors could enter/leave the network, or even change locations. This is challenging since event definitions rely on sensors and their provided observations. Hence, some previously defined events might become obsolete over time. Therefore, in some cases, queries need to be re-written or updated in order to handle the dynamicity of the environment, and keep up with its evolution (cf. Criterion 5 and Challenge 5). This is also possible via the query optimizer. We leave the details of query re-writing to the next Chapter.

### 3.5.2 The EQL-CE Syntax

Here, we detail the syntax that describes the structure of the various connected environment components. To avoid redundancies, we state here that each component has a unique identifier.

#### 3.5.2.1 Environment Component Syntax

Sensors and sensor networks are hosted on platforms. Each platform has a unique identifier and a type (cf. Syntax 1).

**Syntax 1: Defining the structure of a Platform**

```
CREATE PLATFORM ( [ ID ] <platform_id>
[ , [ TYPE ] <type> = 'infrastructure' | 'device' ] ) ;
```

We define two types of platforms: (i) infrastructures; and (ii) devices. Infrastructures represent physical, real world environments (e.g., office, building, forest). Each infrastructure has a location map to describe spatial features, and a set of hosted platforms such as other infrastructures or devices (cf. Syntax 2). A location map has a

set of distinct locations (cf. Syntax 3), and each location has a description that details its geometric shape, coordinates in space, and a set of spatial relations with other locations (cf. Syntax 4). We also allow users to define external spatial relations (from the application domain) and use them for inter-location ties.

**Syntax 2: Defining the structure of an Infrastructure**

```
CREATE INFRASTRUCTURE ( [ ID ] <infrastructure_id>
[ , [ LOCATION MAP ] <location_map_id> ]
[ , { [ PLATFORM ] <platform_id> } ] ) ;
```

**Syntax 3: Defining the structure of a Location Map**

```
CREATE LOCATION MAP ( [ ID ] <location_map_id>
[ , { [ LOCATION ] <location_id> } ] ) ;
```

**Syntax 4: Defining the structure of a Location**

```
CREATE LOCATION ( [ ID ] <location_id>
[ , [ DESCRIPTION ] <description_id> ] [ , { ( [ RELATION ]
<relation> ,  [ LOCATION ] <location_id> ) } ] ) ;

<relation> = 'contains'|'covers'|'crosses'|'equals'|'includes'|
'touches'|'isAbove'|'isBelow'|'isCloseTo'|'overlaps'|'isRightOf'|
'isDisjointWith'|'isFraFrom'|'isLeftOf'| <spatial_relation_id> ;
```

Devices (cf. Syntax 5) are also considered platforms since they are capable of hosting sensors. We describe their hardware, software, and provided services in Syntax 6. The descriptions for hardware might include information about the device's power supply, memory, processor, network interface, and expansion cards where sensors are embedded. The description of software might include details about the operating system. And finally, the services descriptions might specify the provided functionality, and input/output.

**Syntax 5: Defining the structure of a Device**

```
CREATE DEVICE ( [ ID ] <device_id>
[ , { [ HARDWARE ] <hw_id> } ] [ , { [ SOFTWARE ] <sw_id> } ]
[ , { [ SERVICE ] <service_id> } ] ) ;
```

**Syntax 6: Defining the components of a device**

```
CREATE HARDWARE ( [ ID ] <hw_id>
                  [ , [ DESCRIPTION ] <description_id> ] ) ;

CREATE SOFTWARE ( [ ID ] <sw_id>
                  [ , [ DESCRIPTION ] <description_id> ] ) ;

CREATE SERVICE ( [ ID ] <service_id>
                 [ , [ DESCRIPTION ] <description_id> ] ) ;
```

#### 3.5.2.2 Sensor Network Component Syntax

When considering the sensor network, many components can be defined. For the sake of brevity, we choose here to detail the structure of observable properties in the environment (cf. Syntax 7), sensor observations (cf. Syntax 8), and sensors (cf. Syntax 9). Various properties can be monitored in a connected environment (e.g., temperature, noise, humidity). Some are scalar (i.e., textual) and others multimedia (i.e., audio, video, images). And each property is linked to a set of sensor observations. Each observation, has a description (e.g., unit of measurement), data value (if scalar) or a data object/file (if multimedia) alongside a datatype. Finally, each observation is mapped to a set of metadata tag/value pairs.

**Syntax 7: Defining the structure of a Property**

```
CREATE PROPERTY ( [ ID ] <property_id>
[ , [ TYPE ] <type> = 'scalar' | 'audio' | 'image' | 'video' ]
[ , { [ [ SCALAR | MEDIA ] OBSERVATION ] <observation_id> } ] ) ;
```

**Syntax 8: Defining the structure of an Observation**

```
CREATE [ SCALAR | MEDIA ] OBSERVATION ( [ ID ] <observation_id>
[ ,  [ DESCRIPTION ] <description_id> ]
[ , ( [ DATA VALUE ] <value_id> | [ DATA OBJECT ] <object_id> ,
      [ DATATYPE ] <datatype_id> ) ]
[ , { ( [ METADATA TAG ] <tag> : [ METADATA VALUE ] <value> ) } ]
) ;
```

Finally, we define static or mobile sensors. Each having (i) a description; (ii) a location history record containing a set of location/time interval pairs; (iii) a coverage area history record containing a set of coverage area/time interval pairs; (iv) a set of sensed properties/produced observations; and (v) the platform in which the sensor is embedded/hosted. We should mention that the location/coverage area records must contain at all times a current value for the sensor location and coverage area (i.e., a location/coverage area with an ongoing time interval).

**Syntax 9: Defining the structure of a Sensor**

```
CREATE SENSOR ( [ ID ] <sensor_id>
[ , [ TYPE ] <type> = 'static' | 'mobile' ]
( [ , WITH
  [ , { [ DESCRIPTION ] <description_id> } ]
  [ , [ LOCATION HISTORY ] <location_history> =
  { ( [ LOCATION ] <location_id> , [ TIME INTERVAL ] <ti> ) } ]
  [ , [ COVERAGE HISTORY ] <coverage_history> =
  { ( [ COVERAGE AREA ] <area_id> , [ TIME INTERVAL ] <ti> ) } ]
] )
( [ , SENSING { [ PROPERTY ] <property_id> } ] )
( [ , PRODUCING { [ OBSERVATION ] <observation_id> } ] )
( [ , HOSTED ON [ PLATFORM ] <platform_id> ] ) ) ;
```

#### 3.5.2.3 Event Component Syntax

We define an event (cf. Syntax 10) by assigning to it what we called an event space, an n-dimensional space where each dimension represents an event describing feature. In addition, since the events are detected based on sensor data, we map a set of contributing sensors to each event definition. When defining the event, one might choose a specific set of sensors, or any available ones that fit the event needs.

**Syntax 10: Defining the structure of an Event**

```
CREATE EVENT ( [ ID ] <event_id>
[ , [ EVENT SPACE ] <event_space_id> ]
[ , USING { [ SENSOR ] <sensor_id> } ] ) ;
```

The event space (cf. Syntax 11) contains a set of features each having some related conditions (e.g., temperature feature with a condition greater than 35°C). Finally, all observations belonging to an event are found within its space.

**Syntax 11: Defining the structure of an Event Space**

```
CREATE EVENT SPACE ( [ ID ] <event_space_id>
[ , { ( [ FEATURE ] <feature_id>
[ , [ CONDITION ] <condition_id> ] ) } ]
[ , { [ OBSERVATION ] <observation_id> } ] ) ;
```

#### 3.5.2.4 Application Domain Component Syntax

Since event features are better defined by an expert. We leave the feature syntax (cf. Syntax 12) to the application domain part. A feature is represented as a dimension in the event space. Therefore, each feature has a specific datatype for its values, a function that measures the distance between two instances belonging to the same feature, a default value, and a description. We provide a set of basic features, and leave the definition of advanced/complex features to domain experts.

**Syntax 12: Defining the structure of a Feature**

```
CREATE FEATURE ( [ ID ] <feature_id>
[ ,  [ DATATYPE ] <datatype> = 'integer' | 'float' | 'boolean' |
'date' | 'time' | 'date time' | 'character' | 'string' ]
[ ,  [ DISTANCE MEASURE ] <distance_measure_id> ]
[ ,  [ DEFAULT VALUE ] <value> ]
[ ,  [ DESCRIPTION ] <description_id> ] ) ;
```

The application domain experts also define the constraints related to each feature. Syntax 13 defines a condition as a set of statements each having operands and an operator. We provide various operators and allow users to import external operators/functions.

**Syntax 13: Defining the structure of a Condition (Part 1)**

```
CREATE CONDITION ( [ ID ] <condition_id>
                   [ , { STATEMENT <statement_id> } ] ) ;

CREATE STATEMENT ( [ ID ] <statement_id> ,
                   ( [ OPERAND ] <operand_id>,
                     [ OPERATOR ] <op>
                     [ , [ OPERAND ] <operand_id> ] ) ) ;

CREATE OPERAND ( [ ID ] <operand_id> ,
                 ( [ TYPE ] <type> = 'Time'|'Space'|'Other' ,
                   [ VALUE ] <val> ) ) ;

<val> = <string> | [ LOCATION ] <location_id> |
        [ TIMESTAMP ] <ts> | [ TIME INTERVAL ] <ti> ;

<op> = [ COMPARISON ] <cop> | [ TEMPORAL ] <top> |
       [ SPATIAL ] <sop> | FUNCTION <function_id>

<cop> = '=' | '<=' | '>=' | '<' | '>' | 'not' ;
```

**Syntax 13: Defining the structure of a Condition (Part 2)**

```
<top> = 'hasBegining'|'hasEnd'|'inside'|'intervalAfter'|
        'intervalBefore'|'intervalContains'|'intervalDisjoint'|
        'intervalDuring'|'intervalEquals'|'intervalFinishedBy'|
        'intervalFinishes'|'intervalIn'|'intervalMeets'|
        'intervalMetBy'|'intervalOverlappedBy'|
        'intervalOverlaps'|'intervalStartedBy'|'intervalStarts'|
        [ TEMPORAL RELATION ] <temporal_relation_id> ;

<sop> = 'contains'|'covers'|'crosses'|'equals'|'includes'|
        'isAbove'|'isBelow'|'isCloseTo'|'isDisjointWith'|
        'isFraFrom'|'isLeftOf'|'isRightOf'|'overlaps'|
        'touches'|[ SPATIAL RELATION ] <spatial_relation_id> ;

[ FUNCTION ] <function_id> ;
```

Finally, since the application domain description differs from one context to another, one needs a generic definition of application domain components and relationships that could be instantiated in any context. Therefore, we define a component named *Concept* (cf. Syntax 14), and an inter-concept relationship, denoted *Relation* (cf. Syntax 15). Relations can also be used between environment, sensor network, or event components.

**Syntax 14: Defining the structure of a Concept**

```
CREATE CONCEPT ( [ ID ] <concept_id>
                 [ , { ELEMENT <element_id> } ] ) ;

ELEMENT [ ID ] <element_id> = COMPONENT <component_id> |
                              ATTRIBUTE ( <name>, <datatype> ) ;
```

**Syntax 15: Defining the structure of a Relation**

```
CREATE [ <name> ] RELATION ( [ ID ] <relation_id>
[ , { ( CONCEPT SOURCE <concept_id> ,
        CONCEPT TARGET <concept_id> ) } ]
[ , { ( COMPONENT SOURCE <component_id> ,
        COMPONENT TARGET <component_id> ) } ] ) ;
```

To conclude, one can define the structure of various connected environment components (cf. Criterion 1).

### 3.5.3 Query Composition

In addition to component definition, users could rename, drop, or even alter the structure of previously defined components. Moreover, one could manage instances of each component through selection, update, insertion, and deletion queries (cf. Criterion 2). In the following, we provide the structure (skeleton) of component definition and management queries. We leave the component access control queries for a dedicated future work.

#### 3.5.3.1 Component Definition Language

The example below shows the skeleton (or mandatory parts) of a CREATE statement in EBNF. As illustrated in the previous subsection, one can create any component of the connected environment by defining its elements.

**CREATE query skeleton**

```
CREATE COMPONENT ( [ ID ]  <component_id>
{ [ , [ ELEMENT ] <element_id> ] } ) ;

ELEMENT [ ID ] <element_id> = COMPONENT <component_id> |
                              ATTRIBUTE ( <label> = <value> ) ;
```

One could also rename a previously defined component using the RENAME query which we illustrate below.

**RENAME query skeleton**

```
RENAME COMPONENT ( <label> [ TO ] <new_label> ) ;
```

Also, one could remove a component definition from the connected environment using the DROP statement. The CASCADE keyword could be used to remove the component from any other definition.

**DROP query skeleton**

```
DROP COMPONENT ( <label> [ CASCADE ] ) ;
```

Finally, one could ALTER the definition of a component in various ways: (i) by adding (ADD keyword) new elements to its definition; (ii) by removing (REMOVE keyword) existing elements from its definition; or (iii) by modifying (MODIFY keyword) existing elements either by renaming them or changing their definition (content).

**ALTER query skeleton**

```
ALTER COMPONENT ( [ ID ]  <component_id> ,  ADD | REMOVE (
{ [ , [ ELEMENT ] <element_id> ] } ) ) ;

ALTER COMPONENT ( [ ID ]  <component_id> ,  MODIFY (
{ [ , [ ELEMENT ] [ <label> ] <element_id> [ BY ]
      [ ELEMENT ] [ <new_label> ] <new_element_id>
  ] } ) ) ;
```

We provide here a few query examples to illustrate the usage of the RENAME, DROP, and ALTER statements.

**Query examples**

```
RENAME COMPONENT ( 'SENSOR' TO 'SENS' ) ;

DROP COMPONENT ( 'SENSOR' ) ;

ALTER INFRASTRUCTURE ( [ ID ]  <infrastructure_id> ,
ADD | REMOVE ( [ , [ LOCATION MAP ] <location_map_id> ]
               [ , { [ PLATFORM ] <platform_id> } ] ) ) ;

ALTER INFRASTRUCTURE ( [ ID ]  <infrastructure_id> ,
MODIFY ( [ , [ LOCATION MAP ] [ <name> ] <location_map_id> ]
         [ , { [ PLATFORM ] [ <name> ] <platform_id> } ] ) ) ;
```

#### 3.5.3.2 Component Manipulation Language

The example below shows how one might INSERT an instance of a previously defined component. This is done by assigning a specific value (instance) for each of the component's elements. More examples on the INSERT query are provided in the following section.

**INSERT query skeleton**

```
INSERT COMPONENT ( [ ID <component_id> ] <value>
{ [ , [ ELEMENT <element_id> ] <value> ] } ) ;
```

Once instances are created, one might need to retrieve specific data. The skeleton of the SELECT query is provided below. Additional statements could also be added after the WHERE clause (e.g., Filter By).

**SELECT query skeleton**

```
SELECT * | { [ ELEMENT ] <element_id> }
FROM COMPONENT
[ WHERE { [ CONDITION ] <condition_id> } ] ;
```

One might need to remove specific instances from the connected environment (e.g., sensors that no longer exist). The DELETE query is illustrated below.

**DELETE query skeleton**

```
DELETE COMPONENT [ CASCADE ]
WHERE { [ CONDITION ] <condition_id> } ;
```

Finally, since the environment is dynamic and ever changing, one might need to change some instances over time. The UPDATE query below shows how this could be done in EQL-CE.

**UPDATE Query Skeleton**

```
UPDATE COMPONENT
CHANGE { [ ELEMENT ] <element_id> <new_value> }
[ WHERE { [ CONDITION ] <condition_id> } ] ;
```

Before concluding this section, we provide a few examples of SELECT, DELETE, and UPDATE queries. We leave the INSERT query for the following section where it is thoroughly detailed.

**Query examples**

```
(* SELECT all platforms of type device *)
SELECT <platform_id>
FROM PLATFORM
WHERE <platform_type> = 'device' ;

(* DELETE all platforms of type device *)
DELETE PLATFORM
WHERE <platform_type> = 'device' ;

(* UPDATE a platform - Change type to device *)
UPDATE PLATFORM
CHANGE <platform_type> = 'device'
WHERE <platform_id> = 'Platform_1' ;
```

## 3.6 Illustration & Experimental Setup

In this section, we rely on the component definitions provided in Section 3.5 to create instances and illustrate the usage of EQL-CE in the Smart Mall connected environment. Then, we detail our experimental protocol for the evaluation of EQL-CE.

### 3.6.1 Illustration Example

We illustrate here how connected environment components are instantiated using the EQL-CE proposal. For the sake of brevity, we consider a snippet of the Smart Mall example, with less locations and sensors to avoid redundancies, in Figure 3.9.

#### 3.6.1.1 Environment Related Queries

The mall is a platform of type infrastructure. It has a location map containing five locations (the Hallway, Movie Theater, Grocery Store, Coffee Shop, and Shop 1). The locations and location map are instantiated in queries 1 and 2 respectively. Each location instance specifies the relation between the location and its neighbours.

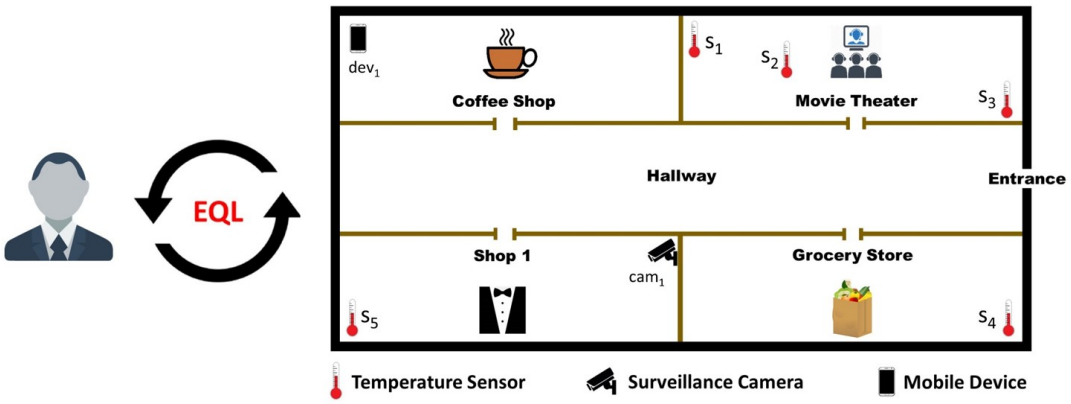


FIGURE 3.9: EQL-CE Illustration Example

**Query 1: Inserting the Locations**

```
INSERT LOCATION ( 'Movie_Theater' ,
{ ( 'touches' , 'Hallway' ) , ( 'isRightOf' , 'Coffee_Shop' ) ,
( 'touches' , 'Coffee_Shop' ) } ) ;

INSERT LOCATION ( 'Coffee_Shop' ,
{ ( 'touches' , 'Hallway' ) , ( 'isLeftOf' , 'Movie_Theater' ) ,
( 'touches' , 'Movie_Theater' ) } ) ;

INSERT LOCATION ( 'Shop_1' ,
{ ( 'touches' , 'Hallway' ) , ( 'isLeftOf' , 'Grocery_Store' ) ,
( 'touches' , 'Grocery_Store' ) } ) ;

INSERT LOCATION ( 'Grocery_Store' ,
{ ( 'touches' , 'Hallway' ) , ( 'isRightOf' , 'Shop_1' ) ,
( 'touches' , 'Shop_1' ) , ( 'isAcrossOf', 'Movie_Theater') } ) ;

INSERT LOCATION ( 'Hallway' ) ;
```

**Query 2: Inserting the Location Map**

```
INSERT LOCATION MAP ( 'Mall_Location_Map' , { 'Movie_Theater',
'Coffee_Shop', 'Shop_1', 'Grocery_Store', 'Hallway' } ) ;
```

The smart mall infrastructure hosts a mobile device $dev_1$. The device instance is shown in Query 3. We do not detail the representation of the hardware, software, or services. We only specify that a temperature sensor ('s_6') is embedded on the device.

**Query 3: Inserting the Device and its elements**

```
INSERT DEVICE ( 'dev_1' , { 'dev_1_Hardware' } ,
{ 'dev_1_Software' } , { 'dev_1_Service' } ) ;

INSERT HARDWARE ( 'dev_1_Hardware' , { 's_6' } ) ;

INSERT SOFTWARE ( 'dev_1_Software' ) ;

INSERT SERVICE ( 'dev_1_Service' ) ;
```

Since all infrastructure elements have been instantiated, one can now insert the infrastructure instance (cf. Query 4).

**Query 4: Inserting the Mall Infrastructure**

```
INSERT INFRASTRUCTURE ( 'Mall_Infrastructure' ,
'Mall_Location_Map' , { 'dev_1' } ) ;
```

The mall infrastructure and mobile device host sensors. Therefore, they are also considered platforms. Queries 5 and 6 insert the two platform instances.

**Query 5: Inserting the Mall Platform**

```
INSERT PLATFORM ( 'Mall_Platform' , <type> = 'infrastructure' ) ;
```

**Query 6: Inserting the Device Platform**

```
INSERT PLATFORM ( 'dev_1' , <type> = 'device' ) ;
```

#### 3.6.1.2 Sensor Network Queries

In regards to the sensor network related queries, we begin by instantiating the two properties that are currently monitored in the example: (i) the scalar property temperature; and (ii) the multimedia property video. This is shown in queries 7 and 8 respectively.

**Query 7: Inserting the temperature property**

```
INSERT PROPERTY ( 'temperature_property' , <type> = 'scalar' ) ;
```

**Query 8: Inserting the video property**

```
INSERT PROPERTY ( 'video_property' , <type> = 'video' ) ;
```

These properties are monitored by various sensors. Six temperature sensors exist in the smart mall. $s_1, s_2, and s_3$ are deployed in the movie theater, $s_4$ in the grocery store, $s_5$ in Shop 1, and $s_6$ is the only mobile sensor deployed on $dev_1$ which is currently located in the Coffee Shop. A surveillance camera $cam_1$ is deployed in Shop 1. Query 9 instantiates all the aforementioned sensors. Note that the mobile sensor $s_6$ has a previous/current location/coverage area.

**Query 9: Inserting all sensors (Part 1)**

```
INSERT SENSOR ( 's_1' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:44:27 ; now' ) } ,
    <coverage_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:44:57 ; now' ) } ) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;
```

**Query 9: Inserting all sensors (Part 2)**

```
INSERT SENSOR ( 's_2' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:54:27 ; now' ) } ,
    <coverage_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:55:27 ; now' ) } ) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;

INSERT SENSOR ( 's_3' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:42:27 ; now' ) } ,
    <coverage_history> = { ( 'Movie_Theater' ,
                             '19-04-2019 11:43:27 ; now' ) } ) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;

INSERT SENSOR ( 's_4' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Grocery_Store' ,
                             '19-04-2019 11:44:27 ; now' ) } ,
    <coverage_history> = { ( 'Grocery_Store' ,
                             '19-04-2019 11:44:57 ; now' ) } ) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;

INSERT SENSOR ( 's_5' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Shop_1' ,
                             '19-04-2019 11:54:27 ; now' ) } ,
    <coverage_history> = { ( 'Shop_1' ,
                             '19-04-2019 11:55:27 ; now' ) } ) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;

INSERT SENSOR ( 's_6' , <type> = 'mobile'  ,
WITH
( <location_history> = {
   ( 'Coffee_Shop' , '19-04-2019 11:42:27 ; now' )
   ( 'Shop_1' , '19-04-2019 10:43:27 ; 19-04-2019 11:23:20' ) } ,
  <coverage_history> = {
   ( 'Coffee_Shop' , '19-04-2019 11:43:27 ; now' ) ,
   ( 'Shop_1' , '19-04-2019 10:43:27 ; 19-04-2019 11:23:20' ) }
) ,
SENSING ( { 'temperature_property' } ] ) ,
HOSTED ON ( 'dev_1' ) ) ;

INSERT SENSOR ( 'cam_1' , <type> = 'static'  ,
WITH (
    <location_history> = { ( 'Shop_1' ,
                             '19-04-2019 11:25:14 ; now' ) } ,
    <coverage_history> = { ( 'Shop_1' ,
                             '19-04-2019 11:25:14 ; now' ) } ) ,
SENSING ( { 'video_property' } ) ,
HOSTED ON ( 'Mall_Platform' ) ) ;
```

When the network becomes operational, sensors will start producing observations.

The latter are sent to a middle-ware that will generate an insert query in order to push the observations into the data model according to the observation syntax (cf. Section 3.5). Then, the middle-ware will update both sensors and properties by mapping them to their related observations. Query 10 shows an insert query generated by the middle-ware for a temperature observation having a float value of '20.3' and two associated metadata for time and location of capture. Similarly Query 11 shows another temperature observation taken from the Coffee Shop. Query 12 instantiates a video observation taken from the surveillance camera in Shop 1. This observation includes the video recording file, temporal, location, and video length related metadata.

**Query 10: Inserting a temperature observation taken in Shop 1**

```
INSERT SCALAR OBSERVATION ( 'temperature_observation_1',
( '20.3' , 'float' ) ,
{ ( 'timestamp' : '19-04-2019 11:34:54' ) ,
  ( 'location' : 'Shop_1' ) } ) ;
```

**Query 11: Inserting a temperature observation taken in the Coffee Shop**

```
INSERT SCALAR OBSERVATION ( 'temperature_observation_2',
( '19.3' , 'float' ) ,
{ ( 'timestamp' : '19-04-2019 11:44:27' ) ,
  ( 'location' : 'Coffee_Shop' ) } ) ;
```

**Query 12: Inserting a video observation taken in Shop 1**

```
INSERT MEDIA OBSERVATION ( 'video_observation',
( 'recording.mpeg' , 'video' ) ,
{ ( 'timestamp' : '19-04-2019 11:35:14' ) ,
  ( 'location' : 'Shop_1' ) ,
  ( 'duration' :  '123 s' ) } ) ;
```

#### 3.6.1.3 Event Queries

In order to give an example of how an event can be instantiated we define next an intrusion event in Shop 1. The mall manager relies on video sensor $cam_1$ for the detection of this event. He/She defines the event as a face detected by $cam_1$ in the Shop after 8 PM. Three features define this event: (i) time with a condition after 8 PM; (ii) location with a restriction to Shop 1; and (iii) a detected face with a Boolean value equals true. In this example, the manager uses our provided basic features for time, location, and detected face (cf. Query 13) where we only define the feature as an identifier assigned to a datatype. The manager also creates the required conditions for each feature (cf. Query 14). However, one can use the application domain queries to define more complex/advanced features/conditions if needed. Finally, query 15 details the event space, and query 16 instantiates the event.

**Query 13: Inserting features for the intrusion event**

```
INSERT FEATURE ( 'time_f' , 'date-time' ) ;

INSERT FEATURE ( 'location_f' , 'string' ) ;

INSERT FEATURE ( 'face_f' , 'Boolean' ) ;
```

**Query 14: Inserting conditions for the intrusion event features**

```
INSERT CONDITION ( 'condition_1' , { 'statement_1' } ) ;
INSERT STATEMENT ( 'statement_1' ,
            ( cam_1.Observation.timestamp , After(' 8 PM ') ) ) ;

INSERT CONDITION ( 'condition_2' , { 'statement_2' } ) ;
INSERT STATEMENT ( 'statement_2' ,
            ( cam_1.Location , Equals(' Movie_Theater ') ) ) ;

INSERT CONDITION ( 'condition_3' , { 'statement_3' } ) ;
INSERT STATEMENT ( 'statement_3' ,
            ( cam_1.Observation , face_detect(' true ') ) ) ;
```

**Query 15: Inserting an event space for the intrusion event**

```
INSERT EVENT SPACE ('event_space_1' ,
{ ( 'time_f' , 'condition_1' ) ,
  ( 'location_f' , 'condition_2' ) ,
  ( 'face_f' , 'condition_3' ) } ) ;
```

**Query 16: Inserting the event definition**

```
INSERT EVENT ( 'intrusion_in_shop_1' , 'event_space_1' ,
               USING { 'cam_1' } ) ;
```

### 3.6.2 Experimental Protocol

We are currently implementing the EQL-CE query run engine and its query optimizer (cf. Chapter 4) as part of an online platform for event detection in connected environments. Since the development is still ongoing, we propose here the experimental protocol that we will use to evaluate the query language once the development is completed. We propose the following experiments.

#### 3.6.2.1 Query Cost Evaluation

Providing the user with the ability to define the entire connected environment allows him/her to manage all the components from scratch (e.g., locations, spatial ties, sensor capabilities, datatypes, event features). This is beneficial since the user has total control over the environment, and the ability to customize everything based on

his/her preferences. However, this might be costly in terms of the number of 'steps' (i.e., queries) required to achieve a specific task/objective. In this experiment, we set a list of objectives (e.g., defining a platform, a sensor, a location map) and quantify the required query batch size and the total cost of achieving the task.

#### 3.6.2.2 Re-usability Evaluation

To ensure re-usability, EQL-CE provides a logical layer where generic queries are composed for all components and functionality. These queries rely on EBNF, a meta-language that is independent from any data model syntax. Then, these queries are parsed to any language (e.g., SQL, SPARQL) based on the underlying infrastructure. This helps cope with technological changes over time (if the infrastructure changes the language is not affected). In this test we evaluate the physical layer's ability to parse EBNF into various other languages (e.g., SQL, SPARQL). We re-iterate this experiment for each query type (i.e., SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, RENAME, DROP) and available component (e.g., sensor, platform, location map, observation).

#### 3.6.2.3 Performance Evaluation

In this test, we measure the run-time, and the resource consumption (CPU/RAM) when executing EQL-CE queries. This evaluation is two-fold. First, we aim to measure the impact of the query parsing algorithm on performance. Then, we would like to do the same for query execution. We set various use cases where we measure run-time and resource consumption (e.g., parsing and executing simple queries, complex queries, and batch queries required for specific tasks).

## 3.7 Summary

Many challenges emerge when proposing a EQL adapted to connected environments. In this work, we addressed the issues of re-usability, data diversity, and covering various components/query types. To do so, we proposed EQL-CE: a three layered event query language for connected environments. We detailed its conceptual, logical, and physical layers. EQL-CE users compose EBNF queries, that can be later parsed into SQL, SPARQL, or other languages (re-usability). Our proposal covers various connected environment components (environments, sensor networks, events, and application domains) and query types (definition, manipulation, access control, event detection). We also proposed a query optimizer that allows query re-writing and the integration of spatial/temporal distribution functions.

