A Reference Architecture and a Software Platform for Engineering Internet of Things Search Engines

Nguyen Khoi Tran

School of Computer Science

The University of Adelaide

This dissertation is submitted for the degree of

Doctor of Philosophy

Supervisors: Prof. Muhammad Ali Babar, Prof. Michael Sheng, and Dr. Lina Yao

August 2018

To my mother and father, I dedicate this thesis.

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Nguyen Khoi Tran

August 2018

Acknowledgements

This thesis would not be possible without the guidance and support of many people. I would like to use this opportunity to acknowledge their great effort.

I am deeply indebted to Prof. Ali Babar, who took me under his wing and supported me tirelessly during the most challenging periods of my candidature. Prof. Ali Babar showed me a different angle to approach my research topic and provided the guidance necessary for me to realize it. He always offered a voice of reason to keep me from wandering off the track and was so patient with my blunders. When I started writing research papers in an unfamiliar field, he assisted me to work through numerous revisions even after the work-hour and on the weekends. Some of these papers, fortunately, got published and formed a large part of this thesis.

I am also thankful for the support of Prof. Michael Sheng, who allowed me to start the Ph.D. journey and introduced me to the exciting research topic that followed me until today. I will always remember the lessons of Prof. Michael Sheng on three pillars of a research career, on striving for excellence, and on pursuing the solution until the end. I would also like to extend my thanks to Dr. Lina Yao for the encouragement and insightful discussions in the early days of my candidature. The success of Dr. Lina Yao is an inspiration for me to work harder and better.

I would also like to thank my colleagues in the research group of Prof. Michael Sheng – Yongrui, Wenjie, Ali – and in the CREST group of Prof. Ali Babar – Mojtaba, Faheem, Chadni, Bakheet – for useful discussions and advice.

I would also like to thank my friends and colleagues in the International Peer Mentor team of the University of Adelaide. The time I spent volunteering and hanging out with them provided the needed fresh-air to keep me going.

Finally, I express my most profound gratitude to my mother and father for their unwavering support and encouragement in this long and challenging journey. Thank you for always being proud of me and believing that I can do it.

This thesis is supported by the Beacon of Enlightenment Ph.D. Scholarship of The University of Adelaide.

Abstract

The Internet of Things (IoT) is here. Enabled by advances in the wireless networking and the miniaturization of embedded computers, billions of physical things have been connecting to the Internet and offering their ability to sense and react to the real-world phenomena. These abilities form the content of IoT, which enable applications such as smart-city, smart-building, assisted living, and supply chain automation. The Internet of Things Search Engines (IoTSE) support human users and software systems to detect and retrieve IoT content for realizing the stated applications. Due to the diversity and sensitivity of IoT content, the literature has suggested that IoTSE will emerge as a large number of small instances, each of which monitors a specific IoT infrastructure and specializes in querying a particular type of IoT content. Various internal activities (i.e., components), as well as the logical and physical arrangement of those activities (i.e., architectural patterns), will overlap between IoTSE instances. The emergence of a large number of IoTSE instances, which possess overlapping operations and architecture, highlights the need for leveraging prior components and architectural patterns in engineering IoTSE instances. However, as an IoTSE reference architecture and a software infrastructure to guide and support such reuse-centric IoTSE engineering have not existed, a majority of IoTSE instances have been engineered from scratch.

This thesis aims at proposing the reference architecture and the software infrastructure to support leveraging prior components and architectural patterns in engineering IoTSE instances. The key contributions of this thesis include a reference architecture that describes the constituting components and architectural patterns of an IoTSE instance, and software infrastructure that supports utilizing the reference architecture in developing reusable, composable IoTSE components and engineering IoTSE instances from those components.