# Chapter 4

# Handling Connected Environment Dynamicity

*"Everything changes and, somewhere along the line, I'm changing with it."*

*— Eric Burdon*

When considering an Event Query Language for Connected Environments, various challenges emerge (e.g., data volume, velocity, variety, sensor mobility, reliability). In this chapter, we focus on one particular challenge: coping with the dynamicity of the Connected Environment.

In EQL-CE, one could define events by detailing their pattern (event describing features and constraints), and relying on specific sensors that will provide the required sensed data (sensor observations). However, in connected environments that host hybrid sensor networks, various issues could challenge event definition and detection such as breakdowns, sensor mobility, and lack of adequate sensor distribution over the space. Due to sensors entering/leaving the network, changing locations, and breaking down, the aforementioned event queries could become obsolete over time (i.e., the data sources or the data itself will no longer be available).

To overcome these challenges, event queries should be updated or re-written in order to cope with the dynamicity of the environment. Therefore, we propose here a query optimizer that automatically discovers obsolete queries, and re-writes them by replacing the missing data sources (i.e., sensors) and data (i.e., sensor observations, event describing features). To do so, one needs to measure the similarity between sensors in order to find adequate substitutes. Moreover, a similarity measure is required for missing event features/sensed data replacement as well.

In the previous chapter, we proposed an Event Query Language (EQL) specifically designed for Connected Environments. Our proposal, denoted EQL-CE, is part of a framework that includes the query optimizer. The purpose of this chapter is to detail how query re-writing can be utilized within the query optimizer module in order to cope with the dynamicity of the environment. We detail here the query re-writing process, the proposed similarity measures, and present an illustration example that shows how the algorithm works. Finally, we discuss experiments and complexity evaluation.

## 4.1 Introduction

As previously mentioned, a Connected Environment is composed of various components (e.g., infrastructures, locations, sensors, observations, events). It is a hybrid and dynamic setup where one can find a diversity of sensors (e.g., static, mobile), platforms (e.g., infrastructures, devices, machines), and data (e.g., scalar, multimedia sensor observations). This diversity is both beneficial and challenging, since it introduces various issues related to:

- Sensor mobility & reliability. The connected environment contains both static and mobile sensors. This presents some challenges regarding the reliability of sensors in general, and in the case of mobile sensors, coping with mobility. More specifically, sensors might become unavailable over time since breakdowns and faults can occur, and mobile sensors could enter/leave the network, or change locations within the premises of the connected environment. The unavailable sensors could render some queries obsolete (i.e., the queries will need updating in order to return significant results).
- Platform compatibility. Since a diversity of platforms can be considered in a connected environment (e.g., infrastructures, devices, machines), issues regarding inter-operability between various platforms emerge. Each platform has a structure, and a specific configuration. This could lead to mismatching components in the connected environment. Bridging the differences between these platforms should also be considered in the query language.
- Data management. Since both scalar and multimedia data exist in a connected environment, any query language should address the diversity from various aspects (e.g., missing data/features, data cleaning and normalization, and data volatility). Also, since sensors are data producers (sources), the diversity of sensors could impact the availability of data (e.g., a mobile sensor that leaves the network can no longer provide data for queries).

In this work, we focus on two main challenges: (i) sensor mobility and reliability; and (ii) missing data/features. Since sensors are the data sources that provide useful observations in the connected environment, data might become missing, lacking, or even mismatching due to mobility, faults, and breakdowns. To highlight the relevance of these two challenges, we consider event-related queries in EQL-CE. Event queries are used to define events, by detailing their features, constraints, and required sensors/data. For instance, a room overheating event could be defined by the feature temperature with a condition of more than $30°C$ using a temperature sensor $s_1$. However, if the sensor leaves the environment, it can no longer serve this query. More importantly, the observations provided by this sensor become missing and the event, as currently defined, will not be detected anymore. Therefore, there is a need for continuously checking and updating obsolete queries (such as the one given in the aforementioned example) in order to ensure significant results when a user queries the environment for events. This entails checking the current status of the environment to verify if sensors (data sources), and their produced observations (data) are still available, since event queries rely on the aforementioned elements.

In the previous chapter, we compared various Event Query Languages (cf. Table 3.1) and found that none of the existing works [7, 8, 9, 13, 14, 21, 26, 35, 41, 42, 58, 75]

could cope with the dynamicity of the environment in order to overcome challenges related to sensor mobility, sensor reliability, and missing data.

To overcome this issue, we propose a query optimizer equipped with a module that re-writes obsolete queries in order to cope with the dynamicity of the connected environment. We suggest a two step process: first, a query analyzer checks if any re-writing is needed; and then, a query re-writer applies our proposed algorithm in order to update obsolete queries. The query rewriting algorithm replaces unavailable sensors (data sources) by available and similar substitutes. This ensures that the required data for query execution is not missing. To do so, we present a sensor similarity measure that quantifies the level of similarity between any two given sensors. If no similar sensors are available, the algorithm attempts to replace the data needed for a specific event query by other obtainable/available data without compromising the event definition. To do so, we also propose a way to measure inter-data similarity. In this chapter, we detail the structure of our query optimizer, the re-writing algorithm, and all its required functions.

The remainder of this chapter is organized as follows. Section 4.2 illustrates a scenario that motivates our proposal. Section 4.3 presents a background study on query re-writing. Section 4.4 provides a formalism that defines some the key terms. Section 4.5 details our proposed query optimizer, its modules, and all the related algorithms and functions. Then, Section 4.6 illustrates how query re-writing works and discusses some experiments. Finally, Section 4.7 summarizes the chapter.

## 4.2 Motivating Scenario

We consider again the Smart Mall example (cf. Figure 4.1) in order to illustrate the importance of coping with the dynamicity of a connected environment. We remind the reader that the smart mall is equipped with a hybrid sensor network (i.e., composed of diverse sensors, platforms, and data). A manager uses an EQL to query the environment for data management, and event definition/detection purposes. To

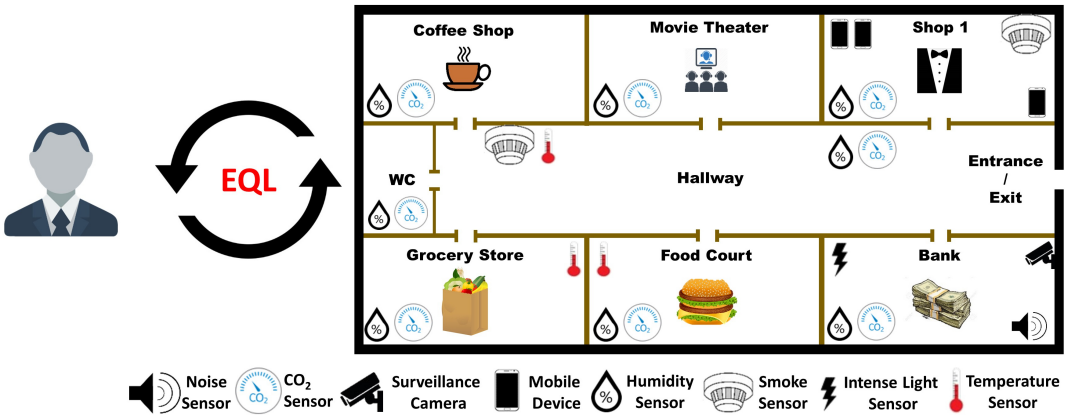


FIGURE 4.1: The Smart Mall

focus on the various needs behind coping with the dynamicity of the smart mall, we consider an event query that defines a *fire event*. This event is defined by three features: (i) *temperature* with a condition *high*; (ii) $CO_2$ with a condition *high*; and (iii) *smoke* with a condition *existence of smoke = true*. The mall manager is interested in

detecting this event in Shop 1. Moreover, the manager needs to assign sensors to this event query. Four mobile sensors capable of observing various properties, and three static sensors for smoke, humidity, and $CO_2$ are available in Shop 1 (cf. Figure 4.2). When considering the aforementioned query the following needs emerge:


FIGURE 4.2: Fire event in Shop 1

**Need 1.** Coping with sensor breakdowns. For the *fire event* query, the manager chooses to rely on Mobile Sensor 1 for temperature readings, the static $CO_2$ sensor, and the static smoke sensor. However, if the smoke sensor breaks down (cf. Figure 4.3) the query will return a null result because one of the data sources (i.e., the smoke sensor) is no longer available. Therefore, the sensor and subsequently the


FIGURE 4.3: Smoke sensor breakdown

data needed for the query cannot be retrieved. The EQL should be capable of coping with this issue. A potential solution could be to replace the smoke sensor by another capable of providing the same observations in Shop 1.

**Need 2.** Coping with sensor distributions. In order to manage the issue discussed in Need 1, the manager redefines the *fire event* by relying on Mobile Sensor 1 for both temperature and smoke readings (since this sensor is capable of providing smoke observations). The static $CO_2$ sensor is still used (cf. Figure 4.4). Since Mobile Sensor 1 is located in the upper left corner of the store, its provided observations are not indicative to the state of the entire shop (the static smoke sensor was located at the center of the shop). Therefore, it would be better to consider a more appropriate solution such as selecting various distributed sensors that could provide smoke observations.


FIGURE 4.4: Mobile Sensor 1 replaces the smoke sensor

**Need 3.** Coping with mobility. To address the sensor distribution issue mentioned in Need 2, the manager redefines the *fire event* query by assigning all mobile sensors, since they are evenly distributed in the shop, to produce smoke observations (cf. Figure 4.5). However, since the event now partially relies on mobile sensors,


FIGURE 4.5: Assigning all mobile sensors to smoke observation

it is possible that these sensors could move around, leave the shop, or even the entire environment. In this case, the query will remain obsolete. The EQL should be capable of coping with mobility related issues.

**Need 4.** Coping with mismatching features/data. Another issue could be that some (or all) mobile sensors are incapable of providing smoke observations (if they are not equipped with smoke sensors). In this case, no sensors could provide well distributed smoke observations, or any smoke observations at all. The EQL should also be capable of addressing such issues. An appropriate solution could be to replace the feature smoke by another that is available in Shop 1, similar to smoke, and does not compromise the *fire event* definition. Figure 4.6 shows how humidity for instance could replace smoke in the event definition.


FIGURE 4.6: Replacing smoke by humidity

However, when considering the aforementioned needs, the following challenges emerge:

**Challenge 1.** How to discover if a query is obsolete? How to know if query re-writing is required?

**Challenge 2.** How to replace sensors with others that are similar, available, and capable of contributing to a specific query? How to measure similarity between sensors?

**Challenge 3.** How to replace mismatching/missing data or features when sensor substitution is not possible? How to measure the similarity between event features?

**Challenge 4.** How to re-write queries when the underlying schema (i.e., the connected environment itself) is changing over time? Similarly, how to re-write queries when the existing sensors and features/data change?

Therefore, we propose a query optimizer that automatically detects and re-writes obsolete event queries. Since events rely on features/data (e.g., temperature, smoke) and data sources (i.e., sensors), the optimizer checks if all the required elements for a query are still available in the current state of the connected environment. If so, no re-writing is needed. However, if any required sensor/data are unavailable/missing our proposed algorithm re-writes the query. First, the algorithm attempts to replace missing sensors with others that are similar. If this is not possible, the missing feature/data is replaced without damaging the definition of the event.

## 4.3 Related Work On Query Rewriting

Query rewriting is a phase of query processing. It refers to the application of a number of transformations to a query (original query $q$) in order to produce an equivalent and optimized one (a new query $q'$). In this section, we recall the application domains that are impacted by query rewriting works. Then, we present the purposes and motivations behind such works. Finally, we detail the techniques used in the literature.

### 4.3.1 Usage Of Query Rewriting

Query rewriting or reformulation has been an interesting research topic for many years. Existing works [29, 36, 37, 64, 73, 91] target different application domains (e.g., information retrieval in search engines [64, 91] database management systems [29, 73], knowledge base management systems [36, 37]). Search engine queries are often considered and handled as strings. This alleviates the complexity of their reformulation. However, rewriting database or knowledge base queries is a more complex task. This is due to the set of constraints that the rewriting process should consider: (i) respecting a specific syntax; (ii) adapting to the structure of the query components (e.g., tables in a database, concepts in an ontology); and (iii) considering the relationships that tie query components together (e.g. inter-table relations in a database, object properties in an ontology).

### 4.3.2 Purposes Of Query Rewriting

Query rewriting can be utilized for different purposes (e.g., improving result accuracy, improving performance) [50, 54, 77]. To address the challenges mentioned in Section 4.2, we focus on rewriting techniques that improve result accuracy. This is due to the fact that missing sensors and data could lead to null, or inaccurate results (cf. Challenges 2 and 3). Performance related issues/optimizations will be addressed in a future dedicated work. When detail in the following three objectives of query rewriting for better result accuracy:

#### 4.3.2.1 Reducing The Gap Between Users & Data

In the context of information retrieval systems, a gap exists between the stored documents/data, and user queries [48]. This is due to many reasons, most notably the challenge of accurately describing and expressing a user's intent when composing a query. One of the various objectives of query rewriting is to reduce this gap by adjusting a given query and improving the accuracy of the query result. This entails rewriting specific query parts to better reflect the user's intent, filter unnecessary statements that could damage results, and/or complement, replace existing statements to enrich and improve the query answer.

#### 4.3.2.2 Increasing Result Recall

If a user submits a very selective or fine grained query, he/she might end up with an empty or null query answer. In this case, rewriting strategies such as relaxation and expansion are used to to retrieve a larger set of relevant results (i.e., increasing recall), thus avoiding null/empty answer sets. Query expansion [11, 25, 46] rewrites the original query in order to add relevant results to the original result set, even if the new additions do not have an exact match with the original query terms. It broadens (expands) the query by introducing additional tokens or keywords. To do so, one needs to know where and how to get the additional relevant tokens/keywords. This could be achieved by enriching the original query with abbreviations and synonyms of the existing terms (e.g., by checking dictionaries, or via supervised/unsupervised machine learning). Moreover, once the query is rewritten (i.e., expanded), one needs to evaluate the relevance of the new result set w.r.t. the initial one. To do so, results are given scores based on machine learning ranking models, or specific scores by comparing the new results with the original ones. Query relaxation [49, 91] is another strategy. It consists of removing or substituting query tokens. Ignoring or substituting fine grained tokens with more generalized ones, makes the query less restrictive and increases recall (achieves a wider set of relevant results). More specifically, query relaxation rewrites the query by only removing/generalizing tokens that aren't necessary for communicating the user's intent. Query relaxation can be achieved in various ways:

- Stop word removal: eliminating stop words that do not affect the meaning or objective of the query.
- Specificity: using lexical databases, TF-IDF techniques (term frequency–inverse document frequency) to assess how essential is every word of the query, hence unnecessary words could be eliminated.
- Syntactic analysis: removing optional/unnecessary query statements based on the query syntax (neglecting semantics).

- Semantic analysis: relying on inter-word semantic dependencies/ties instead of keyword frequency or query syntax, to determine the words that could be dropped without damaging the query meaning. Semantic similarity measures could be used to discover the ties between query words.

#### 4.3.2.3 Increasing Result Precision

If a user submits a very generic (non specific) query, he/she might end up with a huge amount of irrelevant results in the query answer. In this case, rewriting strategies such as segmentation and scoping are used to to retrieve a smaller set of relevant results (i.e., increasing precision), thus avoiding null/empty answer sets. Query segmentation [16, 17, 43, 96] attempts to divide the query into a sequence of semantic units, each of which consists of one or more tokens/words. Then, the most significant part of the query is considered. Although this provides a smaller query answer set, it filters out irrelevant results and keeps the most precise ones. Query segmentation requires (i) a correct segmentation of the query (e.g., "hybrid sensor network" could be segmented into "hybrid sensor" and "network" or "hybrid" and "sensor network"); (ii) choosing the most important segment in the query (e.g., "hybrid" or "sensor network"); and (iii) only keeping answers related to the most relevant segment. Query scoping [74, 98] rewrites queries by taking advantage of data structures. Documents and data often have an explicit structure that mirrors how users search for them. For instance, when querying for a temperature sensor, the word 'sensor' is interpreted as a device that senses data, and 'temperature' is considered as the property (natural phenomena) that could be observed by a sensor. Query scoping is done in two steps. First, each query word is tagged with a label that reflects its category (e.g., sensor to sensing device). This is not straightforward since the categories need to be distinguished correctly (using machine learning, classification techniques). Then, once the class attributes are established and the query words are tagged, the essence of the query can be captured by relying on the most relevant terms and their tags. The remaining terms are dropped.

**Recap.** Query rewriting strategies that increase recall tend to augment the result set size by adding more, potentially less relevant, terms. This helps avoid the issue of queries that return no, or few, results. In contrast, strategies that target precision decrease the query result set by filtering all unnecessary answers. This helps avoid the issue of queries that return huge amounts of heterogeneous results. In both cases, the accuracy is increased by better reflecting the user's intent when querying the data.

### 4.3.3 Existing Approaches

We review here some existing approaches from the literature. Some works [25, 37, 43, 64, 91] provide automatic query rewriting (i.e., do not include the user in the process) while others [29, 38, 49] propose cooperative solutions by integrating the user and his/her preferences in the rewriting process. In both methods, existing approaches rely on one or multiple rewriting techniques (e.g., expansion, relaxation, segmentation, scoping). Moreover, the following works target all the aforementioned application domains (e.g., rewriting for search engines, database management systems, and knowledge base management systems).

**Automatic Query Expansion.** In [25], the authors are interested in automatic query expansion. They present a computationally simple, and theoretically justified method for assigning scores to candidate expansion terms. The suggested ranking method is based on the differences between the distribution of terms in relevant documents, and the distribution of terms in all documents. Then, the authors conduct a series of experiments that highlight the utility of relying on ranking models based on distribution analysis when automatically expanding queries.

**Query Rewriting For SPARQL.** In [37], the authors propose a flexible query processing approach for SPARQL. It is based on query rewriting using query approximation and relaxation operators. Query Approximation consists of applying edit operators (such as deletion, insertion and substitution) to transform a well-defined regular expression into a new one. Query Relaxation relies on a fragment of RDF Schema (RDFS) entailment rules. The relaxation operators provide users with alternative answers instead of an empty result.

**Efficient Query Segmentation.** In [43], the authors revisit the query segmentation problem to propose a novel technique for query segmentation that is easy to implement, fast, and accurate. The use n-gram frequencies and Wikipedia titles that are stored in a hash table. They finally evaluate the performance and accuracy of the segmentation process over 50000 human-annotated queries.

**Medical Information Retrieval.** In [64], the authors use query rewriting to address long medical queries submitted to search engines. Often users are uncertain about their exact questions and unfamiliar with medical terminology. Therefore, they normally submit long queries, describing their symptoms and situation in plain English, and this does not provide accurate and complete query answers. The authors propose a medical search engine that automatically rewrites long queries into moderate-length queries by selectively dropping unimportant terms (i.e., words).

**Query Rewriting For E-Commerce.** In [91], the authors propose a query rewriting technique that targets search engine queries for E-Commerce. Particularly, the authors are interested in queries that return null results. Therefore, they adopt a relaxation strategy for their query rewriting proposal. Their proposal ensures that the retrieved items are as close to the user's original goal as possible.

**Query Rewriting For SQL.** In [29], the authors propose a cooperative rewriting of SQL queries for database management systems. Their solution, denoted CSQL, consists of query relaxation, generalization, specialization, and association on data patterns. Query relaxation can be explicitly specified by the user or implicitly performed by the system. The implicit and explicit relaxations can also be combined and performed interactively by both the system and the user.

**Cooperative Query Relaxation.** In [38], the authors propose a cooperative process for query rewriting. First, they rely on relaxation to automatically (i) generalize query terms; (ii) modify query terms; and/or (iii) break restrictive joins. Then, the authors model the user preferences as rules/constraints and adjust the rewriting based on the latter.

**Generalized & Interactive Selection.** In [49], the authors address the issue of selection in interactive applications where users select items of interest on graphical interfaces. Then, the authors motivate the need for generalized selections (e.g., regions, or attribute ranges instead of individual items). They propose an interactive selection process that relies on query relaxation to achieve their objectives. Finally, they apply their method for information visualization, and graphics editing applications while enabling generalized selection over both static and dynamic interface objects.

### 4.3.4 Handling Connected Environment Dynamicity

Since query rewriting is used for different purposes, and various techniques exist, we highlight here our requirements and objectives in order to put this background study into our context (i.e., handling the dynamicity of a connected environment). Our objective is to rewrite (event related) obsolete queries. Due to the dynamicity of the environment, more precisely due to sensor mobility, breakdowns, and loss of data, a query might lose the sensors and data that it relies on. This restricts the query answer (i.e., either the query is not processed due to missing elements, or no results are found). Therefore, we look at rewriting techniques that increase recall in order to avoid null answers when querying for events in a connected environment.

We propose a solution that is inspired from query expansion and relaxation in order to replace missing sensors (i.e., data sources) and sensor observations (i.e., data/features) in obsolete queries. First, we aim to replace (substitute) unavailable sensors by similar and available ones. It is an exact rewriting (a straightforward substitution of sensors) that does not expand nor relax the query by respectively adding or retracting specific terms/keywords. Then, if the algorithm is incapable of finding adequate substitutes for specific sensors, we propose an approximate rewriting that substitutes unavailable sensed data/features. For this phase, we use semantic similarity measures, similar to semantic analysis in query relaxation, in order to find semantic "synonyms" (similar to query expansion) for obsolete data/features.

## 4.4 Preliminaries & Definitions

Before detailing our proposal for query rewriting, we provide here some preliminaries and formal definitions of key terms. We do not formalize every element of the environment, and limit the definitions to terms that are related to the connected environment, event queries, sensors, and sensor data.

### 4.4.1 The Connected Environment

Various components $c$ exist in a connected environment $CE$. These components could be related to the environment itself, the sensor network, the events, or the application domain. Definition 1 defines a connected environment $CE$.

**Definition 1.** *A **Connected Environment**, denoted $CE$, is defined as the set of its constituent components:*

$$CE = \bigcup_{i=0}^{n} c_i \; \forall i \in \mathbb{N} \tag{4.1}$$

*Where:*

- $c_i$ *is a component instance that belongs to CE*

*Remark.* Component instances could be related to HSSN concepts (e.g., `hssn:Infrastructure`, `hssn:MobileSensor`, `hssn:StaticSensor`, `hssn:ScalarProperty`). Moreover, instances could be related to event and application domain components. ■

### 4.4.2 Sensors & Sensor Data

Sensors are deployed at specific locations in the environment. Regardless of their type (e.g., static, mobile), each sensor has a location, a set of properties, and covers a specific area of the space. Moreover, sensors are the data providers that produce observations related to various event features. They do so by calling and executing specific functions. Definition 2 details the sensor description.

**Definition 2.** *A **Sensor** s is defined as a 7 tuple:*

$$s = (id, F, O, L, C, FUNCT, PROP) \tag{4.2}$$

*Where:*

- *id is a unique identifier*
- $F = \bigcup_{i=0}^{k} f_i \ \forall i \in \mathbb{N}$ *is a set of features observed by s*
- $O = \bigcup_{i=0}^{l} o_i \ \forall i \in \mathbb{N}$ *is a set of observations produced by s*
- $L = \bigcup_{i=0}^{m} previousLocation_i \cup currentLocation \ \forall i \in \mathbb{N}$ *details the location history of s*
- $C = \bigcup_{i=0}^{n} previousCoverage_i \cup currentCoverage \ \forall i \in \mathbb{N}$ *details the coverage area history of s*
- $FUNCT = \{isActive, sensingFunction\}$ *is the set of functions provided by s*
- $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$ *is the set of properties that describe the state of s*

*Remark.* The function *isActive* indicates ($\{TRUE, FALSE\}$) if the sensor is operational. The *sensingFunction* is the process that allows the sensor to make observations for a specific feature $f \in F$. The properties $p \in PROP$ indicate the remaining/free battery, processor, and memory percentages. Finally, the *networkStatus* denotes ($\{YES, NO\}$) the sensor's capability to transmit/receive data. ■

### 4.4.3 Event Queries

Event queries are used in *CE* in order to define, and later detect, targeted events. Since these queries might become obsolete due to the dynamicity of the environment, we formally define them in Definition 3 before addressing their rewriting.

**Definition 3.** *An **Event Query** q is defined as follows:*

$$q = \langle id, context \rangle \tag{4.3}$$

*Where:*

- *id is a unique identifier*

- *context* $= (F, C, S)$ *is the event query context where:*
  - $F = \bigcup_{i=0}^{n} f_i \; \forall i \in \mathbb{N}$ *is the set of features that describe the event*
  - $C = \bigcup_{i=0}^{m} c_i \;\; \forall i \in \mathbb{N}$ *is the set of conditions associated to the event features where* $\forall\, i \in \mathbb{N},\; c \in C,\; f \in q.F: \quad c_i \longrightarrow f_i$
  - $S = \bigcup_{i=0}^{l} s_i \;\; \forall i \in \mathbb{N}$ *is the set of sensors that provide observations for the detection of the event defined in q where:*

$$\forall\, s \in S \quad \exists\, f \in s.F \quad | \quad f \in q.F$$

*Remark.* In an event query, each sensor $s \in S$ is considered to be already known and defined (cf. Definition 2). Moreover, each sensor provides observations $o \in s.O$ that are related to at least one event describing feature $f \in q.F$. This means that the features provided by the sensor should be included in the ones required by the query: $\forall\, s \in S,\; s.F \subseteq q.F$. ■

### 4.4.4 Obsolete Event Queries

Obsolete event queries are defined as event queries that have suffered from sensor mobility and/or breakdowns (and thereafter loss of data for event features). We formally define them as follows:

**Definition 4.** *An **Obsolete Event Query** oq is defined as an **Event Query** q (cf. Definition 3)*

$$oq = \langle id, context \rangle \tag{4.4}$$

*Where at least one of the following conditions is satisfied:*

- *Condition 1:* $\exists\, s \in S \mid s.isActive = FALSE$
- *Condition 2:* $\exists\, s \in S \mid distance(s.L.currentLocation, oq.context.F.location) < \tau$
- *Condition 3:* $\exists\, s \in S \mid oq.context.F.location \notin s.C.currentCoverage$

*Remark.* Condition 1 checks if any sensor is inactive. Condition 2 checks if the distance in meters (using the *distance* function) between the current sensor location and the location requested in the query exceeds an acceptable threshold $\tau$. Condition 3 checks if any sensor no longer covers the location requested in the query context. ■

## 4.5 Query Re-writing Proposal

In this section, we present a query optimizer that allows query rewriting in order to help EQL-CE cope with the dynamicity of the connected environment. We start by detailing the optimizer module, its interaction with the entire framework, and its inner composition. Then, we detail our proposed algorithm.

### 4.5.1 Query Optimizer Module

Our global framework (cf. Figure 1.5) shows how the event query language is used to define and manage a connected environment. The queries are composed, parsed and sent for execution. However, in some cases sensor mobility/breakdown could lead to missing or mismatching data. Therefore, some queries become obsolete. The query optimizer aims to detect and rewrite such queries to ensure that the user does

not end up with null results when querying the connected environment for events. We envision two main rewriting strategies: (i) for single queries submitted by a user; and (ii) for batch queries that automatically run periodically. We detail here the process for the first case (i.e., rewriting a submitted query $q$ that became obsolete). The

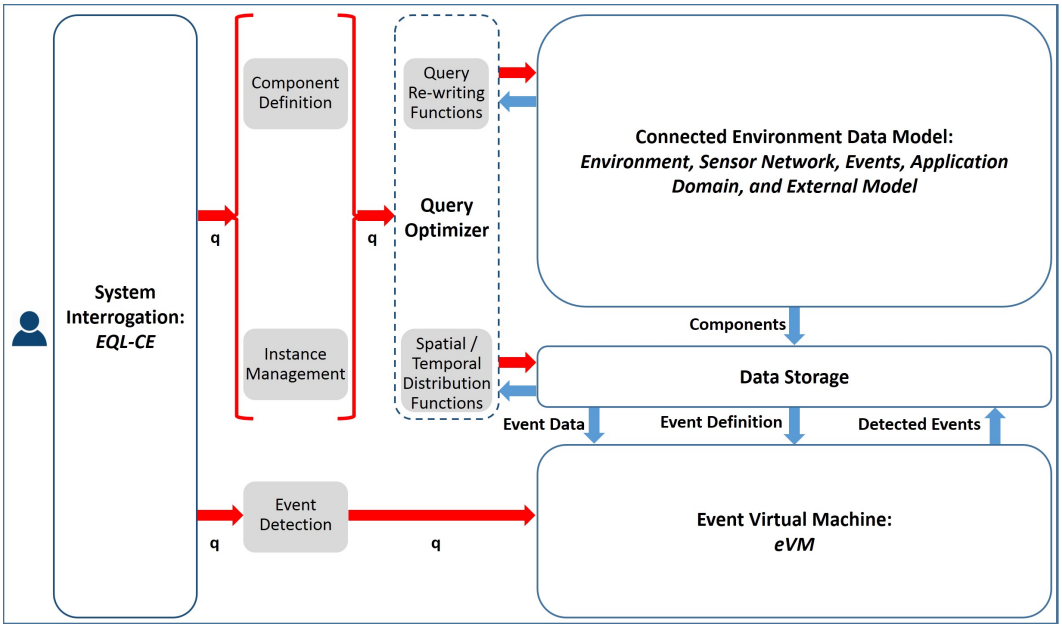


FIGURE 4.7: EDCE Global Framework

query optimizer is an extensible module, that could integrate various optimization engines. Currently, it is equipped with a query rewriting engine and various spatial/temporal distribution functions that could be integrated in EQL-CE queries (cf. Needs 4 and 5 in Section 3.2). Future additions to the query optimizer could include a query performance booster for improved execution run-time and resource consumption (to be discussed in a separate dedicated work). We currently provide three rewriting options for the user:

- Manual Rewriting: The user can rewrite a specific query using the ALTER or UPDATE queries provided by EQL-CE. Although this option is available, it is not recommended since the user is normally unaware of latest state of the connected environment (i.e., the user will rewrite the query without taking into account the latest changes that occurred in the environment).

- Collaborative Rewriting: The user can rewrite a specific query using the ALTER or UPDATE queries provided by EQL-CE with an assistance provided by the system. This entails guiding the user when changing a query by notifying him/her of the current state of the connected environment. Therefore, the user will receive a list of the currently available components that he/she can use when rewriting the query.

- Automatic Rewriting: The query rewriting engine automatically handles the detection and rewriting of obsolete queries. No user intervention is required.

In this study, we focus on the rewriting engine (automatic rewriting). We present first the query analyzer (i.e., the module that automatically detects obsolete event queries). Then, we detail the query re-writer (i.e., the module that executes our proposed algorithm for query rewriting). Figure 4.8 illustrates the inner composition of the query rewriting engine found in the optimizer module.

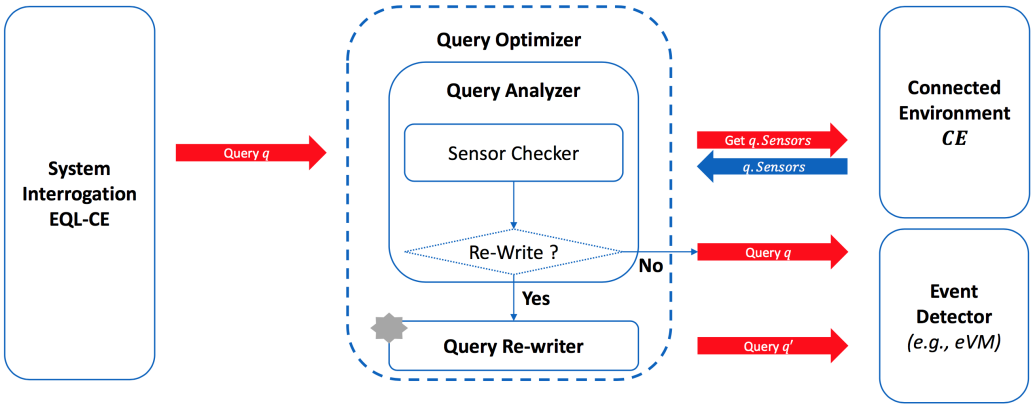


FIGURE 4.8: Query Rewriting Engine

### 4.5.2 Query Analyzer

The query analyzer takes as input any incoming event query $q$ as defined in Section 4.4. These queries could be either submitted individually by the user, or batch queries that are automatically sent for execution periodically. The analyzer checks the current state of the sensors that any incoming query $q$ relies on. The sensor checker queries the connected environment in order to retrieve the state of the sensors found in $q$ (this step can be executed once for a single query, or periodically for batch queries). If all sensors are still currently available, no query rewriting is needed and the query is sent for execution (i.e., the event definition can be used as described in $q$ by the event detector in order to detect the targeted event). However, if any sensor is no longer available (e.g., currently broken, unavailable, changed location, left the network) then the event query $q$ is sent to the query re-writer in order to replace its missing/unavailable data and sensors. Algorithm 1 details how the query analyzer detects obsolete queries. The *check* function takes the query $q$ and the connected environment $CE$ as input. It verifies the current state of all sensors found in the event query. For each of the aforementioned sensors, the algorithm checks if the sensor is running, and available by calling its *isActive* function (cf. Definition 2 in Section 4.4). The algorithm also compares the sensor's location in the query $q$ with its actual current location in $CE$ by calculating the distance in meters between the two locations and verifying against a system configured threshold $\tau$ (line 9). Finally, the algorithm compares the sensor's coverage area in the query $q$ with its current coverage area in $CE$ by measuring the overlap percentage between the two areas and verifying against a threshold $n$. This threshold is a predefined system parameter that could be later modified if necessary. A sensor is saved in the unavailable sensors list $US$ if it is currently inactive, too far from the location mentioned in $q$, or does not currently cover an acceptable percentage of the area mentioned in $q$. At the end of the checking phase, if $US$ is empty no rewriting is needed. If not, the query $q$ and the list $US$ are sent to the re-writer.

**Algorithm 1:** *check* Method - Query Analyzer

```
   /* Begin Input / Output declaration                                          */
   Input  : q, CE              // q is an event query, CE is a connected environment
   Output: US                        // US is the list of unavailable sensors in q
   /* Begin system parameters declaration                                       */
   Parameters: τ, n /* τ is a proximity threshold, n is an acceptable overlapping
      percentage                                                                */
   /* Begin Variable declaration                                               */
   Variables  : s, queryL, currentL, queryCA, currentCA, c1, c2, c3
   /* Begin algorithm                                                          */
1  foreach sensor ∈ q.S do
2      queryL ← s.L.currentLocation
3      currentL ← CE.S.s.L.currentLocation
4      queryCA ← s.C.currentCoverageArea
5      currentCA ← CE.S.s.C.currentCoverageArea
6      c1 ← s.FUNCT.isActive                                  // Get the state of s
7      c2 ← distance(queryL, currentL)        // Distance in meters between two locations
8      c3 ← over(queryCA, currentCA)            // Overlap percentage between two areas
9      if (c1 == False  |  c2 > τ  |  c3 < n) then
           /* If s is currently inactive, no longer located in the same location (or
              nearby), or no longer covers (mostly) the same area               */
10         US ← s
11 end
12 Return US
```

### 4.5.3 Query Re-writer

Once the query analyzer confirms that an event query $q$ needs rewriting, the re-writer takes the query $q$, the connected environment $CE$, and the list of unavailable sensors $US$ as input in order to output a rewritten query $q'$. The query rewriting process is described in Algorithm 2 which is split into two main parts: (i) exact rewriting (lines 1-12); and (ii) approximate rewriting (lines 13-29). The following details each part separately:

- Exact Rewriting: replacing unavailable sensors. First, the algorithm attempts to replace each missing sensor in $q$ by another that is similar, and compatible with the query context. Since this does not alter the definition or meaning of the event, we denote this part of the algorithm 'exact rewriting'. We call the replacing sensor a 'substitute'. To achieve this, the *getSimilarSensor* function (line 2) compares the missing sensor with others that could replace it (candidates). Using our proposed sensor similarity measure, the algorithm elects the successor. If no candidate is 'similar enough' (i.e., the similarity measure does not exceed an acceptance threshold), the original sensor is flagged as 'irreplaceable' and stored in the $IS$ list (dedicated to 'irreplaceable' sensors). At the end of this phase, if $IS$ is empty (i.e., all unavailable sensors were replaced) the algorithm breaks. Otherwise, the algorithm goes into phase two (described below).

- Approximate Rewriting: replacing missing features. If the query $q$ contains at least one 'irreplaceable' sensors, the algorithm goes into its second phase: feature replacement. Here, the algorithm attempts to replace the features provided by the 'irreplaceable' sensors with others that are available and similar. To do so, the features that are no longer covered by any sensor are stored in the missing features list $MF$, and each missing feature is replaced by another similar one using the *getSimilarFeature* function (line 20). We denote this phase

as approximate rewriting since the substitute features are not exact matches of the original ones. Finally, if no adequate replacement if found for a specific feature, the latter is flagged as 'irreplaceable'. The rewritten query $q'$ is outputted.

**Algorithm 2:** Event Query Re-Writing

```
   /* Begin Input / Output declaration                                    */
   Input  : q, CE, US /* q is an event query, CE is a connected environment, US is
      the list of unavailable sensors in q                                */
   Output: q'                                   // q' is the rewritten event query
   /* Begin Variable declaration                                          */
   Variables: IS, FEATURES, MF, IF, sensor, sensor', f, feature, feature'
   /* Begin algorithm                                                     */
1  foreach sensor ∈ US do
2      sensor' ← getSimilarSensor(sensor, q.context, CE)
3      if (sensor' != EMPTY) then
4          q.S.sensor ← sensor'
5      else
6          IS ← sensor                          // List of irreplaceable sensors
7      end
8  end
9  if (IS.size == 0) then
10     q' ← q
11     break
12 else
13     foreach sensor ∈ IS do
14         foreach f ∈ sensor.F do
15             FEATURES ← f /* List of all features provided by the missing
                  sensors                                                 */
16         end
17     end
18     MF = getMissingFeatures(q.context.F, FEATURES)
       /* List of all missing features, i.e., no longer covered by any sensor  */
19     foreach feature ∈ MF do
20         feature' ← getSimilarFeature(feature, q.context, CE)
21     end
22     if (feature' != EMPTY) then
23         q.context.F.feature ← feature'
24     else
25         IF ← feature                         // List of irreplaceable features
26     end
27 end
28 q' ← q
29 Return q'
```

#### 4.5.3.1 Measuring Sensor Similarity

In order to find the successor of a missing or unavailable sensor, one should be able to measure inter-sensor similarity. Therefore, we propose a sensor similarity measure (cf. Algorithm 3) that takes into account four main sensor attributes:

- Capability: Denotes the successor sensor's ability to provide or sense the same required observable properties (fully or partially) as the original sensor.
- Reliability: Denotes the successor sensor's ability to execute the required task (i.e., having a good battery status, memory, processor).

- Spatial closeness: Denotes the presence of the successor sensor close to the location required for the sensing task (i.e., close to the original sensor's location), and its ability to cover a 'similar-enough' area of space.
- Temporal closeness: Denotes the successor sensor's capability to produce fresh (recent) observations in comparison with the original sensor.

**Algorithm 3:** *getSimilarSensor* Method (cf. Algorithm 2 - Line 2)

```
   /* Begin Input / Output declaration                                    */
   Input  : s, q.context, CE /* s is a sensor, q.context is the query context, CE is a
      connected environment                                               */
   Output: simS                                  // simS is the most similar sensor to s
   /* Begin System Parameters declaration                                 */
   Parameters: w1, w3, w3, w4                    // Weights assigned to each score
   /* Begin Variable declaration                                          */
   Variables  : sim, cap, rel, spa, tem, i, j, S, ID, SIM, m, s'
   /* Begin algorithm                                                     */
1  S ← getCompatibleSensors(q.context, CE) /* Retrieves all sensors from CE that comply
      with the query context                                              */
2  i = 0
3  foreach s' ∈ S do
4      cap ← capabilityScore(s, s')
5      rel ← reliabilityScore(s')
6      spa ← spatialScore(s, s')
7      tem ← temporalScore(s, s')
8      sim = w1 × cap + w2 × rel + w3 × spa + w4 × tem
9      ID[i] ← s'.ID
10     SIM[i] ← sim
11     i++
12 end
13 m = MAX(SIM)
14 if (m == 0) then
15     return EMPTY
16     break
17 else
18     j ← getPosition(m) /* getPosition is the is the function that return the first
          position for a value in a list                                  */
19
20     simS ← ID[j]
21     return simS;
22 end
```

Algorithm 3 details the sensor similarity measure function that takes the missing sensor, the query context, and the connected environment as input and outputs the most similar sensor (i.e., a successor if found). To do so, we rely on a sensor similarity measure in order to select the most similar sensor. To avoid comparing the original sensor with all others found in $CE$, we filter the candidates list to sensors that comply with the query context using the *getCompatibleSensors* function (line 1). This boosts the performance by minimizing the number of sensor comparisons. The process continues by calculating, for each compatible sensor, an individual score for each aforementioned attribute (lines 4-7). Then, we calculate an overall similarity score (line 8) as the weighted sum of all individual scores. The weights are system parameters that allow emphasis on a specific attribute (or set of attributes) when calculating the similarity score. For instance, if any temperature observation is needed regardless of any spatial restrictions, one might decrease the weight associated to spatial closeness ($w_3 = 0$) and increase the one related to the sensor's capability ($w_1 = 1$). Each compatible candidate sensor will have an overall similarity score w.r.t the original (missing/unavailable) sensor. The process ends by choosing the

most similar sensor (i.e., having the maximal similarity score). If the maximal score is zero, then no similar sensors were found. In this case the returned output is empty. This means that the original sensor is 'irreplaceable'. In the following, we detail the five functions used in Algorithm 3. We start with the *getCompatibleSensors* function (Algorithm 4).

Since it is meaningless, and costly to compare the original sensor to others that are not compatible with the query requirements (e.g., too far away, do not cover the required area), a filtering process is required prior to similarity comparison. This function takes the query context as input, focuses on the spatial feature, and filters out any sensor that is not close enough to the requested location in $q$ or that does not cover this location. The output is a list of candidate sensors that comply with the query context. Only this list is considered in the similarity calculations when searching for a successor for an unavailable sensor.

**Algorithm 4:** *getCompatibleSensors* Method (cf. Algorithm 3 - Line 1)

```
   /* Begin Input / Output declaration                                   */
   Input  : q.context, CE /* q.context is the query context, and CE is the connected
      environment                                                        */
   Output: S                                  // S is the set of compatible sensors
   /* Begin System Parameters declaration                                */
   Parameters: τ
   /* Begin Variable declaration                                         */
   Variables  : loc, s
   /* Begin algorithm                                                    */
1  loc ← q.context.F.location
2  foreach s ∈ CE.S do
3  |  if (distance(s.L.currentLocation, loc) < τ  ||  loc ∈ s.C.currentCoverage) then
4  |  |  S ← s
5  end
6  return S
```

Algorithm 5 details how the capability similarity score is calculated. The original sensor and a candidate (compatible) sensor are compared. The set of features observed by each sensor are analyzed in order to count the number of common features (line 2). If this number exceeds an acceptable threshold $n$, then the score is given a specific *value* (e.g., 1). If not enough common features are found, the attributed score is the *value* divided by a *degradingFactor*. The acceptable number of common features $n$, the optimal score *value*, and the factor of degradation of the score *degradingFactor* are configurable system parameters.

**Algorithm 5:** *capabilityScore* Method (cf. Algorithm 3 - Line 4)

```
   /* Begin Input / Output declaration                                   */
   Input  : s, s'                                    // s, s' are two a sensor
   Output: score                          // score is the capability similarity score
   /* Begin System Parameters declaration                                */
   Parameters: n, value, degradingFactor
   /* Begin algorithm                                                    */
1  score = 0
2  if (commonFeaturesCount(s.F, s'.F) > n) then
3  |  score = value
4  else
5  |  score = value / degradingFactor
6  end
7  return score
```

Algorithm 6 details how the reliability similarity score is calculated. The candidate (compatible) sensor's properties are analyzed. Each property (e.g., battery, memory, processor) is verified against an acceptance threshold $\tau$ (line 4). Then, the property is given an individual reliability score of *value* if the property level is accepted (line 5). If not, *value* is divided by a *degradingFactor* (line 7). The final reliability score for a candidate sensor is the average of all individual property scores (line 10).

**Algorithm 6:** *reliabilityScore* Method (cf. Algorithm 3 - Line 5)

```
   /* Begin Input / Output declaration                                    */
   Input  : s'                                          // s' is a sensor
   Output: score                    // score is the availability similarity score
   /* Begin System Parameters declaration                                 */
   Parameters: T, value, degradingFactor
   /* The acceptable threshold τ ∈ T in respect to each property p ∈ PROP, the
      optimal score value, and the factor of degradation of the score degradingFactor
      are system parameters that the admin can configure                  */
   /* Begin Variables declaration                                         */
   Variables  : τ, p
   /* Begin algorithm                                                     */
1  score = 0
2  foreach p ∈ s'.PROP do
3      τ ← getRelatedThreshold(T) /* Gets the threshold value related to the current
          property                                                         */
4      if (p > τ) then
5          score = score + value
6      else
7          score = score + value / degradingFactor
8      end
9  end
10 return score / ||s'.PROP||
```

Algorithm 7 details how the spatial similarity score is calculated. The function takes the original and candidate sensors as input. It compares their locations and coverage areas. Location closeness is evaluated based on the metric distance between the two sensors (line 6). Coverage area similarity is evaluated based on the overlap percentage of the two coverage areas (line 7). Finally, spatial similarity is given a score of *value* if the distance between the two sensors is less than a threshold $\tau$ and the overlap percentage between the two coverage areas is greater than a threshold $p$ (line 11). If the distance is greater than $\tau$ and the overlap percentage is less than $p$ (line 14), the spatial similarity score is zero. Otherwise, the score is *value* divided by a *degradingFactor*. The acceptable distance in meters $\tau$, the percentage of overlapping $p$, the optimal score *value*, and the factor of degradation *degradingFactor* are system parameters.

Algorithm 8 details how the temporal similarity score is calculated. The algorithm takes as input the original, and candidate sensors. Then the *getLatest* function retrieves the most recent observation produced by each sensor (line 2 and 3). Finally, the $t_{dist}$ function returns the temporal distance in seconds between two sensor observations. If the temporal distance is less than a threshold $\tau$, the score is assigned a *value* (line 6). If not, the temporal similarity score is *value* divided by a *degradingFactor*. The acceptable temporal distance in seconds $\tau$, the optimal score

*value*, and the score degradation factor, *degradingFactor*, are system parameters.

**Algorithm 7:** *spatialScore* Method (cf. Algorithm 3 - Line 6)

```
   /* Begin Input / Output declaration                                   */
   Input  : s, s'                                   // s,s' are two sensors
   Output: score                          // score is the spatial similarity score
   /* Begin System Parameters declaration                                */
   Parameters: τ, p, value, degradingFactor
   /* Begin Variable declaration                                         */
   Variables  : cl1, cl2, cc1, cc2, d, o
   /* Begin algorithm                                                    */
1  score = 0
2  cl1 = s'.L.currentLocation
3  cl2 = s.L.currentLocation
4  cc1 = s'.C.currentCoverage
5  cc2 = s.C.currentCoverage
6  d = dist(cl1, cl2)          // dist returns the distance in meters between two locations
7  o = over(cc1, cc2)             // over returns the overlap percentage between two areas
8  if (d < τ  &&  o > p) then
9  |  score = value
10 else
11 |  if ((d < τ  &&  o < p)  ||  (d > τ  &&  o > p)) then
12 |  |  score = value / degradingFactor
13 |  else
14 |  |  score = 0
15 |  end
16 end
17 return score
```

**Algorithm 8:** *temporalScore* Method (cf. Algorithm 3 - Line 7)

```
   /* Begin Input / Output declaration                                   */
   Input  : s, s'                                   // s,s' are two sensors
   Output: score                       // score is the temporal similarity score
   /* Begin System Parameters declaration                                */
   Parameters: τ, value, degradingFactor
   /* Begin Variable declaration                                         */
   Variables  : o1, o2
   /* Begin algorithm                                                    */
1  score = 0
2  o1 = getLatest(s'.O.o)
3  o2 = getLatest(s.O.o)
4  if (t_dist(o1, o2) < τ) then
5  |  score = value
6  else
7  |  score = value / degradingFactor
8  end
9  return score
```

#### 4.5.3.2 Measuring Feature Similarity

When replacing an unavailable sensor, one might end up with no adequate successor. In such cases, the sensor originally used in the event query $q$ is flagged as irreplaceable (cf. Algorithm 2 - line 6). However, the aforementioned irreplaceable sensors provide observations for specific event features (e.g., temperature, humidity, noise). Hence, the data needed for the detection of the event is still lacking. To address this issue, the algorithm attempts to replace the feature itself by another similar one (e.g., humidity instead of temperature in case the temperature sensor is

irreplaceable). This requires comparing features by measuring their semantic similarity in order to choose a successor feature.

**Algorithm 9:** *getSimilarFeature* Method (cf. Algorithm 2 - Line 20)

```
/* Begin Input / Output declaration                                    */
Input  : f, CE, q.context /* f is a feature, CE is the connected environment, and
   q.context is the query context                                      */
Output: f'                                        // f' is the most similar feature
/* Begin System Parameters declaration                                 */
Parameters: KB, ssf, τ
/* The knowledge base having various features KB, the semantic similarity
   function ssf, and the acceptable threshold τ are system parameters that the
   admin can configure                                                 */
/* Begin Variable declaration                                          */
Variables  : FEATURES, feature, score, max, id, loc, S, s
/* Begin algorithm                                                     */
1  S ← getCompatibleSensors(q.context, CE) /* Retrieves all sensors from CE that comply
      with the query context                                           */
2  foreach s ∈ S do
3  |   FEATURES ← s.F /* Retrieves all features from CE that are provided by
   |      sensors from S                                              */
4  end
5  max = 0
6  foreach feature ∈ FEATURES do
7  |   score = ssf(feature, f, KB)
8  |   if (score > max) then
9  |   |   max = score
10 |   |   id = feature.ID
11 end
12 if (max > τ) then
13 |   f' ← getFeatureByID(id)                          // Retrieves the successor feature
14 |   q.Context.C ← adaptCondition(f, f') /* Adjusts the conditions related to the
   |      successor feature                                           */
15 |   return f'
16 else
17 |   return EMPTY
18 end
```

Algorithm 9 describes how our rewriting algorithm compares/substitutes features. The *getSimilarFeature* function takes the query context, the connected environment, and a feature (the lacking feature $f$) as input. The aim is to output a successor feature $f'$. To do so, we rely on a sensor feature/observation knowledge base $KB$ where features, and inter feature ties are semantically described (e.g., an ontology). We also call an external semantic similarity function $ssf$ (e.g., such as the ones in [86]). We choose an external function that calculates the semantic similarity between a feature on one hand and 1 to n combined features on the other (e.g., noise combined with vibration could replace motion). Moreover, $ssf$ considers the features and their dependencies within the knowledge base $KB$ when calculating the similarity score. The process of replacing a feature is the following:

1. Step 1: Filtering the replacing candidate features. It is highly costly to compute the similarity between a missing feature $f$ and all features found in $KB$. To alleviate this issue, we call the *getCompatibleSensors* function that returns the set of sensors that currently comply with the query context (cf. Algorithm 3). These sensors are the only ones capable of providing adequate candidate

features ($s.F$ cf. Definition 2). Line 3 shows how all candidate features (provided by compatible sensors) are stored in a list $FEATURES$. These features will be compared with the original missing feature $f$.

2. Step 2: Choosing the most similar (successor) feature. For each candidate feature found in $FEATURES$, the $ssf$ function will calculate the semantic similarity score by considering the features and their dependencies within the knowledge base $KB$ (lines 6-11).

3. Step 3: Verifying against a similarity threshold. Before outputting the successor feature, we compare its similarity score (which is the maximum value) with a similarity threshold $\tau$. If the score exceeds $\tau$, the successor feature $f'$ is outputted. If not, the function return an empty result. This means that the original feature $f$ is 'irreplaceable'.

In the rare case, of rewriting a query without being able to replace all unavailable sensors, and all missing features, the query cannot have a deterministic answer. We detail probabilistic and fuzzy query answers in a separate future work.

### 4.5.4 Complexity Evaluation

Since the implementation is still ongoing, we cannot yet conduct experiments. Meanwhile, we present here an a 'priori' evaluation of the complexity of Algorithms 1 and 2. As the name suggests, we analyze the algorithms prior to running them on a specific system. Therefore, this is a theoretical analysis. The efficiency of the algorithms is measured under the assumption that all other factors, (e.g., processor speed) are constant and have no effect on the implementation. Complexity analysis is performed on two parameters: (i) time complexity gives an indication as to how long an algorithm takes to complete with respect to the input size; and (ii) space complexity gives an indication as to how much memory is required to execute the algorithm with respect to the input size. We calculate here how the time (or space) taken by the algorithms increases as we augment the input size. We rely on 'BigO' (O) notation to present an overview of the worst case scenario (upper bound), and the 'Big Omega' ($\Omega$) notation to measure the best case scenario (lower bound). We recall here each notation:

- BigO: If an algorithm is described by a function $f(n)$, the BigO of $f(n)$ is a function $g(n)$ that bounds it (i.e., after a certain value $g(n)$ would always exceed $f(n)$. The common notations of the BigO are the following: (i) $O(1)$ describes an algorithm that will always execute in the same time (or space) regardless of the size of the input; (ii) $O(n)$ describes an algorithm whose performance grows linearly and in direct proportion to the size of the input; (iii) $O(n^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input. This is common with algorithms that involve nested iterations (deeper nested iterations will result in $O(n^3)$, $O(n^4)$, etc); and (iv) $O(2^n)$ denotes an algorithm whose growth doubles with each addition to the input. The growth curve of an $O(2^n)$ function is exponential.

- Big$\Omega$: This notation provides us with the best case scenario of running an algorithm (i.e., the minimum amount of resources (time or space) an algorithm would take to run). If an algorithm is described by a function $f(n)$. Big$\Omega$ of $f(n)$ is a function $g(n)$ that bounds the lower end of $f(n)$ (i.e., after a certain value $f(n)$ would always exceed $g(n)$).

**Algorithm 1 Evaluation.** The *check* method inputs an array of query sensors. The purpose is to check the availability of each one of them. The input size is $N = \|q.S\|$ (cf. Definition 3). The algorithm begins with a $foreach$ loop that scrolls through the array. For each iteration, the availability of a sensor is evaluated based on its current state in the connected environment $CE$. Besides variable/value assignments, two functions are called in the loop: (i) *distance* that returns the distance between two locations (line 7); and (ii) *over* (line 8) that returns the overlap between two coverage areas without having to run nested loops (e.g., using hash maps, spatial algebra operators). All statements in the loop are executed once (i.e., of complexity $O(1)$).

Therefore, in a worst case scenario the BigO of Algorithm 1 is $O(N)$. In contrast, the best case scenario is when $N = \|q.S\| = 1$. This means that the input query only requires one sensor. In this case, the Big$\Omega$ of Algorithm 1 is $\Omega(1)$. This shows that, in theory, the complexity in time (or space) of Algorithm 1 is linear and directly proportional to the input size.

**Algorithm 2 Evaluation.** Algorithm 2 is more complex to evaluate since it nests various functions (e.g., *getSimilarSensor*, *getCompatibleSensors*, *getSimilarFeature*). Moreover, the overall complexity is affected by the external semantic similarity function $ssf$ and the knowledge base $KB$ used in the *getSimilarFeature* function. The algorithm is a sequence of two main parts (i.e., sensor replacement, feature replacement). When all unavailable query sensors are replaceable, the algorithm breaks and the second part is not executed.

The sensor replacement process is detailed in a $foreach$ loop (lines 1-11) that scrolls through a list of all the unavailable sensors in order to replace them. The size of this list is $N = \|US\|$. We start by detailing the nesting hierarchy of this loop.

- The $foreach$ loop calls the *getSimilarSensor* function (Algorithm 2 - line 2) $N$ times ($N = \|US\|$). The *getSimilarSensor* function calls the functions/loops below:
  - The *getCompatibleSensors* function contains a $foreach$ loop that iterates $M$ times ($M = \|CE.S\|$) as shown in Algorithm 4 - line 2.
  - A $foreach$ loop (Algorithm 3 - lines 3-12) that iterates $L$ times ($L = \|S\|$).

The remaining statements within this loop have constant complexities, which are negligible in comparison to the complexity brought by the two nested loops. Therefore, the overall complexity of the first part of Algorithm 2 (i.e., sensor replacement) is $O(N \times (M + L))$.

The feature replacement process is represented in lines 12-29 of Algorithm 2. We detail the different parts of this process and the subsequent loops:

- Lines 12-18: Here we implement two nested loops to find the features that need replacement. We scroll through the list of irreplaceable sensors in the upper level loop ($P = \|IS\|$ iterations). Then, for each sensor we retrieve the provided features ($F$ iterations). Then in line 18 a the missing features are stored in an array list. The overall complexity of this part is $O(P \times F)$.

- Lines 19-21: Here we call the *getSimilarFeature* function to replace the missing features. This process's complexity is affected by the external $ssf$ function called in Algorithm 9 - line 7). Since the function might change, we denote its complexity as $O_{ssf}$. The overall complexity of this part can be summarized by the complexity brought by its deepest nested loop. Therefore, the complexity of lines 19-21 is $O(A \times (B + C + D \times O_{ssf}))$ where:
  - $A$ is the number of iteration of the $foreach$ loop in Algorithm 2 - line 19.
  - $B$ is the number of iterations of the $foreach$ loop (cf. *getCompatibleSensor* in Algorithm 9 - line 1).
  - $C$ is the number of iteration of the $foreach$ loop in Algorithm 9 - line 2.
  - $D$ is the number of iteration of the $foreach$ loop in Algorithm 9 - line 6.
  - $O_{ssf}$ is the complexity of the semantic similarity function (cf. Algorithm 9 - line 7).

To recapitulate, the sensor replacement part has a BigO notation of $O(N \times (M + L))$, and the feature replacement part has a BigO notation of $O(P \times F + A \times (B + C + D \times O_{ssf}))$. However, the overall complexity can be simplified by only considering the highest nested loops (since they make the others negligible). Therefore, we can say that the BigO notation of Algorithm 2 is $O(A \times D \times O_{ssf})$. This represents the worst case scenario. The best case scenario is when the algorithm breaks at line 11 (if all sensors are replaceable). In this case, the Big$\Omega$ notation is $\Omega(N \times (M + L))$.

**Discussion.** In summary, Algorithm 1 has a linear worst case scenario while Algorithm 2 has a polynomial worst case complexity of degree $n \mid n \geq 3$ depending on the complexity of the semantic similarity function $ssf$. However, event queries rely on a finite number of sensors/features (that in most cases is not excessive). Therefore, although Algorithm 2 has nested loops, the number of iterations should not be, in theory, very costly. To verify this, we need to run the previously mentioned experiments and compare the results as soon as the implementation is completed.

## 4.6 Illustration and Experimental Protocol

### 4.6.1 Illustration Example

In this section, we present an example that illustrates how the query rewriting engine works. We recall the same example of Section 4.2 (cf. Figure 4.9) and rewrite the *fire event* query in Shop 1. We detail the setup, how unavailable sensors are replaced, and how missing features are substituted. The second part of this section discusses the ongoing experiments.