In order to propose the IoTSE reference architecture, we conducted a systematic and extensive survey of over one decade of IoTSE research and development effort from both an academic and an industrial perspective. We identified commonalities among diverse classes of IoTSE instances and compiled this knowledge into a reference architecture, which defines 18 components, 13 composition patterns, and 6 deployment patterns. We assessed the reference architecture by mapping it onto two IoTSE prototypes that represent the most common types of IoTSE in the literature and possess the more complicated architecture compared to other types.

In order to develop the software infrastructure, we first proposed a kernel-based approach to IoTSE engineering, which was inspired by the design of modern operating systems. In this approach, IoTSE instances operate as a collection of independently developed IoTSE components that are plugged into a shared kernel. This kernel provides essential utilities to run IoTSE components and control their interactions to fulfill the functionality of an IoTSE instance. The kernel also provides templates that simplify the development of IoTSE components that are interoperable and compliant with the proposed reference architecture. In a case study, which involves engineering an IoTSE prototype, the kernel managed to reduce the amount of new source line of code to just 30%.

The kernel-based approach supports engineering a majority of prominent IoTSE types detected in the literature. To enhance its support for emerging classes of IoTSE and prepare for future features in the reuse-centric IoTSE engineering, we proposed a platform-based approach to IoTSE engineering that extends the kernel-based approach. The platform-based approach revolves around an Internet of Things Search Engine Platform – ISEP – that supports developing interoperable IoTSE components, accumulating those components, and allowing search engine operators to engineer IoTSE instance from them using any valid architectural pattern defined in the reference architecture, without modifying the implementation of the components. In a case study, the platform-based approach enabled engineering complex IoTSE instances entirely from the components of simpler ones. Both the ability to engineer various IoTSE instances from a set of components and the engineering of new IoTSE instances entirely from accumulated components are unprecedented in the IoTSE literature.

Future research can focus on devising mechanisms that leverage the architecture and the infrastructure proposed in this thesis to accumulate the knowledge generated in the process of engineering IoTSE instances and use it to introduce automation gradually to IoTSE engineering. Eventually, when the automation is proven to be trustworthy and reliable, machines might compose and deploy IoTSE instances in real-time to adapt to the incoming queries and the state of the computing infrastructure. By achieving this degree of automation, we will have realized a search engine for the Internet of Things.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

With the advent of *the Internet of Things (IoT)* [1, 2], the Internet has been rapidly becoming a "library of everything" where humans and software systems can find not only digital documents but also information, real-time states, and interactions of the physical world. Emerging from the advances in embedded computing, wireless networking, telemetry, and global supply chain management, the IoT brings computerization and connectivity to our physical world. In the IoT systems, physical objects ("things") can connect to each other and software systems via the Internet. Things can share information about the surrounding world and themselves ("sensing information") to software systems and humans [3]. Things can receive instructions from software systems. Things can make decisions. By bridging the gap between the physical and digital worlds at the scale and simplicity of the Internet, the IoT promises to make processes more efficient, resource consumption more optimal, and, hopefully, life more secure and comfortable [4–8].

The content of the IoT comprises the ability to sense and interact with the real-world that are offered by IoT-enabled things and the data that things generate. Among IoT content, we can find everything from the temperature and humidity of a venue, the pollution level of the river in our city, the origin of a drug bottle, to a service to the control lighting and heating of our offices. The promises of the IoT – optimal processes, minimal resource consumption, comfort, safety, and security [9, 6, 7] – rely on the ability to locate and retrieve necessary content in this library of everything. The librarians of this IoT library are the *“Internet of Things Search Engines (IoTSE)”*.

The diversity of the solution space is a defining characteristic of IoTSE. While previous classes of search engines such as Web Search Engines and Document Retrieval Systems work primarily with text-based collections, which are relatively static and location-independent, IoTSE instances work with various types of IoT content, such as sensing data, actuating functionalities, digital representatives of things, and data records of things as they move across supply chains. Each type of content possesses its own syntactic and semantic heterogeneity and therefore requires specific mechanisms to discover and query.