#### 4.6.1.1 Example Setup

The connected environment $CE$ is the entire Smart Mall. However, the mall manager is interested in monitoring a *fire event* in Shop 1. The shop contains three static sensors ($s_1, s_2, s_3$), and four mobile devices ($MD_1$, $MD_2$, $MD_3$, $MD_4$) with various sensing capabilities. We detail the representation of the aforementioned sensors based on Definition 2 (cf. Section 4.4).

$$s = (id, F, O, L, C, FUNCT, PROP)$$

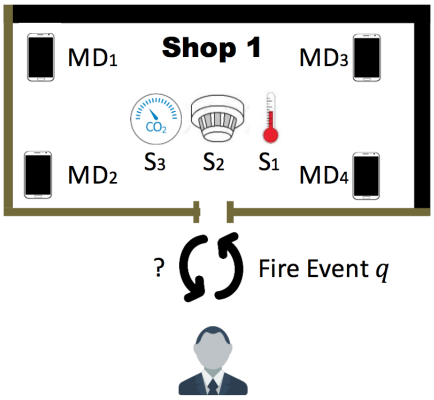


FIGURE 4.9: Rewriting the fire event in Shop 1

- The static sensor $s_1$ is defined with:
  - $id = 1$
  - $F = \{Temperature\}$
  - $O = \{o_1, o_2, o_3\}$
  - $L = \{(Shop_1, TI_1)\}$
  - $C = \{(Shop_1, TI_1)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$
- The static sensor $s_2$ is defined with:
  - $id = 2$
  - $F = \{Smoke\}$
  - $O = \{o_4, o_5, o_6\}$
  - $L = \{(Shop_1, TI_2)\}$
  - $C = \{(Shop_1, TI_2)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$
- The static sensor $s_3$ is defined with:
  - $id = 3$
  - $F = \{CO_2\}$
  - $O = \{o_7, o_8, o_9\}$
  - $L = \{(Shop_1, TI_3)\}$
  - $C = \{(Shop_1, TI_3)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$
- The mobile device $MD_1$ is equipped with the following temperature sensor:
  - $id = 4$
  - $F = \{Temperature\}$

- $O = \{o_{10}, o_{11}\}$
- $L = \{(CoffeShop, TI_4), (Shop_1, TI_5)\}$
- $C = \{(CoffeShop, TI_4), (Shop_1, TI_5)\}$
- $FUNCT = \{isActive, sensingFunction\}$
- $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

- The mobile device $MD_2$ is equipped with the following temperature, and $CO_2$ sensors:
  Temperature Sensor:
  - $id = 5$
  - $F = \{Temperature\}$
  - $O = \{o_{12}\}$
  - $L = \{(MovieTheater, TI_6), (Shop_1, TI_7)\}$
  - $C = \{(MovieTheater, TI_6), (Shop_1, TI_7)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

  $Co_2$ Sensor:
  - $id = 6$
  - $F = \{CO_2\}$
  - $O = \{o_{13}\}$
  - $L = \{(MovieTheater, TI_6), (Shop_1, TI_7)\}$
  - $C = \{(MovieTheater, TI_6), (Shop_1, TI_7)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

- The mobile device $MD_3$ is equipped with the following temperature, and smoke sensors:
  Temperature Sensor:
  - $id = 7$
  - $F = \{Temperature\}$
  - $O = \{o_{14}\}$
  - $L = \{(FoodCourt, TI_8), (Shop_1, TI_9)\}$
  - $C = \{(FoodCourt, TI_8), (Shop_1, TI_9)\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

  Smoke Sensor:
  - $id = 8$
  - $F = \{Smoke\}$
  - $O = \{o_{15}\}$
  - $L = \{(FoodCourt, TI_8), (Shop_1, TI_9)\}$
  - $C = \{(FoodCourt, TI_8), (Shop_1, TI_9)\}$

  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

- The mobile device $MD_4$ is equipped with the following temperature, and humidity sensors:
  Temperature Sensor:
  - $id = 9$
  - $F = \{Temperature\}$
  - $O = \{o_{16}\}$
  - $L = \{(Shop_1, TI_{10})\}$
  - $C = \{(Shop_1, TI_{10})\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

  Humidity Sensor:
  - $id = 10$
  - $F = \{Humidity\}$
  - $O = \{o_{17}\}$
  - $L = \{(Shop_1, TI_{10})\}$
  - $C = \{(Shop_1, TI_{10})\}$
  - $FUNCT = \{isActive, sensingFunction\}$
  - $PROP = \{batteryStatus, processorStatus, memoryStatus, networkStatus\}$

The mall manager defines the *fire event* query denoted $q$ as presented in Definition 3:

$$q = \langle q_{fire}, context_{fire} \rangle$$

Where:

- $q_{fire}$ is a unique identifier.
- $context = (F, C, S)$ is the event query context where:
  - $F = \{Time, Location, Temperature, CO_2, Smoke\}$ is the set of features that best describe the event.
  - $C = \{Any, Shop1, High, High, True\}$ is the set of conditions associated to the event features where: $Any \longrightarrow Time$, $Shop1 \longrightarrow Location$, $High \longrightarrow Temperature$, $High \longrightarrow CO_2$, $True \longrightarrow Smoke$.
  - $S = \{\emptyset, \emptyset, s_1, s_3, s_2\}$ is the set of sensors that provide observations for the detection of the event defined in $q$ where:

$$s_1.F = \{Temperature\}, \quad s_3.F = \{CO_2\}, \quad s_2.F = \{Smoke\}$$

The event definition relies on the static sensors. It needs to detect at anytime the *fire event* in Shop 1. In order to illustrate a query rewriting scenario, consider the following issues: (i) the temperature sensor $s_1$ and the smoke sensor $s_2$ broke down; and (ii) the mobile device $MD_3$ left the shop. The aforementioned issues render the query obsolete since it relies on sensors $s_1$, and $s_2$. In the following, we present how the query analyzer discovers the obsolete query $q$. Then, we detail how the query rewriting algorithm overcomes this issue by producing a new query $q'$.

#### 4.6.1.2 Discovering The Obsolete Query

When $q$ is submitted to the event detector, it is first analyzed in the query optimizer. The *check* method (cf. Algorithm 1), verifies if $q$ relies on any currently missing or unavailable sensors. In this case, the query analyzer will check $s_1$, $s_2$, and $s_3$ in order to make sure that each sensor is currently active (i.e., no breakdowns), and still located near/covering the location targeted by the query (i.e., Shop 1). All unavailable or missing sensors will be stored in the $US$ (unavailable sensors) list. When checking the *fire event* query, the temperature sensor $s_1$, and the smoke sensor $s_2$ will be flagged as unavailable (because their *isActive* property will return false due to the breakdown). Therefore, the query $q$ will be sent to the query re-writer to overcome the lack of sensors/data.

#### 4.6.1.3 Query Rewriting

The query re-writer attempts to replace the unavailable sensors with semantically similar ones (cf. Algorithm 2). For the *fire event* query, the algorithm will seek substitutes for sensors $s_1$ and $s_2$ using the *getSimilarSensor* function (cf. Algorithm 3).

**Replacing $s_1$.** This sensor provides temperature observations, therefore the capability score is the highest for all mobile devices (since they all provide temperature observations). However, since $MD_3$ left the shop, it has a lower spatial score than $MD_1$, $MD_2$, and $MD_4$. All other attributes (i.e., reliability and temporal closeness) are equal for the available three devices. Therefore, devices $MD_1$, $MD_2$, and $MD_4$ are chosen to replace the static temperature sensor $s_1$ (all three are assigned to the query for optimal spatial distribution).

**Replacing $s_2$.** This sensor provides smoke observations. Hence, the capability score would have been the highest for $MD_3$ (since it is the only device or sensor capable of providing smoke observations). However, the latter left the shop and no longer covers it. Therefore, all of the remaining sensors do not exceed the acceptable similarity threshold. As a result, $s_2$ is flagged as 'irreplaceable', and the rewriting algorithm will now attempt to replace the smoke feature with another similar and available one.

**Replacing The Smoke Feature.** When replacing a feature, the first step is to select the available sensors that comply with the query context (that could produce a substitute feature). The *getCompatibleSensors* function provides this list of sensors that could contribute in this process. In the case of our example, the compatible sensors are: $S = \{s_3, MD_1, MD_2, MD_4\}$. These sensors provide observations for the the following features: $F = \{CO_2$, Temperature, Humidity $\}$. In the following, we compare these features to smoke in order to find an adequate replacement. To do so, the *getSimilarFeature* method (cf. Algorithm 9) uses a semantic similarity function $ssf$ to compare the feature smoke to humidity, temperature, and $CO_2$ within a knowledge base $KB$. The similarity scores are based on the features and their dependencies. Also, the best substitute could be a feature or a combination of features. Once the substitute(s) is (are) chosen, the conditions (constraints related to it) are adapted and new sensor to feature mapping is established in the rewritten query $q'$. In this example, humidity could replace the feature smoke. Therefore, the rewritten

query $q'$ is defined as follows:

$$q' = \langle q_{fire}, context_{fire} \rangle$$

Where:

- $q_{fire}$ is a unique identifier.
- $context = (F', C', S')$ is the event query context where:
  - $F' = \{Time, Location, Temperature, CO_2, Humidity\}$ is the set of features that best describe the event.
  - $C' = \{Any, Shop1, High, High, Low\}$ is the set of conditions associated to the event features where: $Any \longrightarrow Time$, $Shop1 \longrightarrow Location$, $High \longrightarrow Temperature$, $High \longrightarrow CO_2$, $Low \longrightarrow Humidity$.
  - $S' = \{\emptyset, \emptyset, (MD_1, MD_2, MD_4), s_3, MD_4\}$ is the set of sensors that provide observations for the detection of the event defined in $q$ where:

$$MD_1.F = MD_2.F = \{Temperature\}, \quad s_3.F = \{CO_2\},$$
$$MD_4.F = \{Temperature, Humidity\}$$

### 4.6.2 Experimental Protocol

#### 4.6.2.1 Implementation

Before detailing the experimentation, we briefly present the ongoing implementation work. We are currently implementing the query optimizer. We are developing an online platform for event detection in connected environments[11]. The architecture of this web-based platform reflects the entire framework presented in this manuscript. Users will be able to define/manage their own connected environments, and detect targeted events of interest using the event query language for connected environments (EQL-CE). Their event queries will be sent to the query optimizer module prior to execution for query analysis and rewriting (if needed). Finally, an event detector will use the provided event queries (definitions) to find occurrences of the aforementioned events.

#### 4.6.2.2 Evaluation Objectives

The objectives of the experimentation are two-fold. First, we aim to evaluate the performance of Algorithms 1 and 2. This entails measuring the execution run-time, and the resource (RAM/CPU) consumption. The other objective, is accuracy evaluation. We aim to measure the accuracy of the sensor/feature substitution. This includes verifying the selected substitute sensor or feature based on the similarity measures. However, since the implementation is still ongoing, we cannot yet experiment with real queries and data in order to measure the accuracy of our query analysis and rewriting as well as the impact on performance. Therefore, we propose next an experimental protocol describing different performance/accuracy related experiments.

[11] Link: http://spider.sigappfr.org/research-projects/event-detection-in-connected-environment/

#### 4.6.2.3 Experimentation

**Performance Experiments.** We aim to run obsolete queries through the optimizer in order to measure the execution run-time, and resource consumption of the query analysis, and rewriting algorithms. For Algorithm 1 we consider the following experiments:

- Experiment 1: Sensor number impact. The *check* method takes an event query $q$ as input and verifies the availability of all its sensors ($q.S$). In this test, we gradually increase the overall number of sensors $q.S$ (from 1 to $n$) while stabilizing the number of unavailable sensors in $q.S$. This test highlights how the number of sensors in a query $q$ impacts the performance of Algorithm 1.
- Experiment 2: Sensor unavailability impact. In contrast with Experiment 1, we stabilize here the overall number of sensors ($q.S$) in an event query ($q$) while gradually increasing the percentage of unavailable sensors within $q.S$. The use cases will range from the best case scenario (no unavailable sensors) to the worst case scenario (all sensors are unavailable). This test highlights how the unavailability of sensors impacts the behaviour of Algorithm 1.
- Experiment 3: Sensor mobility/breakdown impact. Algorithm 1 flags a sensor as unavailable if it is inactive, has moved away, or no longer covers the area requested in $q$. We stabilize here the number of unavailable sensors, and test three different use cases: (i) all unavailable sensors are inactive; (ii) all unavailable sensors have moved away; and (iii) all unavailable sensors no longer cover the requested area. This allows to compare how sensor mobility and breakdowns impact the performance separately.

The query rewriting algorithm takes a list of unavailable sensors as input and attempts to find substitutes for each one. If a substitute is not found, the algorithm tries to replace the feature provided by the 'irreplaceable' sensor instead. Hence, the algorithm is split into two parts: exact rewriting (sensor replacement), and approximate rewriting (feature replacement). Therefore, we propose the following experiments to evaluate the performance of Algorithm 2:

- Experiment 4: Input size impact. We vary the input size (list of unavailable sensors) from one (best case scenario) to $n$ (worst case scenario) in order to measure the impact on the algorithm's performance. We repeat this experiment in different use cases:
  - Use Case 1: We consider here that all sensors are replaceable. This means that the second phase of the algorithm (feature replacement) will not be executed. This represents a best (input size = 1) to a moderate (input size = $n$) case scenario.
  - Use Case 2: We consider here that all sensors are irreplaceable. The second phase of the algorithm (feature replacement) will be executed. This represents a moderate (input size = 1) to a worst (input size = $n$) case scenario.
  - Use Case 3: We consider a random configuration where the list of unavailable sensors (input of Algorithm 2) contains replaceable and irreplaceable sensors. This scenario will trigger first the exact rewriting phase of Algorithm 2 (i.e., substituting the replaceable sensors). Then, the approximate rewriting phase where the algorithm attempts to replace missing features with adequate substitutes.

- Experiment 5: Input distribution impact. In this test, we stabilize the input size and gradually increase the percentage of irreplaceable sensors. This allows to measure the impact of the input distribution (replaceable and irreplaceable sensors) on the performance.
- Experiment 6: Feature similarity impact. When replacing a feature with another, the algorithm uses an external function (denoted $ssf$ in Algorithm 9) that calculates the inter-feature similarity. Therefore, the performance is highly affected by the complexity of $ssf$. Since, various functions exist, this test compares the impact of different functions on the overall performance of our rewriting algorithm.

**Accuracy Experiments.** In our framework, an event is defined using an EQL-CE event query $q$. Then, the latter is used by an event detector that detects the actual event. However, the state of the connected environment $CE$ changes due to its dynamicity (i.e., $CE \longrightarrow CE'$), and $q$ could become obsolete due to missing/unavailable sensors or data in $CE'$. Therefore, query rewriting (i.e., $q \longrightarrow q'$) should allow the user to overcome this issue without changing the meaning of the event. In order to test the accuracy of the query rewriting process, we propose the following experiments:

- Experiment 7: Rewriting accuracy. In this test, we compare the original query $q$ to its rewriting $q'$ from an event detection standpoint. To do so, we consider/simulate both states of the connected environment (i.e., $CE$ and $CE'$). We provide the event detector with the event definition found in $q$ in order to detect the event in $CE$, and then use the same event detector with $q'$ in $CE'$. Finally, we measure the Normalized Mutual Information (NMI) and F-Score between the two results. These metrics measure the similarity between the two results, and their accuracy respectively. Regardless if the event is detected or not, the results should match in order to ensure that the query intent remained the same after the rewriting. We repeat the same exercise with different event queries.
- Experiment 8: Exact rewriting accuracy (sensor similarity evaluation). In this test, we evaluate the accuracy of the exact rewriting process (replacing sensors). To do so, we consider that the input (list of unavailable sensors) only contains replaceable sensors and run Experiment 7. Then, we evaluate the impact of sensor substitution, and our proposed similarity measure, on accuracy.
- Experiment 9: Approximate rewriting accuracy (feature similarity evaluation). In this test, we evaluate the accuracy of the approximate rewriting process (replacing features). To do so, we consider that the input (list of unavailable sensors) only contains irreplaceable sensors and run Experiment 7. However, when replacing features with other similar ones (cf. Algorithm 9), we use an external semantic similarity function denoted $ssf$. Since various functions exist in the related work [86], we test different functions in order to compare them.

#### 4.6.2.4 Preliminary Evaluation

We present here a preliminary evaluation of the query analyzer (cf. Algorithm 1). The aim of this evaluation is two-fold: (i) test the performance of the *check* method in terms of run-time; and (ii) test the accuracy of the obsolete query detection. We

ran the experiments on a MacBook Pro equipped with an Intel 2.8 GHz Core i7 (quad core) processor and 16 GB of RAM. We present first the performance test.

**Performance Test.** In this experiment, we considered an event query and varied the number of sensors that it relies on. This is a simulated environment where we generate sensors and link them to the query before running the *check* method. For each iteration, we increased the overall number of sensors and ran the *check* method - query analyzer (cf. Algorithm 1) and measured the run-time. Figure 4.10 shows that when increasing the number of sensors from 1024 to 16384 the query analyzer run-time maintains a linear evolution and the overall time required to process the worst case scenario (16384 sensors) does not exceed 6 ms. We should note, that normally an event query does not rely on a huge number of sensors, and when we measured the run-time for queries relying on 50 sensors or less the execution time was negligible. This indicates that the detection of obsolete queries is a light task that is scalable and capable of handling complex queries.

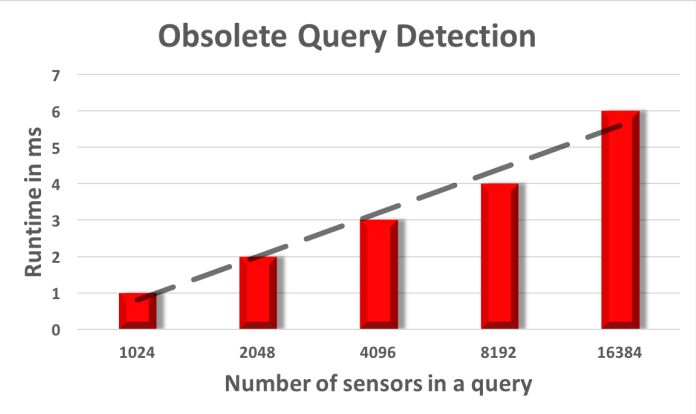


FIGURE 4.10: Query analyzer Run-time

**Accuracy Test.** In this experiment, we considered 10 event queries each relying on 10 sensors. We developed separate functions that simulate sensor breakdowns (randomly deactivating sensors), and sensor mobility (randomly modifying the current location/coverage area of sensors). Once sensors breakdown or change locations/-coverage areas, some event queries become obsolete. Therefore, to test the accuracy of the obsolete query detection we created four use cases:

- Use Case 1. All sensors were active and none changed locations.
- Use Case 2. We randomly deactivated 25% of the sensors (to simulate sensor breakdown). None of the sensors changed locations.
- Use Case 3. All sensors were active, and we randomly changed the locations of 25% of the sensors.
- Use Case 4. We randomly deactivated 50% of the sensors, and changed the location of 50% of the sensors.

In each of the aforementioned use cases, the query analyzer managed to detect the obsolete queries (for every use case the result matched the ground truth).

## 4.7 Summary

In this chapter, we propose a query optimizer that complements our previously proposed Event Query Language for Connected Environments (EQL-CE). The optimizer is equipped with a query rewriting engine that addresses queries that become obsolete due to the dynamicity of the connected environment. We focus on two main issues regarding dynamicity: (i) sensor mobility/breakdowns; and (ii) missing data/features. We propose two algorithms, the first automatically discovers obsolete queries, and the second rewrites them. Algorithm 1 checks the availability of sensors in a given event query by verifying their current locations, coverage areas, and status. Algorithm 2 attempts to replace unavailable sensors by other similar ones. To do so, we compare similarities using our proposed sensor similarity measure. In case a sensor is 'irreplaceable', the algorithm replaces the missing features (provided by the 'irreplaceable' sensors) using an external semantic similarity measure and a feature knowledge base. We evaluate the complexity of both algorithms, detail an experimental protocol to test with real data/queries once the implementation is over, and finally present some preliminary results.

# Chapter 5

# A Generic Event Detection Framework

*"Logic will get you from A to B. Imagination will take you everywhere."*

*— Albert Einstein*

We have already discussed the language that one could use to define a connected environment and all its components. Using EQL-CE queries one could define the environment itself, the deployed hybrid sensor network (based on the HSSN data model), the application domain, and the targeted events. In this chapter, we propose a generic framework for event detection in connected environments, and discuss how event definitions (provided by event queries) can be used to actually detect the aforementioned events.

Many existing applications guide connected environment users in their daily activities (e.g., navigation through traffic in s smart city, managing home comfort in s smart home). Although these applications are different in terms of purpose and application domain, they all detect events and propose actions and decision making aid to users. However, there is no usage of a common backbone for event detection that can be instantiated, re-used, and reconfigured in different use cases.

We propose here eVM, a generic event Virtual Machine that detects events in different contexts based on event definitions (queries) provided by the user. This allows domain experts to model, define, and detect their targeted events. eVM simultaneously considers the various features of the defined events (e.g., temporal, geographical), and uses the latter to detect different feature-centric events (e.g., time-centric, location-centric). Our proposed event detector uses different components (e.g., event queries, an event detection core). We detail here the event detection modules. Finally, we show that eVM is re-usable in different contexts and that the performance of our prototype is quasi-linear in most cases. Our experimental results showed that the detection accuracy is improved when, besides spatio-temporal information, other features are considered.

## 5.1 Introduction

Recent technological advances have allowed the integration of connected environments in various domains (e.g., homes, buildings, cities, factories), and the proposal of various event-based applications that help users in their daily tasks. For instance, smart homes/buildings [22, 108] allow users to manage energy consumption [1]. Moreover, these connected environments allow occupants to live in a more healthy [111], sophisticated [101], and comfortable [59] environment. Smart cities now offer traffic navigation aid [109] that help users avoid traffic congestion and accidents. Smart factories also benefit from connected environment applications [60] for production optimization and better machine maintenance scheduling. This is done by relying on sensors to detect breakdowns and faults. Patients are currently relying on sensed data from wearable sensors in order to improve health monitoring [45, 55, 104]. The provided applications detect medical events such as cardiac arrest, strokes, and bad gait (i.e., a person's manner of walking).

All of the aforementioned works are different (in terms of application domains, purposes, and objectives). However, they share common principles: (i) they need to detect a set of events; and (ii) they need to extract data and apply at least one data mining technique (e.g., clustering, classification). The main differences are two fold: 1) the targeted events and therefore their definitions/features; and 2) the choice of data mining/event detection technique.

Even-though all these works detect events, there is no usage of a common backbone for event detection that can be instantiated in different contexts/application domains. This is restrictive and costly since existing solutions suffer of the following issues: (i) the absence of an evolutionary approach capable of coping with needs that change over time; (ii) the absence of extensibility regarding the integration of new plug ins/complementary systems to an existing event detection approach; (iii) the difficulty of integrating different event-related modifications in the development; (iv) the impossibility of reusing the same framework to detect other events in various domains/contexts; and (v) the lack of expert input, i.e., providing a module where one can provide his/her own input on how to define the corresponding events prior to detection. Moreover, when dealing with huge amounts of heterogeneous sensed data, some technical aspects should be considered: (vi) multi-modality (the ability to consider various features and datatypes at once in the processing); (vii) incremental processing (i.e., allowing a continuous integration of new data in the set of already processed data); (viii) multi-source processing (considering various data providers at once); and (ix) human intervention should be limited to an acceptable level because of the amounts of data that need to be processed when detecting events. Thus, there is a need to design a generic event detection approach, considering expert input, and event features, to provide a more reusable event detector.

To answer these needs, we propose eVM: a re-usable solution for automatic and generic detection of "feature-centric" events (e.g., time-centric events, geo-centric events, temperature-centric events). Our framework integrates several components: 1) The Event Query Language (EQL-CE) in order to define the targeted events and several event describing features. Based on the latter, our approach detects the corresponding feature-centric events. This allows the framework to be generic and reusable in different event detection contexts/domains while allowing domain experts

to provide their input; 2) An easy to integrate API for the event detection core. This makes eVM evolutionary, extensible, and easy to integrate with other modules/systems and programming languages; and 3) access to the global framework's storage space where various repositories are available for storing event related data (e.g., data objects, event definitions, detected feature-centric events). Moreover, eVM's clustering technique simultaneously considers the various features, objects (observations) sensed by various sensors (several data producers) on different dates/times. eVM is based on an adaptation of FCA (Formal Concept Analysis) [39, 105], a backbone that provides a multi-modal, incremental, and multi-source clustering technique that handles high dimensional data, and requires low human intervention.

In order to validate our approach, we implemented eVM as a desktop-based application in order to evaluate the approach in different real case scenarios. Our experimental results show that the event detection accuracy is improved when additional features (i.e., other than time and geo-location) are taken into consideration. In addition, our performance results show quasi-linear behavior in most cases.

The rest of the chapter is organized as follows. Sections 5.2 and 5.3 review existing works on event detection and clustering techniques respectively. Then, Section 5.4 introduces some preliminaries on FCA. Section 5.5 details the eVM approach. The implementation and evaluation are discussed in Section 5.6. Finally, Section 5.7 summarizes the chapter.

## 5.2 Event Detection Background & Related Work

In this section, we review event definitions, types, and features before detailing some sensor-based event detection works in different areas (e.g., building/city management, environment monitoring, industry, medical). Since, in most cases, events are detected based on incoming raw data, without prior knowledge on the occurring events, various approaches use unsupervised clustering techniques. Since there are no commonly adopted criteria, we propose the following set of criteria to compare the referenced works:

**Criterion 1.** *Re-Usability*: This criterion examines the possibility $\{YES, NO\}$ of using the same approach as an event detection backbone for different targeted events in different contexts/domains (cf. Section 5.1 - limitation (iv)).

**Criterion 2.** *Domain Expertise*: This criterion measures the possibility $\{YES, NO\}$ of taking into account input from domain experts in the event definition process (cf. Section 5.1 - limitation (v)).

**Criterion 3.** *Evolution*: This criterion examines $\{YES, NO\}$ if an event detection approach can evolve and adapt to the changing event definition/detection needs over time (cf. Section 5.1 - limitation (i)).

**Criterion 4.** *Extensibility*: This criterion measures $\{YES, NO\}$ the capability of integrating new plug-ins/external modules in an existent event detection approach (cf. Section 5.1 - limitation (ii)).

**Criterion 5.** *Ease of Integration*: This criterion denotes $\{YES, NO\}$ an approach's capability of integrating event-related modifications in the development (cf. Section 5.1 - limitation (iii)).

In addition to the aforementioned criteria, we also consider other technical requirements/criteria such as:

**Criterion 6.** *Multi-modality*: This criterion states $\{YES, NO\}$ if multiple event features having different datatypes are considered (e.g., scalar and multimedia observations regarding various environment properties) in addition to time (instants, or intervals) and locations (GPS coordinates or textual location description) for improved event detection (cf. Section 5.1 - limitation (vi)).

**Criterion 7.** *Multi-source*: This criterion indicates $\{YES, NO\}$ if multiple sensors (i.e., data sources) could be considered at once. This is important since multiple data sources can provide valuable event related data (cf. Section 5.1 - limitation (viii)).

**Criterion 8.** *Incremental (continuous) processing*: This criterion considers the possibility $\{YES, NO\}$ of processing incoming data without having to repeat the entire processing, because data producers could provide event related data on different dates/times (cf. Section 5.1 - limitation (vii)).

**Criterion 9.** *Level of human intervention*: This criterion measures $\{HIGH, MODERATE, LOW\}$ how frequently users participate in the event detection process; since huge amounts of data is produced, it is important that user interventions become less frequent; we consider low intervention if users provide data input and initial configuration; moderate if users also intervene in result correction/optimization; and high intervention when users participate in the whole process (cf. Section 5.1 - limitation (ix)).

In the following, we begin by defining events, before detailing research works from various domains in which event detection has had noticeable impact (e.g., environmental monitoring, industry & manufacturing, building/city management, medical monitoring).

### 5.2.1 Basic Definition Of An Event

In the literature, many works [2, 4] define events as a happening that takes place at a particular time and location. Thus, emphasizing the importance of two main event features: (i) temporal; and (ii) spatial. All events are associated with these two features, since they answer the most common inquiries i.e., where and when. Nonetheless, additional event features are useful to describe the context and semantics of an event (e.g., temperature, movement, noise). The additional 'contextual' features differ from an event to another.

Events are categorized into different types, regardless of their contexts: (i) atomic or primitive events are the simplest events that can occur in a system. They cannot be decomposed into any smaller entity; (ii) composite or complex events are high level derived events, and are defined by combining constituent events. The latter can be atomic, or/and composite [2]. In the case of sensor events, atomic events are considered as observations (e.g., high temperature, high $CO_2$, existence of smoke). Moreover, composite events are considered as a combination of various elementary and/or composite events (e.g., a fire event is a combination of high temperature, high $CO_2$, and smoke).

In our proposal, we currently consider atomic events, nonetheless the framework is re-usable and extensible, and can easily integrate a module for event composition that allows the detection of composite events.

### 5.2.2 Event Detection Applications

Ever since sensor data modeling was extensively detailed (e.g., through ontology-based models such as the Semantic Sensor Network (SSN) [30]), sensor-based event detection started covering a larger spectrum of application domains [28]. From environmental monitoring (e.g., detecting fire hazards in forests, level of air pollution in a city), to building/city management (e.g., detecting energy wastes in smart buildings, detecting traffic congestion in a city), industrial processes (e.g., detecting events that disrupt production flow in a factory, detecting faults and machine maintenance issues), and medical event detection (e.g., monitoring a patient's heart condition) in various sensor networks. In all the aforementioned works, event detection requires a sensing (data collection) phase. During this phase, data is collected from the sensors that produce observations related to certain properties (e.g., temperature, movement, humidity). Moreover, events are usually composed of sensor-based 'contextual' features (e.g., temperature, humidity, $CO_2$) alongside spatio-temporal features since every sensor observation is mapped to an instant in time and a specific location. Many works [2, 62] agree that sensor observations are considered as atomic events (e.g., temperature rise event), therefore works regarding sensor data fusion [12] could target the composite events. In the following, we detail some event detection works in sensor networks, based on the application domains.

#### 5.2.2.1 Environmental Monitoring

In environmental monitoring scenarios, the sensor network contains more nodes (compared to personal medical sensing), and the spatio-temporal data acquisition intervals are wider. For example, to detect high air pollution events in a city, a huge number of air quality sensors should be deployed.

**Wildfire Detection.** In [32], the authors detect wildfire events in the wild by collecting sensor data such as temperature, relative humidity, and barometric pressure. In addition, they integrate spatial features by using a GPS unit in order to localize the detected events. Information is communicated using a wireless sensor network.

**Forest Fire Detection.** In [110], the authors propose an approach for real-time forest fire detection. They rely on spatio-temporal information and fire event context features such as relative humidity, temperature, smoke, and wind speed. They produce a report of abnormal atomic events (e.g., high temperature, smoke rising), and a real time forest fire danger rate from the collected data. Then they use a neural network to detect the fire events.

**Air Pollution Detection.** In [61], the authors use crowd-sensing in order to detect and monitor air quality related events in a city. In addition to time and geo-localization, air quality events share features that are related to the context of air quality (e.g., carbon monoxide (CO), air pressure, nitrogen dioxide ($NO_2$), and temperature). They also develop an android mobile phone application to display results to end users.

#### 5.2.2.2 Building/City Management

The following approaches serve building, home, or even city management in various ways (e.g., increasing comfort, safety, and reducing energy consumption).

**Reducing Energy Consumption.** In [1], the authors address the issue of energy savings in buildings by tackling the HVAC system (heating ventilation and air conditioning). These systems typically run on fixed schedules and do not consider building occupancy information. The authors present a presence sensor platform that can be used for accurate occupancy detection at the level of individual offices. The targeted event in this case is basically the absence of people (i.e., building occupants). Finally, their experiments show that considerable energy savings could be achieved by detecting the aforementioned event and adjusting the HVAC system adequately. Similarly, the authors in [59] tackle the same energy savings problem by relying on occupancy detection in buildings. The difference in this case is that instead of adjusting the HVAC, they adjust the lighting levels.

**Building Access Control.** In [101], the authors show interest in building security, and specifically access control. Instead of traditional pin codes and access cards, they propose a voice-based access control system. An individual's voice cannot be stolen, lost, forgotten, guessed, or impersonated with accuracy. Therefore, the authors propose and implement a system where a user speaks into a microphone in order to gain access to the building (or specific offices within it).

**Air Quality Monitoring.** In [111], the authors focus on indoor air quality because it leads to various illness issues. When reliable information about both the indoor and outdoor air quality is made available, a climate control system can provide the most appropriate amount of ventilation, ensuring safe and comfortable living conditions. Therefore, the authors develop the 'electronic nose', an array of sensors that monitor air quality. Finally, once they detect bad air quality, the ventilation system is activated to address the issue.

**Avoiding Machinery Faults.** In [60], the authors are interested in the monitoring of industrial equipment for optimized productivity. They detect various machine related events through a wide array of sensor data (e.g., speeds, vibrations). The overall objective is to closely monitor and synchronize the physical factory floor and the cyber computational space in an Industry 4.0 context. Moreover, by utilizing advanced information analytics, networked machines will be able to perform more efficiently, collaboratively and resiliently.

#### 5.2.2.3 Medical Monitoring

In medical monitoring scenarios, the sensor network contains a few nodes (e.g., some wearable sensors). The 'connected environment' here is reduced to the body (or part of body) of the patient. In the following we provide three examples.

**Abnormal Gait Detection.** In [45], the authors use lightweight, wearable sensors to monitor patients' gait (i.e., the manner of walking). People who suffer from strokes or spinal cord injuries, tend to have abnormal gaits. During medical treatment, it is beneficial to detect gait events when they occur (e.g., initial foot contact).

The authors propose two different ways for detecting such events, one using accelerometer data, and another using foot switch data (i.e., data from pressure/force sensor). In both cases, the event features are spatio-temporal, and sensor-related (i.e., accelerometer, pressure, force). The authors test both cases on normal, slow, and altered walking subjects and achieve near real time accurate detection of abnormal gait events.

**Walk Pattern Detection.** In [55], the authors declare that a variety of measurements are required for gait analysis (e.g., stride, step lengths, cadence, gait velocity). In order to acquire such measurements, the system needs to know when and where each foot leaves and touches the ground again. Therefore, the authors take interest in detecting the following events: (i) foot end contact (EC); and (ii) initial contact (IC). Thus, the authors propose an approach for IC and EC event detection using linear accelerometers and angular velocity transducers. They then use the event detection results to analyze gait patterns of healthy and injured individuals.

**Heart Arrhythmia Detection.** In [104], the authors propose a wireless smart sensor for heart monitoring. The aim is to detect life threatening events such as cardiac arrhythmia for patients with heart related issues. The sensor monitors heart rate and ECG (electrocardiogram) signals to detect the aforementioned events in real time.

#### 5.2.2.4 Discussion

Table 5.1 summarizes the evaluation of event detection approaches based on the aforementioned criteria. We split these approaches based on their application domains. Nonetheless, they all share two common characteristics: (i) they rely on a data acquisition networks (constituted of one or more sensors); and (ii) although they target different events, spatio-temporal event features are used by all methods. When considering the latter features, the chosen granularity can vary based on the application (e.g., for the spatial feature: we consider cities for environmental monitoring and the specific indoor location of a patient in a fall detection system). What differentiates the event definition from one approach to another are the specific (context related) features (e.g., temperature, movement, humidity). No current approach allows domain experts to contribute in event definition, i.e., the same event (e.g., abnormal gait) has variant definitions in different approaches. These works are not re-usable in different domains and contexts. When considering the other criteria we find that the level of human intervention varies from an approach to another. Finally even though these approaches are incremental, in most cases they are not extensible.

TABLE 5.1: Event Detection Works - Related Work Comparison

| **Criterion** | Event Detection Application Categories | | |
|---|---|---|---|
| | Environmental | Building/City/Industry | Medical |
| | [32, 61, 110] | [1, 59, 101, 60, 111] | [45, 55, 104] |
| *Re-Usability* | *NO* | *NO* | *NO* |
| *Domain Specific Expertise* | *NO* | *NO* | *NO* |
| *Evolution* | *NO* | *NO* | *NO* |
| *Extensibility* | *NO* | *NO* | *NO* |
| *Ease of Integration* | *NO* | *YES* | *NO* |
| *Multi-Modality* | *YES* | *PARTIAL*[12] | *YES* |
| *Multi-Source* | *NO* | *YES* | *NO* |
| *Incremental Processing* | *YES* | *YES* | *YES* |
| *Level of Human Intervention* | *MODERATE* | *MODERATE* | *MODERATE* |

## 5.3 Clustering Related Work

Many works in different areas (e.g., information retrieval, event detection, image searching and annotation), have evolved around clustering techniques since their introduction in 1975 when John Henry Holland wrote a book on genetic algorithms entitled "Adaptation in Natural and Artificial Systems" [52]. Unsupervised clustering is considered since in most cases, we detect events from raw data without prior knowledge on the occurring events. The data are organized by groups (clusters) that represent each a specific event. Clustering techniques are commonly grouped into four categories [97]:

### 5.3.1 Prototype-Based Clustering

A cluster is a set of objects that are closest (most similar) to the prototype that defines the cluster than to the prototype of any other cluster. A prototype can be the centroid or the medoid depending on the nature of the data (continuous attributes or categorical attributes). For continuous data, a centroid represents the object with the average (mean) values of all objects (points) in the cluster. As for categorical attributes, since a centroid is not meaningful, the prototype is often a medoid, the most representative point of the cluster. For many types of data, the prototype can be regarded as the most central point. Therefore, prototype-based clustering is commonly referred to as center-based clustering. For example, K-means [53] is a prototype-based clustering technique that groups objects based on a specified similarity measure (e.g., Euclidean distance, Manhattan distance, cosine similarity, Jaccard measure) and creates a set of K clusters represented each by a centroid. K-medoids [57] is another example of this clustering category. Instead of calculating means, actual points from the data are picked as representatives (prototypes) of the clusters. Points are associated to the clusters where they are most similar to the prototype. An iterative swapping process between prototypes and non prototype points is done as long as the quality of the clustering is improved.

These methods have low complexities for both time and space. But the algorithms attempt to find a predefined number of clusters (K): the final number of clusters should be known prior to clustering. In addition, for K-means, in order to start the clustering, the user has to choose initial cluster centers (centroids). This is a key step, if these centroids are chosen randomly clustering results can be poor.

### 5.3.2 Density-Based Clustering

A cluster is represented as a dense region surrounded by a low density region. Objects in the low density zones are considered as noise while others in high density regions belong to the group limited by the region. For example, DB-Scan [80] produces a partitional clustering based on density measures. This method studies the neighborhood of each point, and partitions data into dense regions separated by not-so-dense regions. To do so, density at a point p is estimated by counting the points within a circle of center p and radius $\epsilon$. Therefore, a dense region is a circle of radius $\epsilon$ containing a minimal number of points.

[12] We use *Partial* to indicate that not all the works are multi-modal

On one hand, DB-Scan determines automatically the number of clusters, is relatively resistant to noise, and can handle clusters of arbitrary sizes and shapes. On the other hand, since clustering is affected by the specified radius, DB-Scan loses accuracy when the clusters have widely varying densities. Also, with high-dimensional data, defining the densities becomes more difficult and more expensive (in term of computation time and space). Finally, points in the low-density areas are considered as noise which means that not all input data will be present in the clusters.

### 5.3.3 Graph-Based Clustering

Data is organized in graphs/hierarchies where nodes are objects and connections among objects are represented as links connecting the nodes. Therefore, a cluster is defined as a connected component, a group of objects that are connected to one another but have no connections to objects from outside the group. For example, Agglomerative Hierarchical clustering is a graph-based clustering method [18]. First, each point is considered as a singleton cluster. Then repeatedly, the closest two clusters (based on similarity/dissimilarity matrices) are merged until a single all-encompassing cluster remains. Hierarchical clustering can also be divisive, this method is symmetrical to the agglomerative technique. In the divisive algorithm, all points are initially assigned to a single cluster and then based on similarity/dissimilarity measures the splitting into different clusters begins, until each point is assigned to a distinct cluster.

The added value of this method is that clusters are nested in a dendrogram (hierarchical structure) which offers a first level of semantic reasoning by exploiting the hierarchy and the inter-cluster relations. In contrast, the method has a high complexity in both time and space. All cluster merges are final, for high dimensional data such as photos, this is considered as a limitation. Since high dimensional data is more complicated, error correction if data is wrongly assigned to a certain cluster is a major issue.

### 5.3.4 Conceptual Clustering (Shared-property)

A cluster is a set of objects that share some properties. For successful clustering, an algorithm would require a very specific definition of a cluster. This means that prior to the clustering, the shared properties that identify a cluster should be defined in order to generate a concept describing a cluster. The process of generating such clusters is called conceptual clustering. Formal Concept Analysis (FCA) is a conceptual, hierarchical clustering method [23, 78]. It analyses data based on object/attribute relationships, extracts concepts, and finally orders them in a lattice.

The advantage of having a lattice of formally defined Concepts is that it assures a more advanced level of semantic reasoning. In addition, FCA automatically generates a brief description for each cluster. Nonetheless, time and space complexities could cause concerns in some worst case scenarios where every data object forms a formal concept. In this case, exponential complexities become major technical difficulties.

### 5.3.5 Discussion

Table 5.2 shows a comparative summary of clustering techniques with respect to our technical (clustering-related) criteria (cf. Section 5.2). Prototype-based methods require excessive human intervention and the number of clusters prior to the processing. This is a major limitation in an event detection scenario where the total number of events is unknown prior to detection. In addition, these approaches are not multi-modal, multi-source nor incremental. Density-based methods detect automatically the number of final clusters, thus reducing human intervention but they are not multi-modal nor multi-source. These methods do not consider different types of data at once. In addition, clustering high dimensional data is complicated when relying on density measures. Graph-based (Hierarchical) clustering offers better semantic reasoning compared to the first two techniques. It enables a first level of semantic-based processing by exploiting the hierarchy and inter-cluster relations. In addition, Hierarchical Clustering is accurate but remains highly expensive computation wise. Finally, Conceptual Clustering presents two main advantages. Firstly, incremental algorithms exist and offer lower complexities for time and space. Secondly, these techniques (e.g., FCA) offer two levels of semantic reasoning: (i) handling formal concepts as nodes, and (ii) generating an ordered lattice of concepts (nodes). Finally, FCA is multi-modal, dynamic, and multi-source.

TABLE 5.2: Clustering Technique Comparison

| **Criterion** | **Clustering Technique** | | | |
|---|---|---|---|---|
| | Prototype-based [53, 57] | Density-based [80] | Graph-based [18] | Conceptual [23, 78] |
| Multi-Modality | No | No | No | Yes |
| Multi-Source | No | No | No | Yes |
| Incremental Processing | No | No | No | Yes |
| Level of Human Intervention | High | Moderate | Moderate | Low |
| Predefined Cluster Number[13] | Yes | No | No | No |

## 5.4 FCA Preliminaries & Definitions

After studying various clustering techniques [18, 23, 53, 80], we chose Formal Concept Analysis (FCA) [39, 105] as the backbone for our approach. FCA is incremental and extensible (criteria 8 and 4). It examines data through object/attribute relationships, extracts formal concepts and orders the latter hierarchically in a Concept Lattice which is generated through a four step process [27]:

**Step 1.** Defining a **Formal Context** (Definition 5) from the input data, based on object/attribute relations represented in a cross-table.

**Definition 5.** *A **Formal Context**: is a triplet $\langle X, Y, I \rangle$ where:*

- *$X$ is a non-empty set of objects*
- *$Y$ is a non-empty set of attributes*
- *$I$ is a binary relation between $X$ and $Y$ mapping objects from $X$ to attributes from $Y$, i.e., $I \subseteq X \times Y$.*

[13] This criterion states if the final number of clusters is required prior to clustering.

*Remark.* Table 5.3 shows an example, where sensor observations are objects and features (time, space, temperature) are attributes. The cross-joins represent the mapping of observations to their respective attributes (e.g., $o_1$ is a temperature observation that was taken in Shop 1 by sensor $s_1$ during time interval $ti_1$). ■

TABLE 5.3: Formal Context example

| | | Features | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Time | | | Location | | | Sensor | | | Contextual | | |
| | | $ti_1$ | $ti_2$ | $ti_3$ | Shop 1 | Coffee Shop | Bank | $s_1$ | $s_2$ | $s_3$ | Temperature | $CO_2$ | Smoke |
| Observations | $o_1$ | X | | | X | | | X | | | X | | |
| | $o_2$ | X | | | X | | | X | | | X | | |
| | $o_3$ | | X | | | | X | | | X | | X | |
| | $o_4$ | | | X | | X | | | X | | | | X |
| | $o_5$ | | | X | | X | | | X | | | X | |

**Step 2.** Adopting **Concept Forming Operators** to extract **Formal Concepts** (Definition 6). FCA has two concept forming operators:

- $\uparrow$: $2^X \rightarrow 2^Y$ (Operator mapping objects to attributes)
- $\downarrow$: $2^Y \rightarrow 2^X$ (Operator mapping attributes to objects).

For example, from the cross-table shown in Table 5.3, we have $\{o_3\}^\uparrow = \{ti_2, Bank, s_3, CO_2\}$ and $\{Smoke\}^\downarrow = \{o_4\}$.

**Definition 6.** *A **Formal Concept** in $\langle X, Y, I\rangle$ is a pair $\langle A_i, B_i\rangle$ of $A_i \subseteq X$ and $B_i \subseteq Y$ such that: $A_i^\uparrow = B_i \wedge B_i^\downarrow = A_i$.*

*Remark.* Consider the set of observations $A_1 = \{o_1, o_2\}$ and the set of attributes $B_1 = \{ti_1, Shop1, s_1, Temperature\}$. $A_1^\uparrow = \{ti_1, Shop1, s_1, Temperature\}$ and $B_1^\downarrow = \{o_1, o_2\}$. Thus, since $A_1^\uparrow = B_1$ and $B_1^\downarrow = A_1$, the pair $\langle A_1, B_1\rangle$ is a Formal Concept. ■

**Step 3.** Extracting a **Subconcept/Superconcept Ordering** relation for **Formal Concept** (cf. Definition 6) ordering by defining the most general concept and the most specific concept for each pair. The ordering relation is denoted $\leq$. For example, from Table 5.3, let $A_1 = \{o_4\}$, $B_1 = \{ti_3, CoffeeShop, s_2, Smoke\}$, $A_2 = \{o_4, o_5\}$, and $B_2 = \{ti_3, CoffeeShop, s_3\}$. According to Definition 6, $\langle A_1, B_1\rangle$ and $\langle A_2, B_2\rangle$ are formal concepts. In addition, $A_1 \subseteq A_2$ therefore, $\langle A_1, B_1\rangle \leqslant \langle A_2, B_2\rangle$. This means that formal concept $\langle A_1, B_1\rangle$ is a subconcept of formal concept $\langle A_2, B_2\rangle$ (which is the superconcept).

**Step 4.** Generating the **Concept Lattice**, which represents the concepts from the most general one (top) to the most specific (bottom). The lattice is defined as the ordered set of all formal concepts extracted from the data (based on $\leq$). A **Concept Lattice** denoted by $\beta(X, Y, I)_\leqslant$ is the set of all formal concepts of $\langle X, Y, I\rangle$ ordered by the subconcept/superconcept ordering relation $\leqslant$ where:

$$\beta(X, Y, I) = \{\langle A, B\rangle \in 2^X \times 2^Y | A^\uparrow = B, B^\downarrow = A\}$$

$\beta(X, Y, I)_\leqslant$ associated with a subconcept/superconcept ordering relation is called a concept (Galois) lattice. For the example shown in Table 5.3, Fig. 5.1 illustrates

the Concept Lattice[14]. The top node is the concept regrouping all objects having no attributes in common. As we go down in the hierarchy, we notice that concepts have less objects and more shared attributes (a logic OR and AND are applied to objects and attributes respectively when scrolling down towards the bottom node). The bottom node is the most specific, thus regrouping all attributes having zero objects in common. The next section formally describes our eVM approach and how the aforementioned FCA steps are integrated and adapted for the clustering of sensor observations.

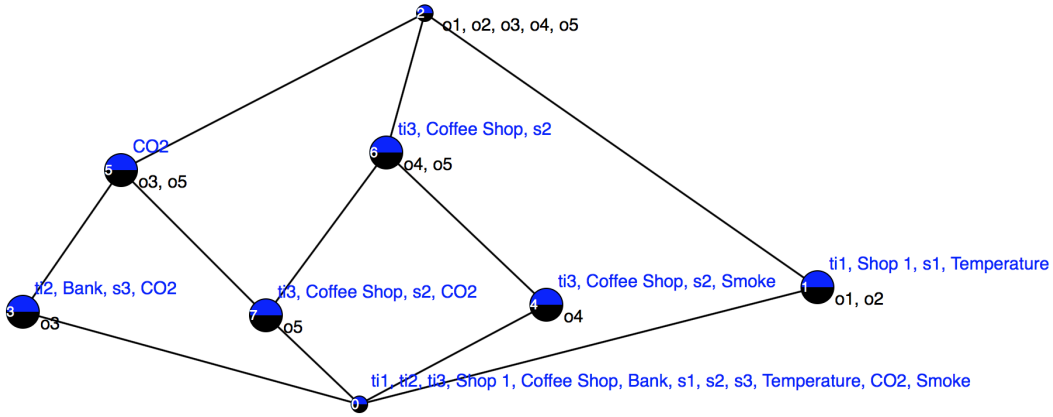


FIGURE 5.1: The Concept/Galois Lattice

## 5.5 Event Detection Framework

In this section, we detail the eVM framework. Firstly, we provide an overview of our proposal. Then, we focus more on the clustering technique used in our event detection process and detail each module separately.

### 5.5.1 Approach Overview

As previously discussed in section 5.2, existing approaches heavily rely on time and location when defining the targeted events. In order to tackle specific events, additional context related information (e.g., temperature, $CO_2$, smoke, humidity) are added. Nonetheless, existing works, such as [1, 32, 45], provide a static event definition that does not allow modifications (adding, removing event features). Since event definition and detection needs change over time, or from one context/domain to another, we provide a generic and extensible event definition in eVM.

#### 5.5.1.1 A Generic Event Definition

We model events as $2D^+$ spaces (having at least two dimensions: temporal and spatial). Depending on the targeted events, one can specify/create additional dimensions, representing the contextual event features based on sensor observations (e.g., temperature, movement, noise, humidity) and various levels of granularity for each feature (e.g., year-month-week-day-hour-minute-second for time, country-region-city-street-building-floor-room for location). Micro granularity can also be considered depending on the application purpose (e.g., time: minutes-seconds for short

[14]The node numbering in the figure does not imply any particular sequence or order

events, location: human body-part of a body for medical applications). This way of defining events is dynamic, extensible, and allows domain expert input. Moreover, the event's space mathematically represents the event query context as defined in Chapter 4 (cf. Definition 2). Nonetheless, it does not consider yet digital events, where the location dimension should be handled differently (e.g., server attacks that could happen on multiple nodes in the same time).

#### 5.5.1.2 Translating Event Queries

Depending on the event detection needs, domain experts can create, insert, update and delete datatypes, event features, granularity, and event definitions using EQL-CE. Every time an event query is submitted, a common Event Query Compiler (cf. Fig. 5.2) executes the submitted query, thus creating an instance of event detection. Finally, using the defined instance, a common Event Detector mechanism is triggered. This process queries event related data, stored in specific repositories (found in the storage space), and starts detecting events based on the provided event model (i.e., event definition query). The Event Detector integrates FCA as the backbone clustering technique, to provide a multi-modal, dynamic, and incremental approach (the API components are later detailed in Fig.5.3). This makes the eVM framework re-usable in different contexts, evolutionary, and easy to integrate with any API friendly programming language. In this chapter, we detail the Event Detection part of this framework.

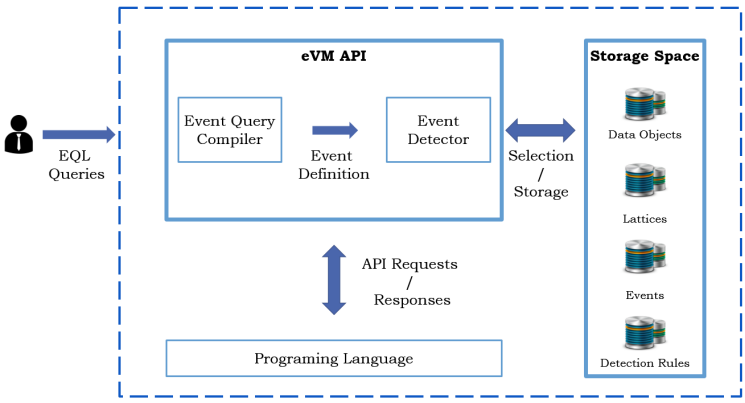


FIGURE 5.2: eVM Overview

### 5.5.2 The eVM Framework

In order to organize a set of event-related data objects (e.g., sensor observations) according to feature-centric events, the eVM process is split into four main steps: (i) Event definition & data pre-processing (executed by the Event Query Parser, Event Query Executor, and Pre-Processor modules); (ii) Attribute extraction (executed by the Attribute Extractor module); (iii) lattice construction (executed by the Event Candidates Lattice Builder module); and (iv) event detection (carried out by the Feature-Centric Event Detector and Rule Selector modules). In the following, we detail each processing step and module.

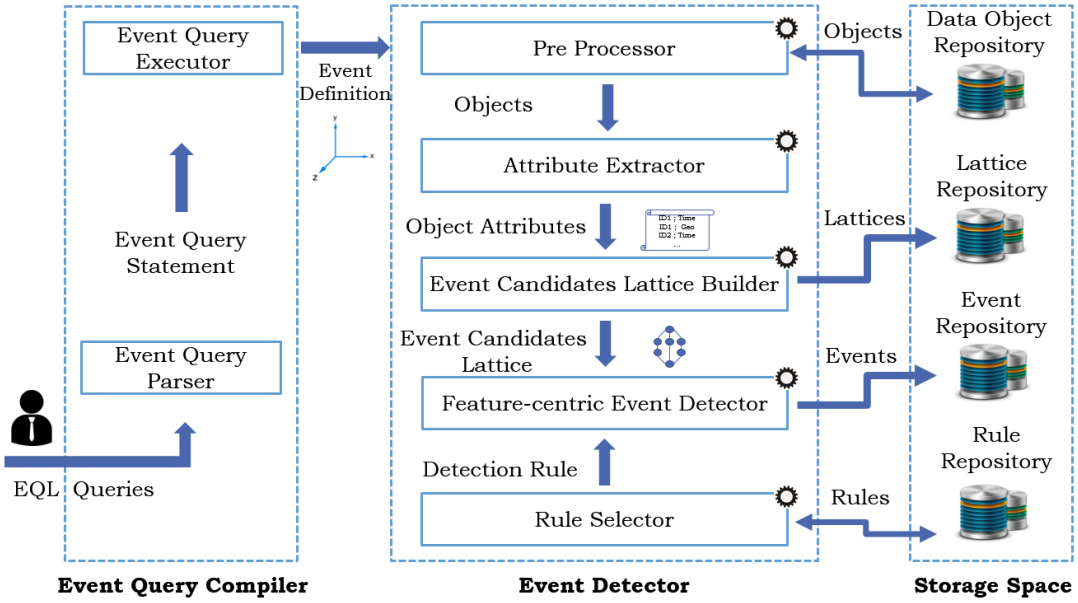


FIGURE 5.3: eVM API Components

#### 5.5.2.1 Event Definition & Data Pre-processing

When the Event Query Compiler receives an incoming event query $q$ (submitted by the user via EQL-CE), the parser checks the syntax of the submitted query. Then, the Event Query Executor extracts the query context (cf. Definition 3) to output the $2D^+$ event space. The aforementioned space is a formal mathematical representation of the event definition where each dimension of the space represents an event feature. In EQL-CE, features are defined by an identifier, a type, a distance measure, a default value, and a description field (cf. Syntax 12 - Section 3.5). Feature related information are extracted and used when creating the dimension that represents a feature. The feature description is a 4-tuple $f : \langle label, G, gran, interval \rangle$ where:

- *label* is the feature's label.
- $G$ is the set of granularity associated with the feature.
- *gran* is a function that converts any granularity value to another (related to the same feature).
- *interval* is a Boolean indicating if the feature is generated as an interval (true), or not (false).

For example, the following shows the description field of five (fire) event features each having a label, a set of granularity, an indication about interval construction, and a conversion function:

- $Time_f : \langle 'Time', \{Year, Month, Week, Day, Hour\}, 1, Convert_{Time} \rangle$
- $Geo_f : \langle 'Geo', \{Country, Region, City, Street\}, 0, Convert_{Geo} \rangle$
- $Temp_f : \langle 'Temp', \{value, setofvalues, mean, max, min\}, 1, Convert_{Temp} \rangle$
- $CO2_f : \langle 'CO_2', \{value, setofvalues, mean, max, min\}, 1, Convert_{CO2} \rangle$
- $Smoke_f : \langle 'Smoke', \{singlevalue, setofvalues\}, 0, Convert_{Smoke} \rangle$

For example, time, geo, temperature, $CO_2$, and smoke features could be used to define a fire event that can be detected from sensor data in a connected environment.

Moreover, in order to formally represent a feature as a dimension in a $2D^+$ space, a data type is required for all data elements that belong to the dimension/feature. We denote this datatype *adt* (attribute data type) and define it as follows:

$$adt : \langle label, f, t, range, dist \rangle \quad where :$$

- *label* is the attribute data type's label.
- $f$ is the event feature mapped to the attribute data type.
- $t$ denotes the primitive data type of the attribute $\mid t = f.datatype$.
- *range* is the domain of the attribute values.
- *dist* is the function that returns the distance between any two values of the same attribute datatype $\mid dist = f.distance\_measure$.

To continue with the fire event example, the following describes five attribute data types each having a label, an associated event feature, a primitive datatype, a range, and a distance function (e.g., time difference for temporal attributes, spatial distance for geographical attributes, temperature, $CO_2$, and smoke differences between various sensor readings for instance):

- $Time_{adt} : \langle 'TimeAttribute', Time_{feature}, Date, Any, TimeDifference \rangle$
- $Geo_{adt} : \langle 'GeoAttribute', Geo_{feature}, String, Any, SpatialDistance \rangle$
- $Temp_{adt} : \langle 'TempAttribute', Temp_{feature}, Float, Any, TempDifference \rangle$
- $CO2_{adt} : \langle 'CO2Attribute', CO2_{feature}, Float, Any, CO2Difference \rangle$
- $Smoke_{adt} : \langle 'SmokeAttribute', Smoke_{feature}, Boolean, Any, SmokeDifference \rangle$

Each time an attribute data type is created, it is saved in the $ADT$ list for future use.

Finally, each dimension can now be created by assigning to it an identifier *id*, a start value *o*, and an attribute data type *adt* (which links the dimension to its event feature as described in the event query). A dimension $d$ is formally defined as follows:

$$d : \langle id, adt, o \rangle \quad where :$$

- *id* is the unique identifier of the dimension.
- *adt* is the attribute datatype that maps a dimension to its feature.
- *o* is the origin point of a dimension $\mid o = adt.f.default\_value$

For example, the following describes five event space dimensions, each having an identifier, an origin value, and an attribute data type (and therefore an associated event feature). These event dimensions help define the event space of a fire event:

- $Time : \langle 1, Time_{adt}, 30/12/20171 : 30pm \rangle$
- $Geo : \langle 2, Geo_{adt}, Paris \rangle$
- $Temp : \langle 3, Temp_{adt}, 20 \text{ (degrees Celsius)} \rangle$
- $CO2 : \langle 4, CO2_{adt}, 250\text{PPM (Parts Per Million)} \rangle$
- $Smoke : \langle 5, Smoke_{adt}, False \rangle$

Finally, an event space can be defined by an identifier, the set of all its dimensions ($D$), and the set of all its sensor observations/data objects ($SO$):

$$eSpace : \langle id, D, SO \rangle \quad where :$$

- $id$ is the unique identifier of the event space.
- $D$ is a set of dimensions that constitute the space (such the $||D|| \geq 2$).
- $SO$ is the set of data objects that belong to the event space (the list of objects is empty at the space creation, after the detection process all data objects are inserted into their respective spaces).

For example, $fire : \langle 1, (Time, Geo, Temp, CO2, Smoke), SO \rangle$ defines the fire event.

Once the event definition (space) is established from the event query, the Pre-processor module requests event related data from the storage unit (i.e., from the object repository). The purpose of this step is to select the sensor observations (taken from the connected environment) that could contribute to the event space. To do so, we analyze the attributes of each data object/sensor observation (cf. Definition 7). An attribute is defined as a value associated with an attribute data type. We define a data type function denoted $dt$, that returns the attribute data type of a value based on the data object attributes.

**Definition 7.** *A **Data Object** is defined as a 2-tuple, so* : $\langle id, V \rangle$*, where:*

- *id is the unique identifier of a data object*
- *V is a set of attribute values according to a given ADT, such that* $\forall a_i \in ADT \quad \exists v_i \in V \mid dt(v_i) = a_i$. ■

*Remark.* In the context of sensor event detection, the input data objects are considered as sensor observations. However, in other application scenarios such as social event detection the data objects could be images or videos shared on social media platforms. Therefore, to ensure that the event detector is reusable in different context we keep the formal definitions as generic as possible.

The Pre-processor will extract objects (i.e., sensor observations) having attributes (i.e., values/metadata) related to the features found in the event space. For instance, if one targets overheating events having temporal, spatial, and temperature features, the Pre-processor extracts from the data object repository (cf. Fig.5.3) all objects having the following attributes: (i) a temporal; (ii) a geo-location; and (iii) a temperature observation value. Finally, the selected data objects are sent to the Attribute Extractor module.