The existing literature on the IoTSE reflects this diversity. Under the moniker of search and discovery in IoT, we can find prototypes that query Internet-connected sensors based on their static description (e.g., types of phenomenon that they observe, units of measurement that they use) [10–12] or their dynamic sensing values [13–19]; we can find prototypes that crawl and resolve queries for Internet-connected physical objects based on their real-time state [20, 21]; we can also find prototypes that query actuating services of physical objects [22–24]. Moreover, processing complex IoT queries might involve multiple types of content, and the IoTSE instances addressing these complex queries can emerge as a new IoTSE class with its own challenges and solutions. For instance, to find “a meeting room which is reporting abnormal energy consumption” in a smart-building, an IoTSE instance must process metadata of things (to answer “is this thing a meeting room?”) and sensing data streams (to answer “is there abnormality in this stream”), and then it must aggregate two sets of results to come up with the final results (Fig. 5.1).

Despite the diversity of the solution space, we have observed overlaps between IoTSE instances regarding internal activities (i.e., components), as well as the logical and physical arrangement of those activities (i.e., architectural patterns). For instance, an IoTSE instance

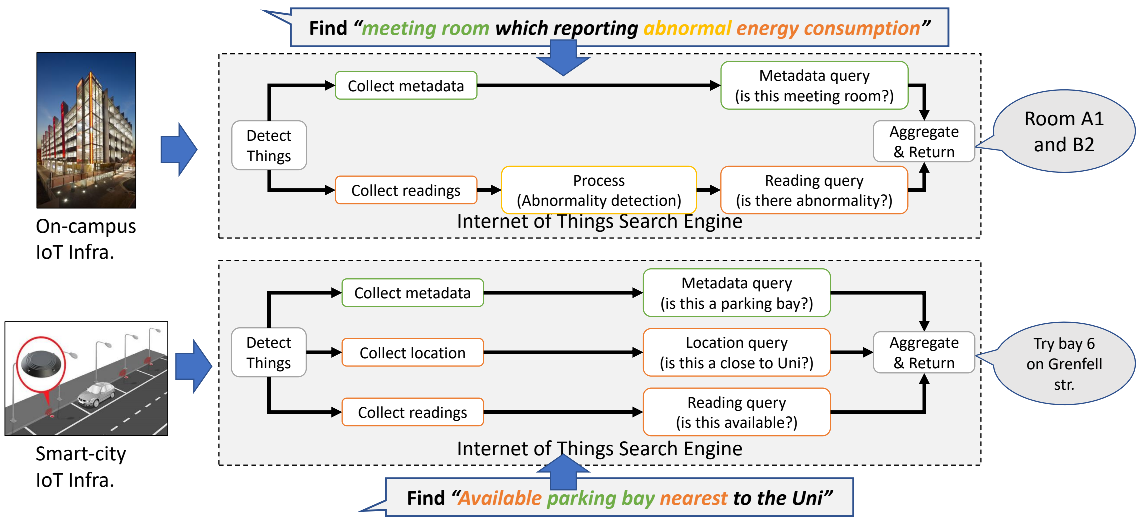

Find "meeting room which reporting abnormal energy consumption"
On-campus IoT Infra.
Detect Things
Collect metadata
Metadata query (is this meeting room?)
Aggregate & Return
Collect readings
Process (Abnormality detection)
Reading query (is there abnormality?)
Internet of Things Search Engine
Room A1 and B2
Smart-city IoT Infra.
Detect Things
Collect metadata
Metadata query (is this a parking bay?)
Collect location
Location query (is this a close to Uni?)
Aggregate & Return
Collect readings
Reading query (is this available?)
Internet of Things Search Engine
Try bay 6 on Grenfell str.
Find "Available parking bay nearest to the Uni"

Fig. 1.1 Involvement of multiple IoT content types in resolving a complex query, and overlaps in terms of internal activities between two IoTSE instances.

that works with a smart-parking infrastructure can share the mechanisms for detecting IoT-enabled things, collecting metadata and sensor readings, and processing queries on such content, with the aforementioned IoTSE instance for a smart-building (Fig. 5.1). In order to increase its sensitivity to changes in real-world phenomena, the smart-parking IoTSE might also use a similar deployment pattern consisting of a hierarchy of edge-nodes and cloud-nodes. The overlaps mentioned above suggest the potential, and the need, to leverage prior components and architectural patterns in engineering new IoTSE instance.