#### 5.5.2.2 Attribute Extraction

In this step, the event definition (provided by the incoming event space $eSpace$) is essential for knowing which data object attributes should be extracted and included in the rest of the processing. The attribute extraction objective is to examine the dimensions that constitute $eSpace$ and select the list of attribute data types needed for event detection. The Attribute Extractor module (cf. Fig.5.4) initiates a cleaning process via the Converter sub-module in order to have the same units for data object attributes (e.g., having all temperature values in Celsius). The cleaned data objects

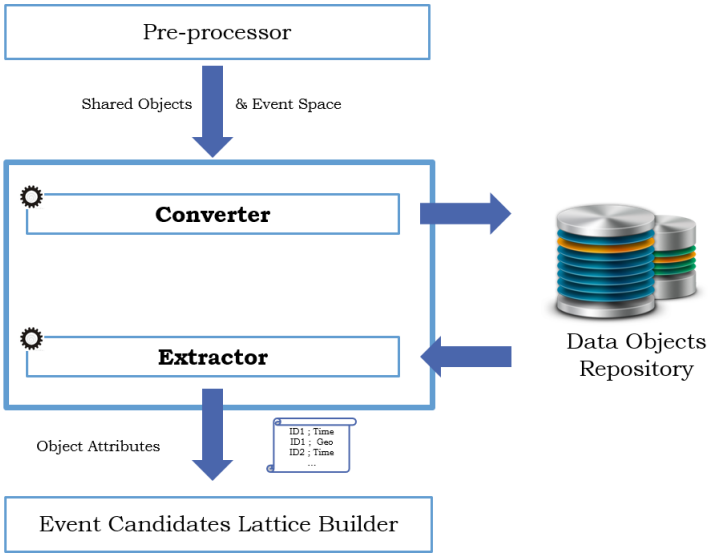


FIGURE 5.4: The Attribute Extractor module

are stored in the data objects repository (cf. Fig.5.3). Finally, from every data object, the Extractor sub-module extracts the needed attributes (based on the event definition). Both data objects and their attributes will be used in the following steps for lattice construction.

#### 5.5.2.3 Lattice Construction

In this step, an event agent processes the previously extracted attributes, and data objects into lattice attributes and objects, in order to generate one output: the lattice. The Feature-centric Event Lattice Builder is the FCA backbone. It integrates the four step process of FCA clustering described in Section 5.4. To do so, we define lattice attribute types in Definition 8. These types will be used when defining the lattice attributes (cf. Definition 9). Lattice attributes, as defined here, ensure that any given object/attribute can be represented in the FCA formal context. Therefore, any object (i.e., sensor observation) having attributes can be properly integrated in the clustered data set. This allows the event detection process of eVM to be generic and applicable in various contexts. Finally, for object/lattice attribute mapping, we define a binary cross rule denoted $BXR$ (cf. Definition 10). This process is repeated for each event detection run.

**Definition 8.** ***lat*** *is a lattice attribute type representing an interval* $[a, b[$ *where* $lat : \langle a, b, T \rangle$*, where:*

- *a is the lower boundary value*
- *b is the upper boundary value*
- *T is a value representing the period having a primitive data type of either integer or float, such that:*
  - $dt(a) = dt(b) \in ADT$ *and*
  - $b = a + T$. ■

**Definition 9.** *A lattice attribute, denoted* ***la****, is defined as a 4-tuple* $la : \langle f, eSpace, lat, y \rangle$ *where:*

- $f \in F$ *is the event feature mapped to lattice attribute la*
- $eSpace$ *is the event space in which the detection will take place*
- $lat$ *(cf. Definition 8) is the lattice attribute type*
- $y$ *is a granularity* $\mid y \in f.description.G$ *and*

$$lat.T = \begin{cases} y & \text{if } f.description.interval = True \\ 0 & Otherwise \end{cases}$$

$lat.a = so_i.v_j$, *where:*

  - $so_i \in eSpace.SO$ *and*
  - $(v_j \in so_i.V) \wedge (dt(v_j).f = f)$.

■

For example, from the fire event example, we can find the following lattice attributes:

- Time intervals
- Geo locations
- Temperature intervals
- CO2 intervals
- Smoke existence (or not)

**Definition 10.** *A binary cross rule, denoted as* ***BXR****, is defined as a function that maps a shared object* $x$ *to its respective lattice attribute* $y$ *where* $x.v_i \in x.V$*:*

$$BXR = \begin{cases} 1 & \text{if } (y.lat.T = 0 \wedge y.lat.a = x.v_i) \vee \\ & (y.lat.T \neq 0 \wedge x.v_i \in [y.lat.a, y.lat.b[) \\ 0 & Otherwise \end{cases}$$

■

Then, the Feature-centric Event Lattice Builder constructs the FED (Feature-centric Event Detection) formal context, denoted $ffc$ (cf. Definition 11). Once the $ffc$ is created, formal concepts are extracted and a lattice (cf. Figure 5.6) is generated. This process is described in steps 2-4 of Section 5.4. This lattice is called an Event Candidate Lattice, where each node is a potential feature-centric event. Figure 5.5 illustrates the inner composition of the Event Candidates Lattice Builder module.

**Definition 11.** *A FED Formal Context, denoted* ***ffc****, is defined as a 6-tuple:*

$$ffc : \langle eSpace, F, f_{LAG}, X, Y, I \rangle \quad where$$

- $eSpace$ *is the event space in which the detection takes place*
- $F$ *is the set of one event features*
- $f_{LAG}$ *is the function that generates the lattice attributes, described in Algorithm 10*
- $X = eSpace.SO$ *is the set of shared objects*

- $Y = \bigcup_{i=0}^{|X.V|-1}\{la_i\}$ *is the set of lattice attributes* $\mid X.V = \bigcup_{\forall so \in X}\{so.V\}$ *is the union of all attribute values from the shared objects in eSpace*
- *I is a BXR(x,y) where* $x \in X \wedge y \in Y$. ■

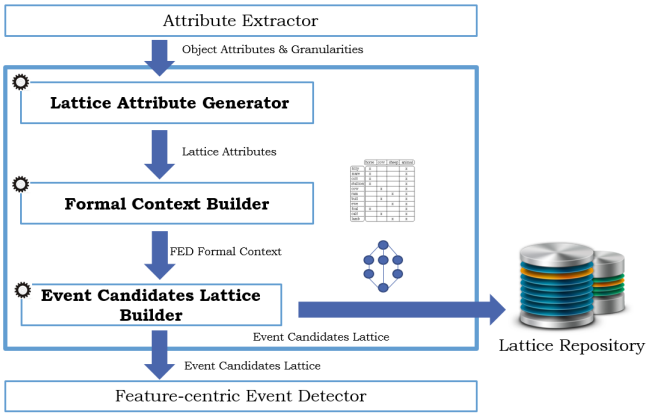


FIGURE 5.5: The Event Candidates Lattice Builder module

To follow up with the fire event example, Table 5.4 illustrates how lattice attributes (columns) are mapped to incoming sensor observations (rows) using the binary cross rule in the FED formal context.

TABLE 5.4: Fire Event Formal Context Example

| | Time | | Geo | | Sensor | | | Temperature | | CO2 | | Smoke | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TI1 | TI2 | Loc1 | Loc2 | S1 | S2 | S3 | High | Low | High | Low | Yes | No |
| **O1** | X | | X | | X | | | X | | | | | |
| **O2** | X | | X | | | X | | | | X | | | |
| **O3** | X | | X | | X | | | | | | | X | |
| **O4** | | X | | X | | | X | | | | | | X |
| **O5** | | X | | X | | | X | | | X | | | |
| **O6** | | X | | X | | | X | | X | | | | |

The example in Table 5.4 shows six sensor observations (objects), mapped to their respective attributes. For instance, observation 1 has a timestamp value inside time interval 1, therefore it is mapped to the lattice attribute $TI1$. This observation is taken from a sensor $S1$ deployed in $Loc1$ and has a temperature reading that is included in the *High* range. Moreover, the other five observations are also mapped to their corresponding attributes using the binary cross rule. This represents the (FED) formal context in this scenario.

In Algorithm 10, we detail the lattice attribute generation process. This starts by extracting all object attribute values (lines 5-11). If the value is mapped to a feature that is generated as an interval (e.g., time), the algorithm calls the Create-Intervals function (lines 19-23). If not (e.g., smoke), the algorithm generates a lattice attribute type having a null period and creates the corresponding lattice attribute (lines 13-18). This step allows the creation of generic lattice attributes from various features, thus providing multi-modality (cf. Criterion 6). Algorithm 11 details the Create-Intervals function. This process extracts all values related to the same feature (lines 4-9), orders them (line 10), selects a minimum and a maximum value (lines 11-12), and creates periodic intervals starting from the minimum to the maximum value (lines 14-22). The period is calculated based on the chosen feature granularity (line 15). Finally, the result is added the the output of Algorithm 10.

**Algorithm 10:** Lattice Attribute Generation (cf. Definition 9 - $f_{LAG}$)

```
1  Input: eSpace
2  Output: RES                                          // List of all lattice attributes
3  VAL = new List()                                     // Shared Objects attribute values list
4  PD = new List()                                      // Processed event features list
5  foreach so ∈ eSpace.SO do
6      foreach v ∈ so.V                                 // This loop extracts all object
7      do                                                  attribute values from all objects in
8          if (v ∉ VAL) then                               eSpace and stores them in the VAL list
9              VAL←v
10     end
11 end
12 foreach v ∈ VAL do
13     if (not dt(v).f.Interval)                        // If the value is not generated
14     then                                                as an interval
15         lat ← LAT(v, lat.a + lat.T, 0)
16         la ← LA(dt(v).f, eSpace, lat, dt(v).f.description.g)     // Create la with lat.T=0
17
18         RES ← la
19     else
20         if (dt(v).f ∉ PD) then
21             RES ← (Create-Intervals(VAL,v,PD, eSpace))          // Call
22                                                                    Create-Intervals
23     end                                                         function
24 end
25 return RES
```

**Algorithm 11:** Create-Intervals

```
1  Input: VAL, v, PD, eSpace                            // Input provided by Algorithm 10, line 21
2  Output: LAI                                          // Generated lattice attributes intervals
3  int i = 0
4  TEMP = new List()                                    // Temporary object attribute list
5  foreach val ∈ VAL do
6      if (dt(val).f == dt(v).f)                        // Extract all object attribute
7      then                                                values having the same feature
8          TEMP ← val                                      as v and store them in TEMP
9  end
10 Order_ascending(TEMP)                                // Order TEMP ascending
11 min ← TEMP.get(0)                                    // min is the first element of TEMP
12 max ← TEMP.get(|TEMP| − 1)                           // max is the last element of TEMP
13 lat ← LAT()
14 while (lat.b < max) do
15     lat ← LAT(min, lat.a + (i+1) × lat.T, dt(v).f.g)
16     if (lat.b > max)                                 // This loop
17     then                                                creates
18         lat.b ← max                                     intervals of
19     la ← LA(dt(v).f, eSpace, lat, dt(v).f.g)            period lat.T =
20     LAI ← la                                            f.g (feature
21     i++                                                 granularity)
22 end
23 PD ← dt(v).f                                         // Add feature to the list of processed features
24 return LAI
```

Once the formal concepts are extracted from the fire event FED formal context using the FCA operators the lattice is generated as shown in Figure 5.6 (where nodes are potential fire events).

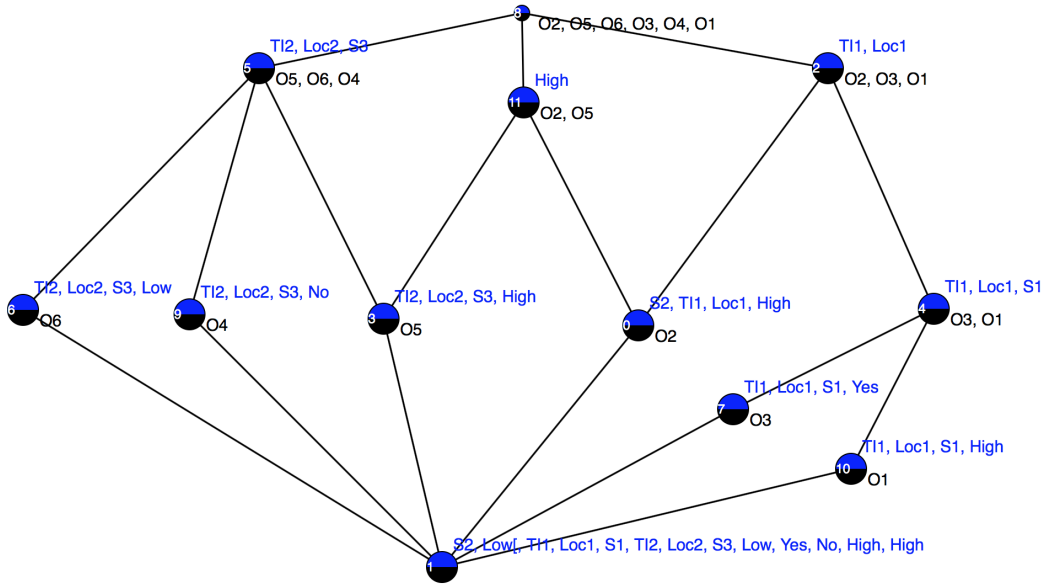


FIGURE 5.6: Fire Event Lattice

#### 5.5.2.4 Event Detection

The Feature-centric Event Detector module uses the previously generated lattice, an event detection rule (an operator that checks the lattice nodes in order to find the targeted events), and the features in order to detect feature-centric events (cf. Definition 12). We define a default event detection rule, as a set of lattice attributes that comply with the two conditions mentioned in Definition 12. The rule is extensible, thus allowing the integration of multiple event features (e.g., time, geo-location, temperature), each represented by the corresponding lattice attribute. Figure 5.7a shows a generic representation of the rule as a set of features and applied conditions. Key features are used in order to target the related feature-centric events. For example, the rule illustrated in Figure 5.7b detects an overheating event. The mandatory features are time, and geo-location. Additional features are added (i.e., sensor, temperature). Since the key feature is sensor, the detection rule will search for any temperature event detected by sensor $s_1$ regardless of the time, and location. Figure 5.7c detects the same event. However, this time location and sensor features are considered as key. This means that the detection rule will search for any temperature event that happened in $Shop_1$ regardless of time. These rules reflect the targeted events' definitions and are extensible. Features can be added/removed (e.g., temperature, $CO_2$ levels, smoke) for specific event detection needs.

The lattice[15] (cf. Figure 5.6) shows that the atomic event high temperature (cf. Figure 5.7b) is represented by node 10 having $(O1, \{S1, TI1, Loc1, High\})$. The atomic event high $CO_2$ is found in node 0 having $(O2, \{S2, TI1, Loc1, High\})$, and the smoke event is found in node 7 having $(O3, \{TI1, Loc1, S1, Yes\})$. Moreover, the composite event fire is the combination of the aforementioned primitive events. Therefore, it is found in their ancestor node 2 $(\{O1, O2, O3\}, \{TI1, Loc1\})$. This node represents the combination of the three events in the same time span and location. Therefore,

[15]The node numbering in the lattice is only used to refer to the nodes (the numbers are not sequential nor do they imply any particular ordering).

| Feature | Condition |
|---|---|
| *time* | *range* |
| *geo* | *range* |
| *sensor* | *range* |
| *contextual* $f_4$ | *range* |
| *contextual* $f_5$ | *range* |
| *key* $f_k$ | *specific value* |

(A) Generic detection rule

| Feature | Condition |
|---|---|
| *time* | *time range* |
| *geo* | *geo range* |
| $sensor_{key}$ | $s_1$ |
| *temperature* | *temp range* |

(B) Sensor-centric

| Feature | Condition |
|---|---|
| *time* | *time range* |
| $geo_{key}$ | $shop_1$ |
| $sensor_{key}$ | $s_1$ |
| *temperature* | *temp range* |

(C) Sensor/Geo-centric

FIGURE 5.7: Default detection rule

traversing the lattice towards the same spatio-temporal parent node reveals the composite event.

As a conclusion, the lattice could be used to detect the atomic events that interest a connected environment manager. Also, it could be further exploited for complex or composite event detection. Finally, for testing purposes, developers can change/add detection rules using the Rule Selector module. Since the lattice is not affected by the rule change, only the event detection step is repeated based on the new detection rule.

**Definition 12.** *A feature-centric Event, denoted* ***fce****, is a Formal Concept defined as a 4-tuple* $fce : \langle ffc, central_F, A, B \rangle$*, where:*

- *ffc is a FED Formal Context (Definition 11)*
- $central_F$ *is the set of selected key features* $|central_F \subseteq ffc.F$
- *A is a set of data objects* $| A \subseteq ffc.X$
- *B is a set of lattice attributes* $| B \subseteq ffc.Y$ *where* $\forall b_i, b_j \in B \wedge i \neq j$*:*
  - ***Condition 1:*** $b_i.f \neq b_j.f$
  - ***Condition 2:*** *if* $b_i.f.label = c_f.label | \forall c_f \in central_F$*, then* $dist(b_i.lat.a, so_j.v_k)=0 \mid \forall so_j \in A \wedge \forall v_k \in so_j.V$, $dt(b_i.lat.a) = dt(so_j.v_k)$. ■

Finally, Figure 5.8 details the interaction between the Rule Selector module and the Feature-centric Event Detector module. A detection rule change can be requested through the Event Query Language. One can create a new rule, select, or update an existing one. Based on the user's choice, the Rule Validator sub-module checks the syntax of the newly created or updated rule prior to storage. If one decides to select an existing rule, the Rule Selector sub-module returns the chosen rule. In both cases, the event detection is repeated using the chosen rule and new results are generated.

The following illustrates the syntax of a detection rule. The optional KEY notation indicates if a feature should be considered as centric (i.e., a key feature).

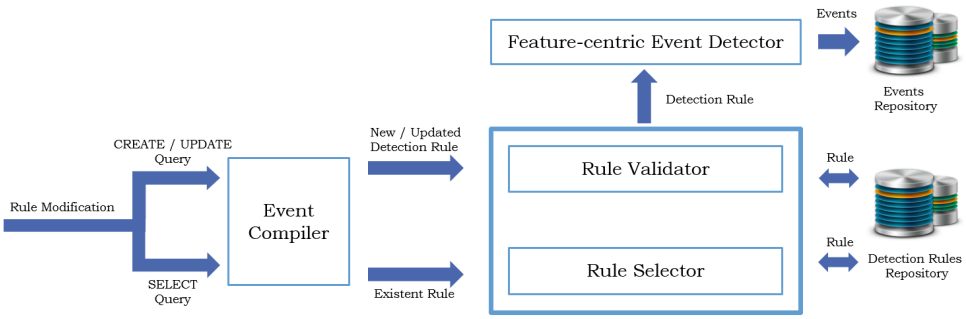


FIGURE 5.8: The Rule Selector module

**Defining the structure of a detection rule**

```
CREATE DETECTION RULE ( [ ID ] <dr_id>
[ , {
       ( [ FEATURE ] <feature_id> [KEY],
         [ CONDITION ] <condition_id> )
    }
] ) ;
```

## 5.6 Experimentation & Results

### 5.6.1 Evaluation Objectives

We implemented the event detection modules as a java library that could be easily integrated in the online platform that we are developing for event detection in connected environments. The global objectives were to evaluate the feasibility of applying eVM for sensor event detection, and the re-usability of the event detector in different contexts (e.g. social, conflict event detection). Therefore, we detected events in three different application domains/contexts: (i) sensor event detection in a smart building; (ii) social event detection on social media platforms; and (iii) armed conflict event detection from news data. These different contexts provide a variety of features each (sensor, social, and armed conflict related features), and different data objects (sensor observations in the building, multimedia data shared on social media, published multimedia data, such as videos and news stories related to wars/armed conflicts). More particularly, we aimed to validate the components related to the Event Detection part of the eVM framework (Event Detector modules cf. Fig.5.3). In order to do so, we measured, for each application (sensor, social, and conflict event detection), the event detection accuracy (with the integration and adaptation of FCA). We also evaluated the algorithm's performance based on execution time and memory consumption. The objectives of the experimentation were the following: (i) show that the approach is re-usable (Criterion 1) by detecting feature-centric events in different contexts (e.g., sensor events, social events, conflict events); (ii) measure the impact of domain specific expertise (Criterion 2) on the detection accuracy (regarding the choice of event features and granularity in the event definition phase) while reducing human intervention (Criterion 9); (iii) demonstrate that eVM is multi-modal (Criterion 6) and multi-source (Criterion 7) by measuring the impact of adding various features on the performance from various data sources; and (iv) proving that eVM is accurate when given (through the event queries) optimal

features/granularity, and event definition. We do not aim at comparing accuracy results with other works since the objective is to provide a re-usable, easy to integrate, accurate backbone for event detection, that can be adjusted/configured by users (i.e., by defining events, detection rules, features, granularity). We do not present here the evaluation results related to (i) the incremental processing (even though FCA is incremental [99]); (ii) evolution; (iii) extensibility; and (iv) ease of integration. This will be presented in a future work.

#### 5.6.1.1 Evaluation Scenarios

We evaluated the detection accuracy and the performance of the event detector in eVM. The performance tests were conducted on a machine equipped with an Intel i7 2.60 GHZ processor and 16 GB of RAM. The aim was to test the performance of our eVM Event Detector algorithm. To evaluate the re-usability of our approach, we detected re-iterated the aforementioned performance and accuracy experiments in three different contexts/scenarios:

- Application scenario 1: Sensor Event detection. In this scenario, we generated an environment where sensors are deployed. We modeled a smart building having different sensors. We generated 8400 observations during a one week span in the entire building. Then, we ran the event detector to find interesting feature-centric events. In this section, we detail the generated dataset, the experimental setup, and the accuracy results. We leave the performance evaluation for the other two scenarios (social event detection, and conflict event detection) where we used datasets that contain huge amounts of real data (i.e., the input size is considerably larger and tests the performance in a more complex setup).

- Application scenario 2: Social Event Detection. In this scenario, we use a publicly available dataset that contains real social data (e.g., images, videos) crawled from Flickr (a social media platform) in order to detect social feature-centric events (e.g., birthday, wedding). We test the detector with an input size of 60434 data objects, i.e., the event related data that need to be clustered into feature-centric events, were considered to be photos and videos taken/shared during the social events by participants. We detail the dataset, experimental setup, and accuracy/performance evaluation.

- Application scenario 3: Conflict Event Detection. In this scenario, we use a publicly available dataset that contains real armed conflict data. In this case, the targeted events were defined based on time, geo-location, and a set of contextual features such as the actors (i.e., aggressor, defender), the news source that covered the conflict, the conflict type (e.g., protest, war, planned attack), and finally the number of casualties. The data objects are considered news stories regarding the targeted conflict events. We detail the dataset, experimental setup, and accuracy/performance evaluation.

### 5.6.2 Sensor Event Detection

In the following, we detail the experimental protocol and generated dataset. Then, we discuss the event detection accuracy use cases, tests, and results. Finally, we end this section with a discussion that analyzes the obtained results.

#### 5.6.2.1 Experimental Setup & Dataset

**Connected Environment.** We simulated a smart building connected environment that hosts a sensor network. The sensed data are used for monitoring and management purposes. The Location map of the building is composed of three floors each containing a specific set of locations. Therefore, when defining and detecting events we considered two main granularity levels for the spatial feature (i.e., floor, and location).

**Sensor Network.** Sensors were deployed in each location in the building. For this example, we considered mainly five observable properties (temperature, humidity, $CO_2$, smoke, and noise). We considered a total 8400 sensor observations accumulated over a period of one week. Therefore, we considered week and day as the two main granularity levels for time. Each observation has an identifier and five attributes: (i) a timestamp; (ii) a geo-location; (iii) a sensor; (iv) a observable property (e.g., temperature, humidity, $CO_2$, smoke, noise); and an observation value. We only considered granularity levels for time and geo-location (i.e., the granularity is fixed for sensor, observable property, and observation value attributes).

**Targeted Events.** The purpose is to detect various atomic events. For instance, room overheating (high temperature), bad air quality (high $CO_2$), uncomfortable working environment (high humidity), safety hazards (high smoke levels), noise pollution (high noise). The events are heterogeneous in terms of the number of observations that constitute them (from 1 to hundreds per event) and are well distributed over the location map (i.e., events occur in every location in the building).

#### 5.6.2.2 Accuracy Evaluation

We chose to consider the following criteria for clustering quality evaluation. We calculated the F-score, based on the Recall (R) and Precision (PR), and the Normalized Mutual Information (NMI). These criteria are commonly adopted in information retrieval and event detection. A high F-score indicates a high quality of observation to event assignment while NMI was used to measure the information overlap between our clustering result and the ground truth data. Therefore, a high NMI indicates accurate clustering result.

**Use Cases.** Since we considered the time, geo-location, sensor, observable property, and observation as the five features that define our targeted events, we identified all possible combinations of the detection rule (cf. Table reftab:DR). In order to test granularity impacts, Table 5.5 sums up the different granularity combinations. When applying detection rules to granularity combinations, we get 124 use cases. We measured for each one the NMI and F-Score.

TABLE 5.5: Granularity Combinations - Sensor Event Detection

| Combination Number | Granularity (Time/Geo) |
|---|---|
| 1 | Week / Floor |
| 2 | Week / Location |
| 3 | Day / Floor |
| 4 | Day / Location |

TABLE 5.6: Detection Rule Combinations - Sensor Event Detection

| Combination Number | Number of Features | Considered Features in the Detection Rule |
|---|---|---|
| 1 | 5 | Time, Geo, Sensor, Property, Value |
| 2 | 4 | Time, Geo, Sensor, Property |
| 3 | | Time, Geo, Sensor, Value |
| 4 | | Time, Geo, Property, Value |
| 5 | | Time, Sensor, Property, Value |
| 6 | | Geo, Sensor, Property, Value |
| 7 | 3 | Time, Geo, Sensor |
| 8 | | Time, Geo, Property |
| 9 | | Time, Geo, Value |
| 10 | | Time, Sensor, Property |
| 11 | | Time, Sensor, Value |
| 12 | | Time, Property, Value |
| 13 | | Geo, Sensor, Property |
| 14 | | Geo, Sensor, Value |
| 15 | | Geo, Property, Value |
| 16 | | Sensor, Property, Value |
| 17 | 2 | Time, Geo |
| 18 | | Time, Sensor |
| 19 | | Time, Property |
| 20 | | Time, Value |
| 21 | | Geo, Sensor |
| 22 | | Geo, Property |
| 23 | | Geo, Value |
| 24 | | Sensor, Property |
| 25 | | Sensor, Value |
| 26 | | Property, Value |
| 27 | 1 | Time |
| 28 | | Geo |
| 29 | | Sensor |
| 30 | | Property |
| 31 | | Value |

**Results & Discussion** Results shown in Table 5.7, highlight the following:
*Detection Rule/Features Impact:* Detection rule 1 (i.e., based on time, geo, sensor, property, and value features) generates the highest NMI and F-score (NMI: 0.9994 and F-Score: 0.9987). It also exceeds all other detection rules in every granularity combination. However, detection rule 3 generates the same accuracy as detection rule 1 that relies on all features. This is due to the fact that in our generated dataset, each sensor observes only one property. This means that the sensor implies the observed property. Therefore, removing the feature property from detection rule 3 does not damage accuracy. To further test this, we considered an additional use case where sensors could observe more than one property. In this case, detection rule 1 surpasses detection rule 3. In contrast, when removing the value feature (e.g., detection rule 2) the accuracy decreases. This is explained by the fact that the property has various values (e.g., high, low, and moderate temperatures). Therefore, not considering the value in the detection rule leads to the lack of distinction between different events (e.g., overheating and under-heating events). To sum up the features impact, considering more features in the detection rule improves the accuracy. The aforementioned observations underline that eVM can cope with various features in the detection task. Thus, improving the event detection accuracy. Moreover, this highlights eVM's multi-modality, which allows the integration of additional features (having different datatypes) and the accurate detection of feature-centric events (in this case we the set of central features were sensor, property, and value).
*Granularity Impact:* The results improve, when the clustering is based on granularity closer to the ones used in the ground truth. For example, in the case of granularity (week, floor), the F-Score achieved based on time and geo features is 0.1446, but for the detection rule that considers only the sensor feature the F-Score is higher:

0.2295. This is because the granularity for time and geo are the most general (week and floor). Therefore, the impact factor of granularity is more important than that of the number of features considered in the detection rule. Some rules can exceed others for specific granularity combinations. For instance, the accuracy results of detection rule 17 for granularity combination day/location exceed the accuracy results of detection rule 1 for granularity combination week/floor. The best result can be achieved by considering the maximal number of features having correct granularity. This indicates that the granularity should not be fixed for all scenarios. When given the best granularity, our approach detects the feature-centric events very accurately.

### 5.6.3 Social Event Detection

#### 5.6.3.1 ReSEED Dataset

To evaluate the detection results, we used the ReSEED Dataset, generated during the Social Event Detection of MediaEval 2013 [81]. It contains real photos crawled from Flickr, that were captured during real social events which are heterogeneous in size (cf. Figure 5.9) and in topics (e.g., birthdays, weddings). The dataset contains 437370 photos assigned to 21169 events. In our evaluation, we used three event features: time, location, and social, since ReSEED photos have time, geo-location, and social attributes. In ReSEED, 98.3% of photos contain capture time information, while only 45.9% of the photos have a location. We had to select photos having these attributes from the dataset. This left us with 60434 photos from the entire dataset. In

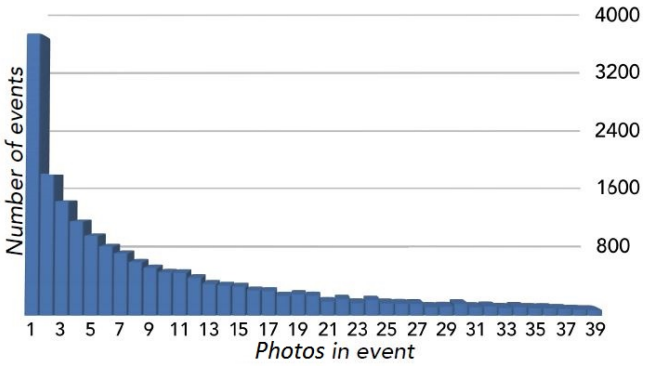


FIGURE 5.9: ReSEED Photo distribution

ReSEED, the ground truth used for result verification assigns photos to social events. Since, our approach is focused on feature-centric events (in this experimentation, user-centric events), we modified the ground truth to split the social events into their corresponding user-centric events. Since the splitting is based on the event features, we need to specify the feature granularity during the process. The latter are not specified in ReSEED, therefore we chose the lowest granularity values: day for time, street for geo-location, and photo creator name for social. The ground truth re-factoring process is described in Figure 5.10. First, we extracted the photos of each event in the ground truth. Second, we used the time stamps of photo capture to group photos by day. Third, we split the resulting clusters into distinct groups based on street values. Finally, the result was further split based on distinct photo creators.

TABLE 5.7: Accuracy Results - Sensor Event Detection

| Combination | Detection Rule | Measure | Granularity | | | |
|---|---|---|---|---|---|---|
| | | | Month | | Day | |
| | | | Floor | Location | Floor | Location |
| 1 | Time, Geo, Sensor, Property, Value | NMI | 0.6447 | 0.6555 | 0.9969 | **0.9994** |
| | | F-Score | 0.2470 | 0.2665 | 0.9919 | **0.9987** |
| 2 | Time, Geo, Sensor, Property | NMI | 0.6009 | 0.6126 | 0.9731 | **0.9758** |
| | | F-Score | 0.2295 | 0.2492 | 0.9508 | **0.9579** |
| 3 | Time, Geo, Sensor, Value | NMI | 0.6447 | 0.6555 | 0.9969 | **0.9994** |
| | | F-Score | 0.2470 | 0.2665 | 0.9919 | **0.9987** |
| 4 | Time, Geo, Property, Value | NMI | 0.6447 | 0.6555 | 0.9969 | **0.9994** |
| | | F-Score | 0.2470 | 0.2665 | 0.9919 | **0.9987** |
| 5 | Time, Sensor, Property, Value | NMI | 0.6447 | 0.6447 | 0.9969 | **0.9969** |
| | | F-Score | 0.2470 | 0.2470 | 0.9919 | **0.9919** |
| 6 | Geo, Sensor, Property, Value | NMI | 0.6447 | 0.6555 | 0.6447 | **0.6555** |
| | | F-Score | 0.2470 | 0.2665 | 0.2470 | **0.2665** |
| 7 | Time, Geo, Sensor | NMI | 0.6009 | 0.6126 | 0.9731 | **0.9758** |
| | | F-Score | 0.2295 | 0.2492 | 0.9508 | **0.9579** |
| 8 | Time, Geo, Property | NMI | 0.6009 | 0.6126 | 0.9731 | **0.9758** |
| | | F-Score | 0.2295 | 0.2492 | 0.9508 | **0.9579** |
| 9 | Time, Geo, Value | NMI | 0.4936 | 0.6445 | 0.9165 | **0.9962** |
| | | F-Score | 0.1563 | 0.2470 | 0.7438 | **0.9903** |
| 10 | Time, Sensor, Property | NMI | 0.6009 | 0.6009 | 0.9731 | **0.9731** |
| | | F-Score | 0.2295 | 0.2295 | 0.9508 | **0.9508** |
| 11 | Time, Sensor, Value | NMI | 0.6447 | 0.6447 | 0.9969 | **0.9969** |
| | | F-Score | 0.2470 | 0.2470 | 0.9919 | **0.9919** |
| 12 | Time, Property, Value | NMI | 0.6447 | 0.6447 | 0.9969 | **0.9969** |
| | | F-Score | 0.2470 | 0.2470 | 0.9919 | **0.9919** |
| 13 | Geo, Sensor, Property | NMI | 0.6009 | 0.6126 | 0.6009 | **0.6126** |
| | | F-Score | 0.2595 | 0.2492 | 0.2595 | **0.2492** |
| 14 | Geo, Sensor, Value | NMI | 0.6447 | 0.6555 | 0.6447 | **0.6555** |
| | | F-Score | 0.2470 | 0.2665 | 0.2470 | **0.2665** |
| 15 | Geo, Property, Value | NMI | 0.6447 | 0.6555 | 0.6447 | **0.6555** |
| | | F-Score | 0.2470 | 0.2665 | 0.2470 | **0.2665** |
| 16 | Sensor, Property, Value | NMI | 0.6447 | 0.6447 | 0.6447 | **0.6447** |
| | | F-Score | 0.2470 | 0.2470 | 0.2470 | **0.2470** |
| 17 | Time, Geo | NMI | 0.4394 | 0.6008 | 0.8887 | **0.9724** |
| | | F-Score | 0.1446 | 0.2295 | 0.7057 | **0.9491** |
| 18 | Time, Sensor | NMI | 0.6009 | 0.6009 | 0.9731 | **0.9731** |
| | | F-Score | 0.2295 | 0.2295 | 0.9508 | **0.9508** |
| 19 | Time, Property | NMI | 0.6009 | 0.6009 | 0.9731 | **0.9731** |
| | | F-Score | 0.2295 | 0.2295 | 0.9508 | **0.9508** |
| 20 | Time, Value | NMI | 0.0882 | 0.0882 | 0.7216 | **0.7216** |
| | | F-Score | 0.0548 | 0.0548 | 0.3291 | **0.3291** |
| 21 | Geo, Sensor | NMI | 0.6009 | 0.6126 | 0.6009 | **0.6126** |
| | | F-Score | 0.2295 | 0.2492 | 0.2295 | **0.2492** |
| 22 | Geo, Property | NMI | 0.6009 | 0.6126 | 0.6009 | **0.6126** |
| | | F-Score | 0.2295 | 0.2492 | 0.2295 | **0.2492** |
| 23 | Geo, Value | NMI | 0.4936 | 0.6445 | 0.4936 | **0.6445** |
| | | F-Score | 0.1563 | 0.2470 | 0.1563 | **0.2470** |
| 24 | Sensor, Property | NMI | 0.6009 | 0.6009 | 0.6009 | **0.6009** |
| | | F-Score | 0.2295 | 0.2295 | 0.2295 | **0.2295** |
| 25 | Sensor, Value | NMI | 0.6447 | 0.6447 | 0.6447 | **0.6447** |
| | | F-Score | 0.2470 | 0.2470 | 0.2470 | **0.2470** |
| 26 | Property, Value | NMI | 0.6447 | 0.6447 | 0.6447 | **0.6447** |
| | | F-Score | 0.2470 | 0.2470 | 0.2470 | **0.2470** |
| 27 | Time | NMI | 0.0000 | 0.0000 | 0.6828 | **0.6828** |
| | | F-Score | 0.0505 | 0.0505 | 0.3071 | **0.3071** |
| 28 | Geo | NMI | 0.4394 | 0.6008 | 0.4394 | **0.6008** |
| | | F-Score | 0.1446 | 0.2295 | 0.1446 | **0.2295** |
| 29 | Sensor | NMI | 0.6009 | 0.6009 | 0.6009 | **0.6009** |
| | | F-Score | 0.2295 | 0.2295 | 0.2295 | **0.2295** |
| 30 | Property | NMI | 0.6009 | 0.6009 | 0.6009 | **0.6009** |
| | | F-Score | 0.2295 | 0.2295 | 0.2295 | **0.2295** |
| 31 | Value | NMI | 0.0882 | 0.0882 | 0.0882 | **0.0882** |
| | | F-Score | 0.0548 | 0.0548 | 0.0548 | **0.0548** |

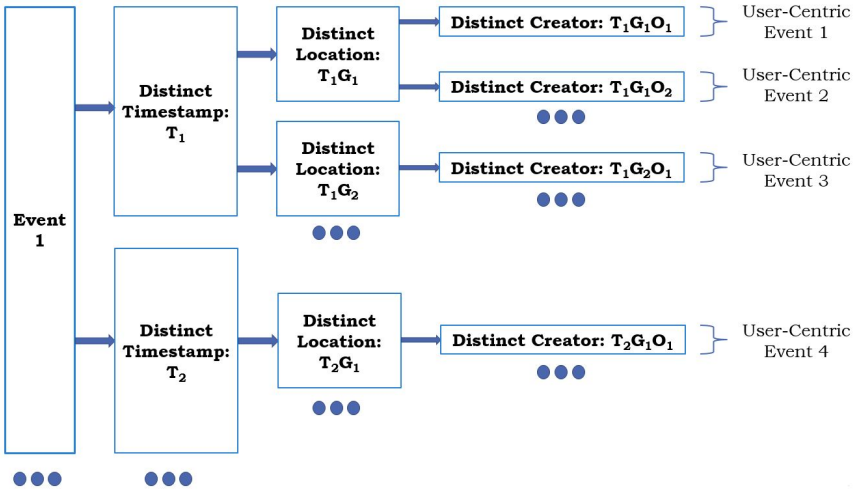


FIGURE 5.10: Re-factoring ground truth

#### 5.6.3.2 Performance Evaluation

We considered two criteria for this task: (i) total execution time and (ii) memory overhead.

**Use Cases.** The performance is highly affected by the number of photos, generated attributes, and clusters. We noticed that granularity day for time and street for geo-location generate more clusters and attributes than any other granularity combination. Therefore, we used day and street to test the prototype's performance in three worst case scenarios:

- Case 1: We selected the biggest event (1400 photos) as input. We varied the number of photos progressively from 1 to 1400. Since all photos are related to one event, the number of detected clusters should be one.
- Case 2: We extracted 400 events each having exactly one photo. We varied the number of photos from 100, 200, 300 to 400. The number of generated clusters for each iteration should be 100, 200, 300, and 400 respectively.
- Case 3: The goal is to test with as many photos as possible related to different events. We varied the number of photos from 15000, 30000, 45000 to 60434. Since thousands of events contain only one or two photos per event (worst case scenario), this case will generate the most clusters.

**Results & Discussion.** In Cases 1 and 2 (Figures 5.11.a and 5.11.b), where the number of photos does not exceed 1400 and 400 respectively, the total execution time is quasi-linear. However, in Case 3 (Figure 5.11.c), we clustered the entire dataset (60434 photos). The total execution time tends to be exponential, in accordance with the time complexity of FCA. When considering RAM usage, we noticed a linear evolution for the three cases (Figures 5.11.d, 5.11.e, and 5.11.f). RAM consumption is significantly higher in Case 2, where we generated 400 clusters, than in Case 1, where we generated one cluster. In Case 3, RAM consumption is the highest because both the number of photos at the input, and the number of generated clusters (detected events) were the highest. Other tests were conducted, Figure 5.12 (left) shows that low granularity (e.g., day) consume more execution time than high ones

(e.g., year). This is due to the generation of more lattice attributes and clusters. In addition, Figure 5.12 (right), shows that considering more features in the processing is also more time consuming. Nonetheless, the evolution from one to three features remains quasi-linear, making the process extensible.

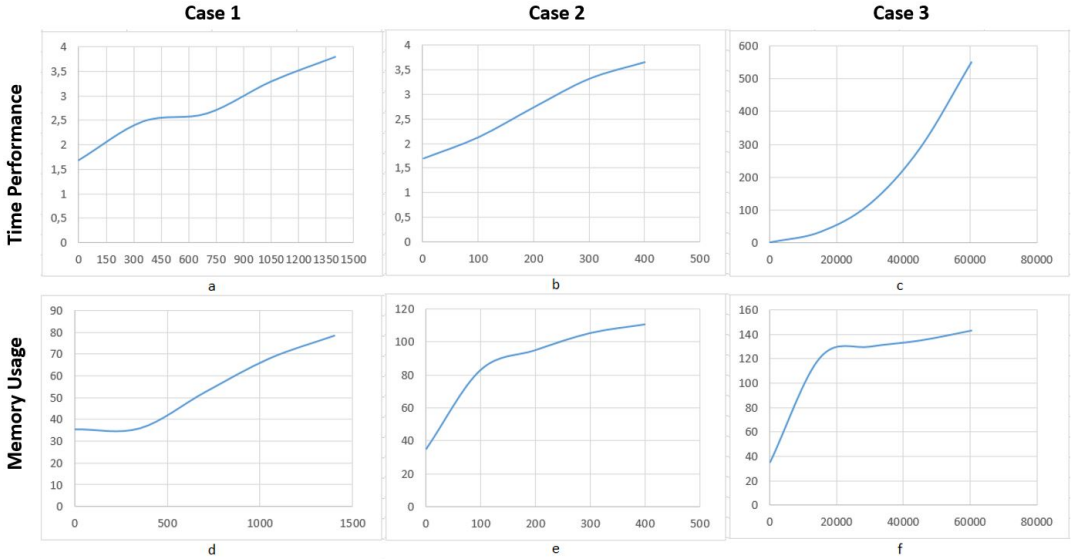


FIGURE 5.11: Performance Results

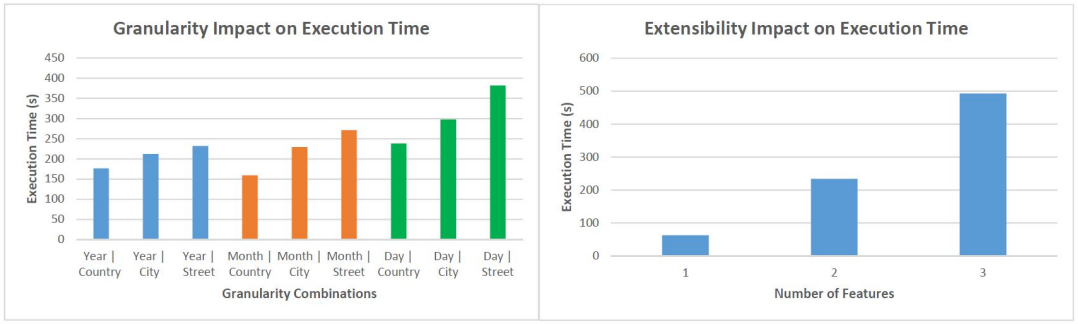


FIGURE 5.12: Granularity and Extensibility Impact

#### 5.6.3.3 Accuracy Evaluation

We chose to consider the criteria proposed by MediaEval for clustering quality evaluation. We calculated the F-score, based on the Recall (R) and Precision (PR), and the Normalized Mutual Information (NMI) using ReSEED's evaluation tool. These criteria are commonly adopted in information retrieval and social event detection. A high F-score indicates a high quality of photo to user-centric event assignment while NMI will be used to measure the information overlap between our clustering result and the ground truth data. Therefore, a high NMI indicates accurate clustering result.

**Use Cases.** Since we considered the time, geo, and social features, we identified all possible combinations of the detection rule (see Table 5.8). In order to test granularity impacts, Table 5.9 sums up the different granularity combinations. When applying

detection rules to granularity combinations, we get 63 use cases. We measured for each one the NMI and F-Score.

TABLE 5.8: Detection Rule Combinations - Social Event Detection

| Combination | Number of Features | Features Considered in the Detection Rule |
|---|---|---|
| 1 | 3 | Time, Geo, Social |
| 2 | 2 | Time, Geo |
| 3 | | Time, Social |
| 4 | | Geo, Social |
| 5 | 1 | Time |
| 6 | 1 | Geo |
| 7 | 1 | Social |

TABLE 5.9: Granularity Combinations - Social Event Detection

| Combination | Granularity: Time / Geo |
|---|---|
| 1 | Year / Country |
| 2 | Year / City |
| 3 | Year / Street |
| 4 | Month / Country |
| 5 | Month / City |
| 6 | Month / Street |
| 7 | Day / Country |
| 8 | Day / City |
| 9 | Day / Street |

**Results & Discussion.** Results shown in Table 5.10, highlight the following:

*Detection Rule/Features Impact:* The detection rule based on time, geo, and social features generates the highest NMI and F-score (NMI: 0.9999 and F-Score: 0.9995). It also exceeds all other detection rules (e.g., the one including solely time and geo features) in every granularity combination. This underlines that eVM can cope with various features such as the social feature in the detection task. Moreover, it highlights eVM's multi-modality, which allows the integration of additional features (having different datatypes) and the accurate detection of user-centric events.

*Granularity Impact:* The results improve, when the clustering is based on granularity closer to the ones used in the ground truth. For example, in the case of granularity (year, country), the F-Score achieved based on time and geo features is 0.1911, but for the detection rule that considers only the social feature the F-Score is higher: 0.5376. This is because the granularity for time and geo are the most general (year and country). Therefore, the impact factor of granularity is more important than that of the number of features considered in the detection rule. Some rules can exceed others for specific granularity combinations (e.g., {Time, Geo} exceeds {Time, Social} and {Geo, Social} for granularity Year/Month/Day-Street while {Time, Social} exceeds the other two rules for Year/Month/Day-Country). The best result can be achieved by considering the maximal number of features having correct granularity. This indicates that the granularity should not be fixed for all scenarios. When given the best granularity, our approach detects the user-centric events very accurately.

TABLE 5.10: Accuracy Results - Social Event Detection

| **Detection Rule** | **Measure** | **Granularity** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Year | | | Month | | | Day | | |
| | | Country | City | Street | Country | City | Street | Country | City | Street |
| Time Geo Social | F-Score | 0.6399 | 0.8180 | 0.8662 | 0.7964 | 0.8619 | 0.8948 | 0.9535 | 0.9742 | **0.9995** |
| | NMI | 0.9181 | 0.9602 | 0.9729 | 0.9549 | 0.9703 | 0.9789 | 0.9880 | 0.9938 | **0.9999** |
| Time Geo | F-Score | 0.1911 | 0.7678 | 0.8473 | 0.4943 | 0.8367 | 0.8821 | 0.8854 | 0.9542 | 0.9892 |
| | NMI | 0.7113 | 0.9475 | 0.9684 | 0.8707 | 0.9637 | 0.9759 | 0.9729 | 0.9894 | 0.9977 |
| Time Social | F-Score | 0.6245 | 0.6245 | 0.6245 | 0.7939 | 0.7939 | 0.7939 | 0.9534 | 0.9534 | 0.9534 |
| | NMI | 0.9143 | 0.9143 | 0.9143 | 0.9544 | 0.9544 | 0.9544 | 0.9879 | 0.9879 | 0.9879 |
| Geo Social | F-Score | 0.5085 | 0.7718 | 0.8357 | 0.5085 | 0.7718 | 0.8357 | 0.5085 | 0.7718 | 0.8357 |
| | NMI | 0.8742 | 0.9470 | 0.9653 | 0.8742 | 0.9470 | 0.9653 | 0.8742 | 0.9470 | 0.9653 |
| Time | F-Score | 0.0220 | 0.0220 | 0.0220 | 0.1399 | 0.1399 | 0.1399 | 0.7278 | 0.7278 | 0.7278 |
| | NMI | 0.3971 | 0.3971 | 0.3971 | 0.7069 | 0.7069 | 0.7069 | 0.9392 | 0.9392 | 0.9392 |
| Geo | F-Score | 0.0559 | 0.6958 | 0.8343 | 0.0559 | 0.6958 | 0.8343 | 0.0559 | 0.6958 | 0.8343 |
| | NMI | 0.5084 | 0.9241 | 0.9646 | 0.5084 | 0.9241 | 0.9646 | 0.5084 | 0.9241 | 0.9646 |
| Social | F-Score | 0.5376 | 0.5376 | 0.5376 | 0.5376 | 0.5376 | 0.5376 | 0.5376 | 0.5376 | 0.5376 |
| | NMI | 0.8755 | 0.8755 | 0.8755 | 0.8755 | 0.8755 | 0.8755 | 0.8755 | 0.8755 | 0.8755 |

### 5.6.4 Conflict Event Detection

#### 5.6.4.1 ACLED Dataset

The aforementioned experiments targeted feature-centric social events which have features such as time, location, social (e.g., participants), and topics (e.g., birthday, marriage). In the following experiments, we targeted feature-centric conflict events. The latter, have different features (e.g., time, location, aggressor, defender, press news source, conflict type, casualties). This allowed to have a different event definition, as well as test the accuracy of the detection process in different contexts. For this purpose, we used the ACLED[16] (Armed Conflict Location & Event Data Project) Dataset. It is a dis-aggregated conflict collection, analysis and crisis mapping project. ACLED collects the dates, actors, types of violence, locations, and fatalities of all reported political violence and protest events across Africa, South Asia, South East Asia and the Middle East. Political violence and protest include events that occur within civil wars and periods of instability, public protest and regime breakdown. The dataset contains event records that span over years. We selected 49000 events, from the African subset, that date from April 1998 till January 2018. For each event, we generated shared objects having twelve attributes each (a object owner (press news source), a latitude, a longitude, a country, a region, a city, a street, a datetime, a conflict type, two actors, and the number of casualties). In total, we tested the accuracy of the detection process, the impact of the number of included features on the performance based on an input of 50000 shared objects related to the events mentioned above.

#### 5.6.4.2 Performance Evaluation

To give a detailed view of the algorithm's performance, we measured the impact of: (i) the number of objects at the input; and (ii) the number of included features on the execution time.

**Use Cases.** The performance is affected by the input size, i.e., the number of objects to be processed, and the number of event features included in the clustering. Therefore, we experimented the following cases:

- Case 1: We ran the detection module five times and measured the execution time of each run. Every iteration has the following configuration: (i) all seven features are considered in the clustering; and (ii) the granularity choices for time and geo-location are day and street respectively. The only variable is the input size. The first run processes 10000 objects, the second 20000, the third run 30000, then we considered 40000 in the fourth run, and finally 50000 in the last run.

- Case 2: We ran the detection module seven times and measured the execution time of each run. Every iteration has the following configuration: (i) the input size is the same, 50000 objects (the entire dataset); (ii) the granularity choices for time and geo-location are day and street respectively. In the first run, we only consider one feature (time) in the clustering. Then, for each iteration, we include one additional feature (e.g., time and geo-location in the second run). The last run includes all seven features (time, geo-location, press news source, actor 1, actor 2, conflict type, and casualties).

[16] https://www.acleddata.com

**Results & Discussion.** Figure 5.13 shows the impact of augmenting the size of the input on the algorithm's execution time (Case 1). We notice that the evolution of execution time is quasi-linear. Figure 5.14 shows the impact of including more features in the processing on the execution time (Case 2). The evolution is also quasi-linear in a worst case scenario from a granularity and input size point of view. We also analyzed the execution time of each step of the event detector (i.e., attribute extraction, lattice construction, event detection cf. Section 5.5). The highest cost in terms of execution time is related to the event detection step (not FCA computation), which consists of scrolling through the nodes of the lattice in order to select nodes that comply with the chosen feature-centric event definition. Results can be optimized by looking into better ways of scrolling through the lattice (graph analysis techniques). This will be conducted in a future work.

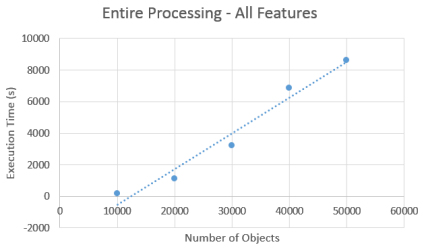


FIGURE 5.13: Case 1 Results

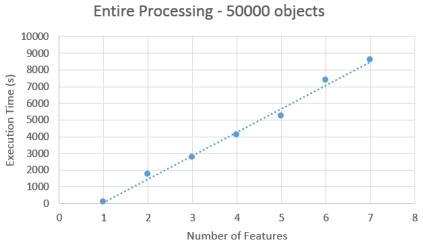


FIGURE 5.14: Case 2 Results

#### 5.6.4.3 Accuracy Evaluation

We used the same metrics (F-Score & NMI) for accuracy evaluation. We tested the same granularity combinations shown in Table 5.9 for time and geo-location features. We did not vary the granularity of the five other features (press news source, actor 1, actor 2, conflict type, and casualties) found in the ACLED dataset. Finally, we used the same tool (provided by the ReSEED work) for F-Score and NMI calculation. Accuracy results are detailed in Table 5.12.

**Use Cases.** We limited the detection rule combinations to three features: (i) time; (ii) geo-location; and (iii) press news source (chosen as central feature in this experimentation). This could be extended to include the actors of events (e.g., to detect the group most involved in armed conflicts), the number of casualties (e.g., to detect the deadliest conflict events), and the conflict types (e.g., to compare occurrence of protests between countries). The detection rule combinations are detailed in Table 5.11.

TABLE 5.11: Detection Rule Combinations - Conflict Event Detection

| Combination | Number of features | Considered features |
|---|---|---|
| 1 | 3 | Time, Geo, Press News Source |
| 2 | 2 | Time, Geo |
| 3 | | Time, Press News Source |
| 4 | | Geo, Press News Source |
| 5 | 1 | Time |
| 6 | 1 | Geo |
| 7 | 1 | Press News Source |

**Results & Discussion.** Table 5.12 highlights the following:

*Detection Rule/Features Impact:* The detection rule 1 (based on time, geo-location, and press news source features) generates the highest NMI and F-score (NMI: 0.9949 and F-Score: 0.9631). It also exceeds all other detection rules (e.g., the one including solely time and geo-location features) in every granularity combination. This underlines that eVM can provide the appropriate way to conduct the clustering and integrate various datatypes related to a multitude of features due to its multi-modality. However, accuracy can still be improved, few objects were assigned to the wrong clusters. These errors can be minimized by including more features in the detection rule (i.e., the actors, conflict type, and casualties).

*Granularity Impact:* We notice here (as we also did for the ReSEED dataset) that when the clustering is closer in terms of granularity to the ground truth, accuracy improves. For example, in the case of granularity {year, country}, the F-Score achieved based on time and geo-location features is 0.0254, but for the detection rule that considers only the press news source feature the F-Score is higher: 0.1938. This is because the granularity for time and geo are the most general (year and country). Some rules can exceed others for specific granularity combinations. The best result can be achieved by considering the maximal number of features having correct granularity. Finally, these results prove that eVM is re-usable since event detection accuracy was high in different event detection contexts.

TABLE 5.12: Accuracy Results - Conflict Event Detection

| Detection Rule | Measure | Granularities | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Year | | | | Month | | | | Day | | | |
| | | Country | Region | City | Street | Country | Region | City | Street | Country | Region | City | Street |
| 1 | F-Score | 0.3194 | 0.6181 | 0.6737 | 0.732 | 0.5012 | 0.7598 | 0.7873 | 0.8195 | 0.8439 | 0.9454 | 0.9528 | **0.9631** |
| | NMI | 0.8234 | 0.9287 | 0.9392 | 0.9515 | 0.9011 | 0.9604 | 0.9645 | 0.9701 | 0.9764 | 0.9923 | 0.9933 | **0.9949** |
| 2 | F-Score | 0.0254 | 0.3366 | 0.4385 | 0.5465 | 0.1789 | 0.6257 | 0.6771 | 0.7374 | 0.7252 | 0.921 | 0.9325 | 0.9477 |
| | NMI | 0.6652 | 0.8668 | 0.8885 | 0.9144 | 0.8263 | 0.9382 | 0.9461 | 0.9567 | 0.9598 | 0.9889 | 0.9904 | 0.9928 |
| 3 | F-Score | 0.2564 | 0.2564 | 0.2564 | 0.2564 | 0.429 | 0.429 | 0.429 | 0.429 | 0.8269 | 0.8269 | 0.8269 | 0.8269 |
| | NMI | 0.7892 | 0.7892 | 0.7892 | 0.7892 | 0.886 | 0.886 | 0.886 | 0.886 | 0.9742 | 0.9742 | 0.9742 | 0.9742 |
| 4 | F-Score | 0.232 | 0.5028 | 0.5751 | 0.6522 | 0.232 | 0.5028 | 0.5751 | 0.6522 | 0.232 | 0.5028 | 0.5751 | 0.6522 |
| | NMI | 0.7509 | 0.897 | 0.9133 | 0.9327 | 0.7509 | 0.897 | 0.9133 | 0.9327 | 0.7509 | 0.897 | 0.9133 | 0.9327 |
| 5 | F-Score | 0.001 | 0.001 | 0.001 | 0.001 | 0.0116 | 0.0116 | 0.0116 | 0.0116 | 0.291 | 0.291 | 0.291 | 0.291 |
| | NMI | 0.4261 | 0.4261 | 0.4261 | 0.4261 | 0.6671 | 0.6671 | 0.6671 | 0.6671 | 0.8941 | 0.8941 | 0.8941 | 0.8941 |
| 6 | F-Score | 0.0015 | 0.0865 | 0.1809 | 0.311 | 0.0015 | 0.0865 | 0.1809 | 0.311 | 0.0015 | 0.0865 | 0.1809 | 0.311 |
| | NMI | 0.4209 | 0.7434 | 0.7858 | 0.836 | 0.4209 | 0.7434 | 0.7858 | 0.836 | 0.4209 | 0.7434 | 0.7858 | 0.836 |
| 7 | F-Score | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 | 0.1938 |
| | NMI | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 | 0.6913 |

## 5.7 Summary

Event Detection is an essential part of many applications/services from various application domains (e.g., home/building management, energy management, navigation systems, industry and manufacturing, medical). All these approaches are task-centric, and designed for a specific application domain/purpose. However, these works do not share a common re-usable backbone for event detection that can be instantiated/used in different contexts. In this chapter, we propose a generic framework, the event virtual machine (eVM), for feature-centric event detection. Our approach uses event queries to target specific events, thus creating a specific instance for each use case using the same framework. The detection part is based on Formal Concept Analysis (FCA), an incremental and dynamic clustering technique. We developed a prototype for testing purposes. The results show that our approach achieved high accuracy in most cases, especially when additional features (in addition to time and location) are considered. Results also proved that eVM is re-usable and multi-modal.

# Chapter 6

# Conclusion & Future Work

*"Education is the most powerful weapon which you can use to change the world."*

*— Nelson Mandela*

## 6.1 Report Recap

The study presented in this thesis focuses mainly on event detection in connected environments.

**In Chapter 1** we give the reader an insight on why connected environments are considered a topic of interest nowadays. Then, we focus on this thesis's objectives of modeling and querying such environments as well as coping with their dynamicity, and detecting specific events that occur within their premises. We present a smart mall connected environment scenario that illustrates the motivation behind this work and the challenges that lie ahead. Then, we briefly present our proposed framework for Event Detection in Connected Environments (EDCE) where each module answers a specific objective and addresses a set of needs and challenges. Finally, we list the publications related to this report before introducing the following chapters.