The emerging trend of decentralizing IoTSE, which is driven by the diversity and sensitivity of IoT content, also motivates leveraging prior components and patterns in IoTSE engineering. The content collected by IoTSE instances is highly sensitive as it comes from IoT infrastructure in smart-homes, smart-buildings, smart-cities, and supply-chains. Therefore, the value offered by IoTSE might not be able to offset the risk associated with sending IoT content out of the owner's network and surrendering this data to an external party. Due to the diversity and sensitivity of IoT content, the latest literature in IoTSE has suggested that this system will emerge as a large number of small instances, each of which monitors a specific IoT infrastructure and specializes in querying a particular type of IoT content, instead of a large centralized entity.

As the IoTSE might emerge as an extensive collection of small instances, and significant overlaps regarding components and architectural patterns exist between these instances, it would be possible and beneficial to leverage the implementation and architecture of prior IoTSE instances in engineering new ones. The reuse-centric IoTSE engineering would enable more effective and efficient IoTSE instances by allowing them to leverage state-of-the-art implementations for their internal operations. It would encourage specialization by allowing researchers and developers to focus on components of IoTSE that align with their interest and expertise, knowing that their research and engineering efforts will be compatible with most IoTSE instances. Finally, the reuse-centric IoTSE engineering would also reduce the coupling between architecture and implementation of an IoTSE instance as it allows IoTSE components to be used in instances with different architectural patterns. As a result, researchers can assess the impact of architectural design decisions on the quality of an IoTSE instance experimentally.

Despite the stated potential and benefits of the reuse-centric IoTSE engineering, a majority of the IoTSE instances in the literature has been designed and developed from scratch. This problem can be contributed to the lack of a reference architecture and software infrastructure to guide and support the engineering of IoTSE instances.

1.1 Research Objectives

This thesis aims to propose an IoTSE reference architecture and software infrastructure to support the engineering of IoTSE that allow leveraging prior components and architectural patterns. The reference architecture provides a blueprint for IoTSE engineering, while the infrastructure supports utilizing the reference architecture in developing IoTSE components and engineering IoTSE instances (Fig. 1.2). The objective of this thesis is threefold:

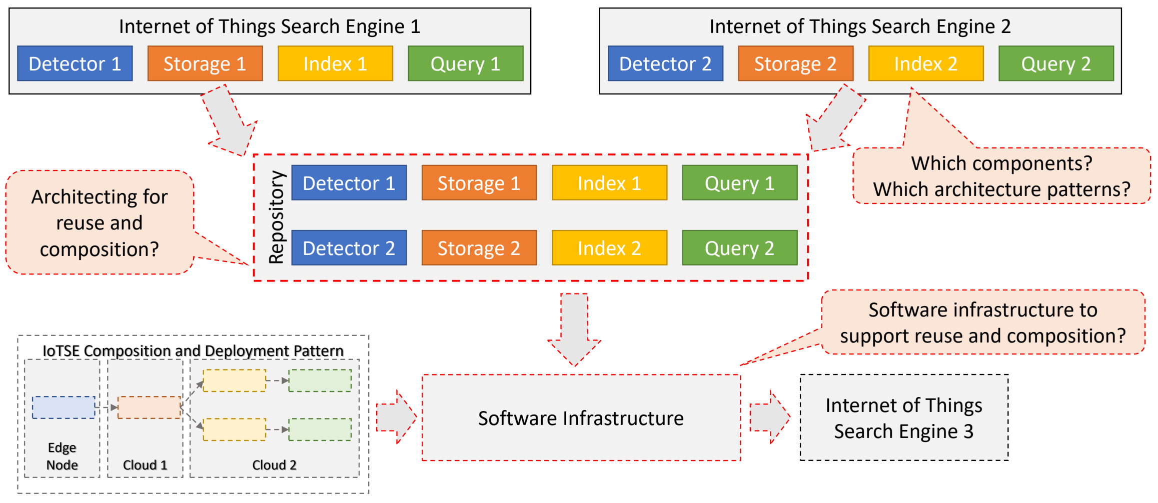

Internet of Things Search Engine 1
Detector 1
Storage 1
Index 1
Query 1
Internet of Things Search Engine 2
Detector 2
Storage 2
Index 2
Query 2
Repository
Detector 1
Storage 1
Index 1
Query 1
Detector 2
Storage 2
Index 2
Query 2
Architecting for reuse and composition?
Which components? Which architecture patterns?
Software infrastructure to support reuse and composition?
IoTSE Composition and Deployment Pattern
Edge Node
Cloud 1
Cloud 2
Software Infrastructure
Internet of Things Search Engine 3

Fig. 1.2 A vision of the reuse-centric IoTSE Engineering based on a reference architecture and supporting software infrastructure.

- **To provide a holistic insight into the current state of IoTSE research and development.** While over one decade has passed since the pioneering works on IoTSE a holistic view of IoTSE has not been established, partially due to the diversity of its solution space.

- **To propose a reference architecture that captures the commonalities of IoTSE**. This architecture specifies the shared components and architectural patterns to provide a blueprint for engineering IoTSE instances.

- **To propose a software infrastructure to support the reuse-centric engineering of IoTSE.** It simplifies the development of reusable, composable IoTSE components, and enables the engineering of IoTSE instances from those components.

1.2 Contributions

The contributions of this thesis to the body of knowledge of the research and engineering of the Internet of Things Search Engines are summarized as follows.

- We provide an extensive review of over one decade of IoTSE research and development, taking into the consideration both academic and industrial efforts. We assessed the growth and the focus of IoTSE research by analyzing the number of published works and in-field citation count from over 200 relevant IoTSE works, spanning the period between 2001 and 2016. We proposed a model for describing IoTSE instances and used it to assess 36 representative IoTSE prototypes. We also conducted an assessment of 6 industrial efforts and 4 standards related to IoTSE.

- We propose a reference architecture for IoTSE, which specifies 18 functional components, 13 composition patterns, and 6 deployment patterns. This reference architecture forms the basis for enabling the reuse-centric engineering of IoTSE instances.

- We propose a kernel-based approach to engineering IoTSE. This approach revolves around a kernel that provides a skeleton to develop components of an IoTSE instance, which help to reduce the development effort. The kernel also ensures the interoperability between IoTSE components as it defines the component interfaces and the format of exchanged messages. In a reference implementation of an IoTSE instance for smart-building management, the kernel managed to reduce the amount of newly developed source lines of code to just 30%.

- We propose a platform-based approach to engineering IoTSE. This approach extends the kernel-based approach by separating the composition and deployment decisions from the implementation of IoTSE components. It allows accumulating IoTSE components, and then composing and deploying them based on different architectural patterns to generate IoTSE instances. The conducted case studies showed that the platform-based approach enabled engineering multiple IoTSE instances with different architecture and quality from the same set of IoTSE components, without modifying their implementation. Notably, this approach allowed engineering complex IoTSE instances entirely from the components of simpler ones. This capability is unprecedented in the IoTSE literature.

1.3 Publications Related to this Thesis

Ten published or submitted papers were produced during my Ph.D. candidature. Seven of them are direct results of the research presented in this thesis and form the central part of its chapters. These publications are listed below, and the chapter on which each publication is based is presented in brackets.