**In Chapter 2** we present an ontology-based data model for Hybrid Semantic Sensor Networks (HSSN) where we address two major components of the connected environment data model: (i) the environment description; and (ii) the sensor network modeling. We review related works on the topic, before proposing an extension of the SSN ontology to include a wider set of divers sensors (e.g., static, mobile, simple nodes, multi-sensor devices, scalar sensors, multimedia sensors). Moreover, HSSN enriches the description of different platform types (e.g., infrastructures, devices), and sensed data (e.g., scalar, multimedia sensor observations). The extension does not add domain specific knowledge to the ontology to ensure it could be reused in various application domains. Then, we evaluate the accuracy of our additions, their clarity, consistency, and the overall impact on performance.

**In Chapter 3** we detail the query language that one uses to interact with a connected environment. We propose an event query language specifically designed for connected environments, denoted EQL-CE. After presenting the motivations behind this contribution and some related works, we propose a three layered framework for the language that (i) considers all parts of the connected environment data model (i.e., the environment, sensor network, events, and application domain); (ii)

provides various query types (e.g., for component definition, instance/data management); and (iii) covers various datatypes in the queries. Moreover, one could optimize the queries by enriching them with spatial or/and temporal distribution functions and constraints. Finally, the EQL-CE syntax is generic, written in EBNF, and could be parsed to several domain specific languages (e.g., SQL, SPARQL).

**In Chapter 4** we focus on one specific need of EQL-CE: coping with the connected environment dynamicity. We motivate our proposal by showing how event definition queries could become obsolete over time due to sensor mobility and breakdowns as well as loss of data/features. Then, we propose a way to automatically detecting obsolete queries and rewriting them in order to ensure they evolve with the environment. When rewriting an obsolete query, we aim to replace unavailable data sources (i.e., sensors) and data/features with other similar ones. To do so, we propose similarity measures that take into account various attributes (e.g., spatial, temporal, property similarity). Finally, we analyze the complexity of both algorithms and propose an experimental protocol for their evaluation (in terms of accuracy and performance).

**In Chapter 5** we propose an event detector, denoted eVM (event Virtual Machine). This module requires an event query (event definition) and sensed data to detect targeted feature-centric events. We start by reviewing related work on event detection, and presenting some background on Formal Concept Analysis (FCA), the clustering technique that the detector relies on. Then, we detail the event detection process before extensively testing the detector in different contexts (e.g., sensor, social, and conflict event detection). We evaluate the event detction accuracy and performance.

## 6.2 Future Research Directions

Various improvements still need to be considered for this work. We detail future research directions for each contribution separately.

### 6.2.1 A Data Model For Hybrid Connected Environments

**Completeness Evaluation.** We would like to continue the ongoing evaluation of the completeness of the HSSN ontology through comparisons with mobility, sensor, and environment taxonomies. This evaluation will potentially help us discover missing concepts or properties that could complement HSSN.

### 6.2.2 An Event Query Language For Connected Environments

**EQL-CE Implementation.** We are developing an online simulator to allow users to run tests on a connected environment (e.g., the smart mall). Currently, we have to finish the development on two simultaneous projects: (i) the EQL-CE parsing engine that converts EBNF queries to specific data model specific syntax (e.g., SQL, SPARQL); and (ii) integrating the parsed queries with a query execution engine. The implementation has the highest priority from all future work, since it allows the evaluation of EQL-CE and thereafter the end to end evaluation of the entire EDCE framework.

**EQL-CE Evaluation.** Once the development is complete, we aim to evaluate the accuracy of the parsing and query execution functions. Moreover, we need to evaluate the performance in terms of query run-time and resource consumption (e.g., RAM, CPU usage). Finally, we need to quantify the cost (in terms of number of steps/queries) required for various objectives (e.g., defining a platform, an entire environment, deploying sensors).

**Developing Distribution Queries.** We would like to develop the query optimizer by integrating advanced spatial/temporal distribution functions to the queries for specific event definitions. The objective here is to provide the user with three query types:

- Range queries that allow users to select all sensors within a specific range of a point in space or all sensor observations within a specific range of a point in time.
- Nearest Neighbour queries that allow users to select the k nearest sensors/sensor observations to a specific spatial/temporal entity.
- Distribution queries that allow users to import an external mathematical distribution function and apply on the spatial distribution of sensors or the temporal distribution of sensor observations (e.g., selecting evenly distributed sensors over a specific location/area, selecting evenly distributed sensing rates for various sensor observations).

### 6.2.3 Handling Connected Environment Dynamicity

**Experimentation & Evaluation.** We need to finish the implementation of the two main algorithms proposed in Chapter 4 (i.e., Algorithms 1 and 2) in order to later evaluate the accuracy and performance of the query rewriting engine. We would like to evaluate our query re-writing engine and test it in various scenarios by running the experiments described in Section 4.6.

### 6.2.4 A Generic Event Detection Framework

**Optimizing Detection Results.** The experimentation results show that the chosen granularity for any given feature impacts the detection accuracy. Therefore, we are investigating the detection of optimal granularity combinations based on the event definitions and input data distribution. Furthermore, even when granularity combinations are optimal for the detection task, some data objects are associated to wrong clusters. This is due to their temporal or spatial closeness to more than one cluster. Therefore, automatically considering temporal/spatial distances between clusters could help improve the detection accuracy. To do so, matrix-based intersection models [95], that include spatial and temporal cluster attributes could potentially help reduce the number of wrong data object to cluster associations. Moreover, integrating data cleaning, and noise handling techniques at the pre-processing phase could also alleviate the issue.

**Composite Event Detection.** In the eVM description (cf. Chapter 5), we show how the FCA lattice could be exploited and traversed to go beyond the detection of feature-centric atomic events and reach the composite high level events. For this purpose, we aim to develop a composite event detection module. We are currently

investigating Relational Concept Analysis (RCA) [83] to explore the relations between lattice nodes in order to find the parent composite event of any given set of atomic feature-centric events.

# Appendix A

# HSSN Implementation Details

Table A.1 lists the contributing ontologies used in our proposal. Moreover, Tables A.2 and A.3 detail the added HSSN object properties and concepts respectively.

TABLE A.1: Concept/Object Property Summary

| Ontology | Added Concepts Count | Added Object Properties Count |
|---|---|---|
| SOSA [44] | 14 | 21 |
| SSN [44] | 6 | 15 |
| SSN-Systems [44] | 23 | 8 |
| HSSN [68] | 35 | 54 |
| MSSN [6] | 51 | 20 |
| TIME [51] | 6 | 23 |
| FOAF [20] | 1 | 0 |
| VOAF [100] | 1 | 0 |
| **Total** | **137** | **141** |

TABLE A.2: HSSN added Properties

| Object Property | Domain | Range | Inverse Property |
|---|---|---|---|
| centers | SensingLocation | CoverageArea | isCenteredAround |
| composes | Location | LocationMap | isComposedOf |
| currentlyCovers | Sensor | CoverageArea | isCurrentlyCoveredBy |
| currentlyLocates | Location | Sensor | isCurrentlyLocatedAt |
| definesHorizontalSpreadOf | HorizontalSensingAngle | CoverageArea | hasHorizontalSpreadDefinedBy |
| definesVerticalSpreadOf | VerticalSensingAngle | CoverageArea | hasVerticalSpreadDefinedBy |
| delimits | SensingRange | CoverageArea | isDelimitedBy |
| describes | LocationMap | Infrastructure | isDescribedBy |
| hasCapability | Service | Capability | ∅ |
| hasCoverageTime | CoverageArea | TemporalEntity | ∅ |
| hasEmbeddedSystem | ExpansionCard | System | isEmbeddedOn |
| hasExpansionCard | Hardware | ExpansionCard | isExpansionCardOf |
| hasHardware | Device | Hardware | isRelatedToDevice |
| hasInterfaces | Service | Interface | ∅ |
| hasLocationTime | Location | TemporalEntity | ∅ |
| hasMemory | Hardware | Memory | ∅ |
| hasMetadata | Service | Metadata | ∅ |
| hasMultimediaValue | MultimediaProperty | MultimediaValue | isMultimediaValueOf |
| hasNetworkInterface | Hardware | NetworkInterface | ∅ |
| hasPastCoverageArea | MobileSensor | CoverageArea | isPreviouslyCoveredBy |
| hasPastLocation | MobileSensor | Location | hasPreviouslyLocated |
| hasPowerSupply | Hardware | PowerSupply | ∅ |
| hasProcessor | Hardware | Processor | ∅ |
| hasScalarValue | ScalarProperty | ScalarValue | isScalarValueOf |
| hasSensingTime | ObservationValue | TemporalEntity | ∅ |
| hasSoftware | Device | Software | ∅ |
| hasSubMap | LocationMap | LocationMap | ∅ |
| hasVariables | Service | Variables | ∅ |
| includes | CoverageArea | Location | isIncludedIn |
| isMultimediaSensedBy | MultimediaProperty | Sensor | mediaSenses |
| isProducedBy | ObservationValue | Observation | produces |
| isProvidedBy | Service | Device | provides |
| isRequestedBy | Capability | UserGoal | ∅ |
| isScalarSensedBy | ScalarProperty | Sensor | scalarSenses |

TABLE A.3: HSSN added Concepts

| Concept | Description |
|---|---|
| Capability | The functionality provided by a Service is described by its Capability |
| Coverage Area | A Coverage Area is a geographical zone that includes a set of Locations within the range of at least one sensor |
| Device | A Device is an electronic equipment capable of one or more computing functions and supporting the installation of firmware or third-party software |
| Expansion Card | The Expansion Card is a plugin (additional) card. Systems (e.g., sensors) are deployed on the Expansion Card of a Device |
| Hardware | A Hardware is a component of a Device responsible of specific functionality |
| HorizontalAngle | HorizontalAngle defines the trigonometric horizontal spread of a system's capability |
| HorizontalSensingAngle | The maximal horizontal sensing angle provided by the sensor |
| Infrastructure | An infrastructure is a real world physical infrastructure or environment where a sensor network might be deployed |
| Input | Input is the set of variables and constraints required at the input end of a service in order to achieve a correct execution of the service's functionality (capability) |
| Interface | The interface represents how a service can be contacted by / connected to other entities such as service requesters or other services |
| Memory | The set of all memories of a Device |
| Metadata | Metadata represent the properties that describe a Service |
| MobileSensor | A Mobile Sensor is a sensor that has the ability to move around, and change position in time |
| MultimediaProperty | A Property of type multimedia |
| MultimediaValue | An ObservationValue of type Multimedia |
| NetworkInterface | The Network Interface is part of the Hardware responsible for communication |
| ObservationValue | Specifies the actual concrete value generated due to an Observation |
| Output | Output is the set of variables and effects generated by a service after a correct execution of the service's functionality (capability) |
| PowerSupply | Power Supply is the Hardware responsible for providing power in order to allow the Device optimal functioning |
| Processor | The Processor is the Hardware responsible for the processing of data in the Devices |
| Range | The range is the maximal distance (in meters) that delimits a system's operational capability. i.e., The maximal distance that delimits a sensor's sensing capability, an actuator's actuation capability, a sampler's sampling capability |
| ScalarProperty | A Property of type Scalar |
| ScalarValue | An ObservationValue of type Scalar |
| SensingLocation | The SensingLocation is the location of the sensor that covers this CoverageArea |
| SensingRange | The maximal sensing distance provided by the sensor |
| Service | A Service represents entities capable of providing some value (or a set of values) relative to a given domain |
| ServiceInteractionInterface | A ServiceInteractionInterface is used to connect a Service to other Services for composite Service orchestration |
| Software | The set of all operating systems, applications and programs running on a Device |
| StaticSensor | A Static Sensor is a sensor that does not move, or change position after its deployment |
| Textual | Datatype Category of a ScalarValue |
| UserGoal | Goals describe user desires. They provide the means to specify user objectives |
| UserInteractionInterface | The UserInteractionInterface is used by users who request services to allow them the exchange of requests/information/variables with a service |
| Variables | Variables represent the variables that are used / generated by a service |
| VerticalAngle | VerticalAngle defines the trigonometric vertical spread of a system's capability |
| VerticalSensingAngle | The maximal vertical sensing angle provided by the sensor |

# Bibliography


[1] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-efficiency in Building (BuildSys)*, pages 1–6, 2010.

[2] C. C. Aggarwal. *Managing and mining sensor data*. Springer Science & Business Media, 2013.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[4] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st ACM Annual International Conference on Research & Development in Information Retrieval (SIGIR)*, pages 37–45, 1998.

[5] G. e. a. Amato. Querying moving events in wireless sensor networks. *Pervasive & Mobile Computing*, 16:51–75, 2015.

[6] C. Angsuchotmetee, R. Chbeir, and Y. Cardinale. MSSN-Onto: An ontology-based approach for flexible event processing in multimedia sensor networks. *Future Generation Computer Systems*, 2018.

[7] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: A unified language for event processing and stream reasoning. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 635–644, 2011.

[8] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. Etalis: Rule-based reasoning in event processing. In *Reasoning in event-based distributed systems*, pages 99–124. Springer, 2011.

[9] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[10] S. Avancha, A. Joshi, and C. e. a. Patel. Ontology-driven adaptive sensor networks. In *Proceedings of the 1st Annual International Conference on Mobile & Ubiquitous Systems: Networking & Services (MOBIQUITOUS)*, pages 194–202, 2004.

[11] R. Baeza-Yates and B. e. a. Ribeiro-Neto. *Modern information retrieval*. ACM press, 1999.

[12] M. Bahrepour, N. Meratnia, and P. J. Havinga. Sensor fusion-based event detection in wireless sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services (MobiQuitous)*, pages 1–8, 2009.

[13] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 1061–1062, 2009.

[14] R. S. Barga and H. Caituiro-Monge. Event correlation and pattern detection in CEDR. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, pages 919–930, 2006.

[15] P. Barnaghi, S. Meissner, M. Presser, and K. Moessner. Sense and sens'ability: Semantic data modelling for sensor networks. In *Proceedings of the ICT Mobile Summit*, 2009.

[16] M. Bendersky, W. B. Croft, and D. A. Smith. Two-stage query segmentation for information retrieval. In *Proceedings of the 32nd ACM International Conference on Research & Development in Information Retrieval (SIGIR)*, pages 810–811, 2009.

[17] S. Bergsma and Q. I. Wang. Learning noun phrase query segmentation. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing & Computational Natural Language Learning (EMNLP-CoNLL)*, pages 819–826, 2007.

[18] P. Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[19] M. Botts and A. Robin. OpenGIS sensor model language (SensorML) implementation specification. *OpenGIS Implementation Specification OGC*, 7, 2007.

[20] D. Brickley. FOAF vocabulary specification. `http://rdfweb.org/foaf/`, 2005.

[21] F. Bry and M. Eckert. Rule-based composite event queries: The language XChange EQ and its semantics. In *Proceedings of the International Conference on Web Reasoning & Rule Systems (RR)*, pages 16–30, 2007.

[22] A. Buckman, M. Mayfield, and S. BM Beck. What is a smart building? *Smart & Sustainable Built Environment*, 3(2):92–109, 2014.

[23] P. Burmeister. *Formal concept analysis with ConImp: Introduction to the basic features*. Fachbereich Mathematik, Technische Universität Darmstadt, 2003.

[24] J.-P. Calbimonte, O. Corcho, and A. J. Gray. Enabling ontology-based access to streaming data sources. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, pages 96–111, 2010.

[25] C. Carpineto, R. De Mori, G. Romano, and B. Bigi. An information-theoretic approach to automatic query expansion. *ACM Transactions on Information Systems*, 19(1):1–27, 2001.

[26] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[27] V. Choi. Faster algorithms for constructing a concept Galois lattice. In *Clustering challenges in biological networks*, page 169. 2006.

[28] C.-Y. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003.

[29] W. W. Chu and Q. Chen. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge & Data Engineering*, 6(5):738–749, 1994.

[30] M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, 2012.

[31] S. Devarakonda, P. Sevusu, H. Liu, R. Liu, L. Iftode, and B. Nath. Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing (UrbComp)*, page 15, 2013.

[32] D. M. Doolin and N. Sitar. Wireless sensors for wildfire monitoring. In *Smart Structures and Materials 2005: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, volume 5765, pages 477–485, 2005.

[33] N. Durand, S. Derivaux, G. Forestier, C. Wemmert, P. Gançarski, O. Boussaid, and A. Puissant. Ontology-based object recognition for remote sensing image interpretation. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 472–479, 2007.

[34] M. Eid, R. Liscano, and A. El Saddik. A universal ontology for sensor networks data. In *Proceedings of the IEEE International Conference on Computational Intelligence for Measurement Systems & Applications (CIMSA)*, pages 59–62, 2007.

[35] EsperTech. EsperTech. Chapter 5. EPL reference:Clauses. `http://esper.espertech.com/release-5.3.0/esper-reference/html_single/index.html#epl_clauses`. Accessed: 2019-02-07.

[36] G. Fokou, S. Jean, A. Hadjali, and M. Baron. Cooperative techniques for sparql query relaxation in rdf databases. In *Proceedings of the 12th European Semantic Web Conference (ESWC)*, pages 237–252, 2015.

[37] R. Frosini, A. Calì, A. Poulovassilis, and P. T. Wood. Flexible query processing for sparql. *Semantic Web*, 8(4):533–563, 2017.

[38] T. Gaasterland. Cooperative answering through controlled query relaxation. *IEEE Expert*, 12(5):48–59, 1997.

[39] B. Ganter and R. Wille. *Formal concept analysis: Mathematical foundations*. Springer Science & Business Media, 2012.

[40] M. Gomez, A. Preece, M. P. Johnson, G. De Mel, W. Vasconcelos, C. Gibson, A. Bar-Noy, K. Borowiecki, T. La Porta, D. Pizzocaro, H. Rowaihy, and G. Pearson. An ontology-centric approach to sensor-mission assignment. In *Proceedings of the 16th International Conference on Knowledge Engineering & Knowledge Management (EKAW)*, pages 347–363, 2008.

[41] F. Grandi. T-SPARQL: A TSQL2-like temporal query language for RDF. In *Local Proceedings of the 14th East-European Conference on Advances in Data Bases & Information Systems (ADBIS)*, pages 21–30, 2010.

[42] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: Complex event processing over streams. Technical report, arXiv cs/0612128, 2006.

[43] M. Hagen, M. Potthast, B. Stein, and C. Bräutigam. Query segmentation revisited. In *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pages 97–106, 2011.

[44] A. Haller, K. Janowicz, S. J. Cox, M. Lefrançois, K. Taylor, D. Le Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, and C. Stadler. The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web*, 10(1):9–32, 2019.

[45] M. Hanlon and R. Anderson. Real-time gait event detection using wearable sensors. *Gait Posture*, 30(4):523–527, 2009.

[46] D. Harman. Relevance feedback revisited. In *Proceedings of the 15th ACM Annual International Conference on Research & Development in Information Retrieval (SIGIR)*, pages 1–10, 1992.

[47] N. Hasni and R. Bouallegue. Ontology for mobile phone operating systems. Technical report, arXiv:1207.2606, 2012.

[48] Y. He, J. Tang, H. Ouyang, C. Kang, D. Yin, and Y. Chang. Learning to rewrite queries. In *Proceedings of the 25th ACM International on Conference on Information & Knowledge Management (CIKM)*, pages 1443–1452, 2016.

[49] J. Heer, M. Agrawala, and W. Willett. Generalized selection via interactive query relaxation. In *Proceedings of the ACM International Conference on Human Factors in Computing Systems (SIGCHI)*, pages 959–968, 2008.

[50] J. M. Hellerstein, M. Stonebraker, and J. e. a. Hamilton. Architecture of a database system. *Foundations and Trends in Databases*, 1(2):141–259, 2007.

[51] J. R. Hobbs and F. Pan. Time ontology in OWL. https://www.w3.org/TR/owl-time/, 2017. Accessed: 2019-10-24.

[52] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[53] A. K. Jain. Data clustering: 50 years beyond *k*-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[54] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, 1984.

[55] J. M. Jasiewicz, J. H. Allum, J. W. Middleton, A. Barriskill, P. Condie, B. Purcell, and R. C. T. Li. Gait event detection using linear accelerometers or angular velocity transducers in able-bodied and spinal-cord injured individuals. *Gait Posture*, 24(4):502–509, 2006.

[56] K. Kemalis and T. Tzouramanis. SQL-IDS: A specification-based approach for SQL-injection detection. In *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC)*, pages 2153–2158, 2008.

[57] K. Koperski, J. Adhikary, and J. Han. Spatial data mining: Progress and challenges survey paper. In *Proceedings of the 1st ACM SIGMOD Workshop on Research Issues on Data Mining & Knowledge Discovery (DMKD)*, pages 1–10, 1996.

[58] T. Kurz, S. Schaffert, K. Schlegel, F. Stegmaier, and H. Kosch. SPARQL-MM-extending SPARQL to media fragments. In *Proceedings of the 11th European Semantic Web Conference (ESWC)*, pages 236–240, 2014.

[59] T. Labeodan, C. De Bakker, A. Rosemann, and W. Zeiler. On the application of wireless sensors and actuators network in existing buildings for occupancy detection and occupancy-driven lighting control. *Energy & Buildings*, 127:75–83, 2016.

[60] J. Lee, B. Bagheri, and H.-A. Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[61] C. Leonardi, A. Cappellotto, M. Caraviello, B. Lepri, and F. Antonelli. SecondNose: An air quality mobile crowdsensing system. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction (NordCHI)*, pages 1051–1054, 2014.

[62] S. Li, S. H. Son, and J. A. Stankovic. Event detection services using data service middleware in distributed sensor networks. In *Proceedings of the 2nd International Conference on Information Processing in Sensor Networks (IPSN)*, pages 502–517, 2003.

[63] X. Li, D. Li, J. Wan, A. V. Vasilakos, C.-F. Lai, and S. Wang. A review of industrial wireless networks in the context of industry 4.0. *Wireless Networks*, 23(1):23–41, 2017.

[64] G. Luo, C. Tang, H. Yang, and X. Wei. MedSearch: A specialized search engine for medical information retrieval. In *Proceedings of the 17th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 143–152, 2008.

[65] N. Maisonneuve, M. Stevens, and B. Ochab. Participatory noise pollution monitoring using mobile phones. *Information Polity*, 15(1-2):51–71, 2010.

[66] E. Mansour, R. Chbeir, and P. Arnould. eVM: An event virtual machine framework. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 39:130–168, 2018.

[67] E. Mansour, R. Chbeir, and P. Arnould. EQL-CE: an event query language for connected environments. In *Proceedings of the 23rd International Database Applications & Engineering Symposium (IDEAS)*, pages 7:1–7:10, 2019.

[68] E. Mansour, R. Chbeir, and P. Arnould. HSSN: an ontology for hybrid semantic sensor networks. In *Proceedings of the 23rd International Database Applications & Engineering Symposium IDEAS*, pages 8:1–8:10, 2019.

[69] E. Mansour, G. Tekli, P. Arnould, R. Chbeir, and Y. Cardinale. F-SED: feature-centric social event detection. In *Proceedings of the 28th International Conference on Database & Expert Systems Applications DEXA, Part.III*, pages 409–426, 2017.

[70] D. Mendez, A. J. Perez, M. A. Labrador, and J. J. Marron. P-sense: A participatory sensing system for air pollution monitoring and control. In *Proceedings of the IEEE International Conference on Pervasive Computing & Communications Workshops (PERCOM) Workshops*, pages 344–347, 2011.

[71] V. Mezaris, I. Kompatsiaris, and M. G. Strintzis. An ontology approach to object-based image retrieval. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, volume 2, pages II–511, 2003.

[72] M. V. Moreno, L. Dufour, A. F. Skarmeta, A. J. Jara, D. Genoud, B. Ladevie, and J.-J. Bezian. Big data: The key to energy efficiency in smart buildings. *Soft Computing*, 20(5):1749–1762, 2016.

[73] C. B. Necib and J. C. Freytag. Using ontologies for database query reformulation. In *Local Proceedings of the 8th East-European Conference in Data Bases and Information Systems (ADBIS)*, volume 4, pages 1–1, 2004.

[74] K. Peltonen and D. Meyerzon. Scoping queries in a search engine, 2005. US Patent 6,898,592.

[75] M. Perry, P. Jain, and A. P. Sheth. SPARQL-ST: Extending SPARQL to support spatiotemporal queries. In *Geospatial semantics and the semantic web*, pages 61–86. Springer, 2011.

[76] D. Petersen, J. Steele, and J. Wilkerson. WattBot: A residential electricity monitoring and feedback system. In *Extended Abstracts on Human Factors in Computing Systems (CHI EA)*, pages 2847–2852, 2009.

[77] H. Pirahesh, J. M. Hellerstein, and W. Hasan. Extensible/rule based query rewrite optimization in starburst. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, volume 21, pages 39–48, 1992.

[78] U. Priss. Formal concept analysis in information science. *Annual Review of Information Science & Technology*, 40(1):521–543, 2006.

[79] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu. Ear-phone: An end-to-end participatory urban noise mapping system. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 105–116, 2010.

[80] S. U. e. a. Rehman. DBSCAN: Past, present and future. In *Proceedings of the 5th International Conference on Applications of Digital Information & Web Technologies (ICADIWT)*, pages 232–238, 2014.

[81] T. e. a. Reuter. Reseed: Social event detection dataset. In *Proceedings of the 5th ACM Multimedia Systems Conference (MMSys)*, pages 35–40, 2014.

[82] D. Roman, U. Keller, H. Lausen, J. De Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[83] M. Rouane-Hacene, M. Huchard, A. Napoli, and P. Valtchev. Relational concept analysis: Mining concept lattices from multi-relational data. *Annals of Mathematics & Artificial Intelligence*, 67(1):81–108, 2013.

[84] D. J. Russomanno, C. Kothari, and O. Thomas. Sensor ontologies: From shallow to deep models. In *Proceedings of the 37th Southeastern Symposium on System Theory (SSST)*, pages 107–112, 2005.

[85] S. Sagar, M. Lefrançois, I. Rebaï, M. Khemaja, S. Garlatti, J. Feki, and L. Médini. Modeling smart sensors on top of SOSA/SSN and WoT TD with the semantic smart sensor network (S3N) modular ontology. In *Best Workshop Papers of International Semantic Web Conference (ISWC)*, 2018.

[86] D. Sánchez, M. Batet, D. Isern, and A. Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert systems with Applications*, 39(9):7718–7728, 2012.

[87] C. Schlenoff, T. Hong, C. Liu, R. Eastman, and S. Foufou. A literature review of sensor ontologies for manufacturing applications. In *Proceedings of the IEEE International Symposium on Robotic & Sensors Environments (ROSE)*, pages 96–101, 2013.

[88] N. P. e. a. Schultz-Møller. Distributed complex event processing with query rewriting. In *Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS)*, page 4, 2009.

[89] O. Seye, C. Faron-Zucker, O. Corby, and C. Follenfant. Bridging the gap between RIF and SPARQL: Implementation of a RIF dialect with a SPARQL rule engine. *Proceedings of the ECAI Workshop on Artificial Intelligence meets the Web of Data (AImWD)*, page 19, 2012.

[90] P. H. Shaikh, N. B. M. Nor, P. Nallagownden, I. Elamvazuthi, and T. Ibrahim. A review on optimized control systems for building energy and comfort management of smart sustainable buildings. *Renewable & Sustainable Energy Reviews*, 34:409–429, 2014.

[91] G. Singh, N. Parikh, and N. Sundaresan. Rewriting null e-commerce queries to recommend products. In *Proceedings of the 21st International Conference on World Wide Web (WWW)*, pages 73–82, 2012.

[92] K. Sohraby, D. Minoli, and T. Znati. *Wireless sensor networks: Technology, protocols, and applications*. John Wiley, 2007.

[93] S. Staab and R. Studer. *Handbook on ontologies*. Springer Science & Business Media, 2010.

[94] T. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades. Bonsai: A smart building ontology for ambient intelligence. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining & Semantics (WIMS)*, page 30, 2012.

[95] C. Strobl. Dimensionally extended nine-intersection model (de-9im). *Encyclopedia of GIS*, pages 470–476, 2017.

[96] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. In *Proceedings of the 17th International Conference on World Wide Web (WWW)*, pages 347–356, 2008.

[97] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar. *Introduction to data mining*. Pearson Education, 2nd edition, 2006.

[98] D. Tunkelang. Query rewriting: An overview. https://queryunderstanding.com/query-rewriting-an-overview-d7916eb94b83, 2017. Accessed: 2019-08-08.

[99] D. Van Der Merwe, S. Obiedkov, and D. Kourie. Addintent: A new incremental algorithm for constructing concept lattices. In *Proceedings of the 2nd International Conference on Formal Concept Analysis (ICFCA)*, pages 372–385, 2004.

[100] P.-Y. Vandenbussche and B. Vatant. Metadata recommendations for linked open data vocabularies, 2011.

[101] W. A. Wahyudi and M. Syazilawati. Intelligent voice-based door access control system using adaptive-network-based fuzzy inference systems (ANFIS) for building security. *Journal of Computer Science*, 3(5):274–280, 2007.

[102] S. Wang, J. Wan, D. Zhang, D. Li, and C. Zhang. Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101:158–168, 2016.

[103] X. Wang, X. Zhang, and M. Li. A survey on semantic sensor web: Sensor ontology, mapping and query. *International Journal of u-and e-Service, Science & Technology*, 8(10):325–342, 2015.

[104] J. Welch, F. Guilak, and S. D. Baker. A wireless ECG smart sensor for broad application in life threatening event detection. In *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine & Biology Society (IEMBS)*, volume 2, pages 3447–3449, 2004.

[105] R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. 1982.

[106] N. Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Communications of the ACM*, 20(11):822–823, 1977.

[107] N. Wirth. *Compiler construction*. Addison-Wesley, 1996.

[108] J. K. Wong, H. Li, and S. Wang. Intelligent building research: A review. *Automation in Construction*, 14(1):143–159, 2005.

[109] Y.-H. Wu, H. J. Miller, and M.-C. Hung. A GIS-based decision support system for analysis of route choice in congested urban road networks. *Journal of Geographical Systems*, 3(1):3–24, 2001.

[110] L. Yu, N. Wang, and X. Meng. Real-time forest fire detection with wireless sensor networks. In *Proceedings of the IEEE International Conference on Wireless Communications, Networking & Mobile Computing (WCNM)*, volume 2, pages 1214–1217, 2005.

[111] S. Zampolli, I. Elmi, F. Ahmed, M. Passini, G. Cardinali, S. Nicoletti, and L. Dori. An electronic nose based on solid state sensor arrays for low-cost indoor air quality monitoring applications. *Sensors & Actuators B: Chemical*, 101(1-2):39–46, 2004.