1. **Tran, Nguyen Khoi.** “Searching the Web of Things: Resolving a Real Library of Babel.” In International Conference on Service-Oriented Computing, pp. 127-132. Springer, Cham, 2016. *(Chapter 2)*

2. **Tran, Nguyen Khoi**, Quan Z. Sheng, Muhammad Ali Babar, and Lina Yao. “Searching the Web OF Things: state of the art, challenges, and solutions.” ACM Computing Surveys (CSUR) 50, no. 4 (2017): 55. *(Chapter 2)*

3. **Tran, Nguyen Khoi**, Quan Z. Sheng, M. Ali Babar, and Lina Yao. “A Kernel-Based Approach to Developing Adaptable and Reusable Sensor Retrieval Systems for the Web of Things.” In International Conference on Web Information Systems Engineering, pp. 315-329. Springer, Cham, 2017. *(Chapter 4)*

4. *(Accepted)* **Tran, Nguyen Khoi**, Quan Z. Sheng, M. Ali Babar, Lina Yao, Wei Emma Zhang, and Schahram Dustdar. "Internet of Things Search Engine: Concepts, Classification, and Open Issues." Communications of ACM *(Chapter 2)*

5. (Under review) **Tran, Nguyen Khoi**, M. Ali Babar, Quan Z. Sheng, and John Grundy. "A Software Platform for Enabling Architecture Level Reuse in Engineering IoT Search Engines" In 41st ACM/IEEE International Conference on Software Engineering (ICSE), 2019. *(Chapter 5)*

6. (To be submitted) **Tran, Nguyen Khoi**, M. Ali Babar, and Quan Z. Sheng. "A Reference Architecture for Internet of Things Search Engines" ACM Transactions on Software Engineering and Methodology (TOSEM). *(Chapter 3)*

7. (To be published) **Tran, Nguyen Khoi**, M. Ali Babar, and Quan Z. Sheng. "A Service-oriented Architecture for Internet of Things Search Engines." Technical Report. *(Chapter 5*

1.4 Thesis Structure

This thesis comprises six chapters. Chapter 2 presents a comprehensive survey of over one decade of IoTSE research and development from both an academic and industrial perspective. It starts by presenting a background of IoTSE and then proposing a model for describing IoTSE instances. Afterward, it introduces the framework for conducting the survey and then presents the survey results. This chapter concludes with a discussion on open issues of IoTSE research and engineering, including leveraging reuse in IoTSE engineering to address the diversity of the solution space.

Chapter 3 presents an IoTSE reference architecture. It captures the commonalities of IoTSE that have emerged from the survey. This chapter starts by presenting the method that was utilized to derive the reference architecture. Then, it introduces 18 types of IoTSE components, which are arranged into 5 horizontal layers and 3 vertical layers. Then, it specifies 13 composition patterns and 6 deployment patterns that have emerged from the IoTSE literature. Afterward, it proposes a framework for utilizing the reference architecture to instantiate IoTSE instances. Finally, this chapter presents the assessment of the reference architecture by mapping it into two representative IoTSE prototypes.

Chapter 4 and 5 present our work on the software infrastructure for enabling the reuse-centric IoTSE engineering. The kernel-based approach is proposed in Chapter 4. The platform-based approach, which extends the kernel-based approach to separate the architectural design decisions from the implementation of components, is proposed in Chapter 5. This chapter concludes with three case studies which generate 16 IoTSE instances in total from 8 IoTSE components. These case studies have demonstrated the feasibility of the reuse-centric IoTSE engineering, based on the architecture and the software infrastructure proposed in this thesis.

Chapter 6 concludes this thesis with final remarks and a discussion on future research directions.

Chapter 2

Searching the Internet of Things: State-of-the-art, Challenges, and Solutions

This chapter presents a systematic and extensive survey of over one decade of IoTSE research and development, taking into the consideration both academic and industrial efforts. It covers over 200 research works on IoTSE, 6 representative industrial efforts, and 4 related international standards. By analyzing the number of publication, the number of in-field citations, and the number of works receiving in-field citations between 2001 and 2016, we provide an insight into the growth of IoTSE research and a cluster of works published between 2010 and 2012 that have a significant influence on the field. By proposing a novel model called meta-path for describing the functionality of an IoTSE instance, and a basic modular decomposition of IoTSE, we analyze and provide insights into the internal operations of representative IoTSE prototypes. Based on these results, we identify the open issues in IoTSE research, including addressing the diversity of IoTSE solution space.

2.1 Introduction

Our world is becoming a resource library for software applications. Advances in embedded computing and low-power wireless communication bring Internet connectivity to physical objects, forming the Internet of Things (IoT) [6]. An emerging trend is reusing technologies and techniques of the World Wide Web (e.g., HTTP protocol, HTML documents, and REST architectural style) to represent and serve IoT content. The so-called Web of Things (WoT) [25, 26] has emerged from such integration of the IoT and the Web and has been becoming increasingly prevalent. As the WoT is a subset of IoT, the survey presented in this chapter covers and is relevant to the content discovery and search mechanism for both the IoT and the WoT. Besides some intricacies, this chapter refers to the IoT and the WoT interchangeably, unless indicated otherwise.

Internet of Things Search Engines (IoTSE) bridge users and applications with the resource from IoT. For instance, consider a cyber-physical application in a smart home for seniors that blinks the lamp closest to the house owner to notify that the meal in the oven is done (Fig. 2.1). This application requires control service of lamps and light bulbs in the house, a sensor stream from the meat thermometer, a stream of results from the installed indoor localization system, and a Web service showing the optimal temperature for the meal being cooked.

Assuming that these resources are available, the task of developers is finding and linking them to the application logic. IoTSE decouples resource retrieval from the application. By querying an appropriate IoTSE, the application can retrieve resources needed for its operation without the manual configuration of developers. As long as the application has access to the IoTSE managing its deployment area, it can configure itself to work. Also, as long as the IoTSE continues to manage changes of objects in its scope, the application always has access to the latest resources.

Research related to IoTSE begins from early 2000s and enjoys steady expansion ever since. It branches into different directions including object search, sensor search and functionality

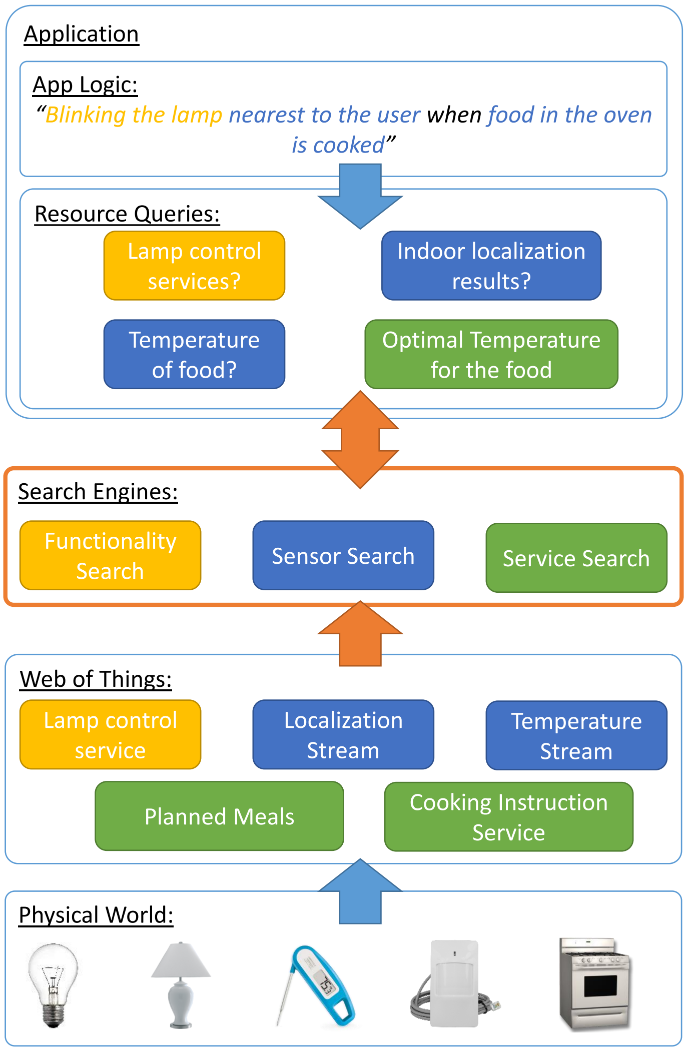


Fig. 2.1 Search engines as middle-ware to decouple application logic from resource retrieval in the Internet of Things

search. Essentially, IoTSE comprises of different types of systems, including unseen ones that will emerge when the adoption of WoT increases. This diversity complicates both the development of new IoTSE and the assessment of its state of the art. Therefore, a survey on IoTSE must focus on the whole field, not just only what happens within one type of IoTSE. Existing surveys either focus on one form of IoTSE [27–29] or focus on potential technical problems without considering the state of the whole field [30].

In this chapter, we perform a systematic survey on over 200 works related to IoTSE, with the focus on their diversity. Our contributions are fourfold:

- Proposing a conceptual model that describes IoTSE succinctly with resources involving in their query resolution process.
- Proposing a modular architecture as a reference to evaluate representative IoTSE prototypes.
- Reviewing the growth and state of the art of research and industrial efforts around IoTSE.
- Identifying open issues and potential solutions for the diversity challenge.

The remaining of this chapter is organized as follow. Section 2.2 introduces WoT and IoTSE concepts. Section 2.3 presents our proposed conceptual model and reference architecture for IoTSE. In Section 2.4, we present the analytical framework of our survey, which is built on our proposed models. We apply this framework on academic and industrial efforts and present the results in Section 2.5 and 2.6, respectively. Finally, we discuss prominent open research issues around IoTSE in Section 2.7.

2.2 Background

2.2.1 The Web of Things

The Web of Things (WoT) emerges from applying Web technologies to the Internet of Things to access information and services of physical objects. As the IoT has been increasingly accessible from the Web, techniques for discovering and searching for IoT content on the Web have been becoming more prevalent. Therefore, we devote this section to present an overview of the WoT.

In the WoT, each physical object possesses a digital counterpart which is commonly referred to as “Virtual Object” [31] or “Web Thing” [32]. These objects are built according to Representational state transfer (REST) architecture and accessed with HTTP protocol via RESTful API. A Web Thing can have an HTML or JSON representation, REST API to access its properties and actions, and an OWL-based semantic description.

Web Things are integrated into the Web in three ways. They can be hosted directly by Web Servers embedded into physical objects. With clever optimization, a Web Server can operate on an embedded computer with only 200 bytes of RAM and 7 KB of code [33]. For objects that cannot be modified, their virtual objects can be hosted by the Web Server embedded in a gateway device, or a cloud service. In these arrangements, the gateway device translates traffics in HTTP into the proprietary communication of the physical object. These three modes of integration are presented in Fig. 2.2. An overview of enabling technologies for bridging physical objects to the Internet is provided in [34]

In WoT, applications interact with physical objects with the familiar HTTP prococol and RESTful API. This simplifies the access to physical objects, allowing them to be used in Web applications and merged with existing Web resources [33]. It enhances the creation of value-added cyber-physical services by exposing sensing and actuating capabilities to a global open market [25]. Essentially, WoT turns the real-world into a library of software resources that is accessible via the Web.

2.2.2 Discovery and Search in the IoT

Internet of Things Search Engines (IoTSE) are “librarians” of IoT. They *discover* and gather IoT resources in a specific scope and allow users to *“search”* on these resources. For brevity and consistency, we use the term IoTSE for both systems designed specifically for WoT, and IoT or telemetric solutions that can be adapted to IoT.

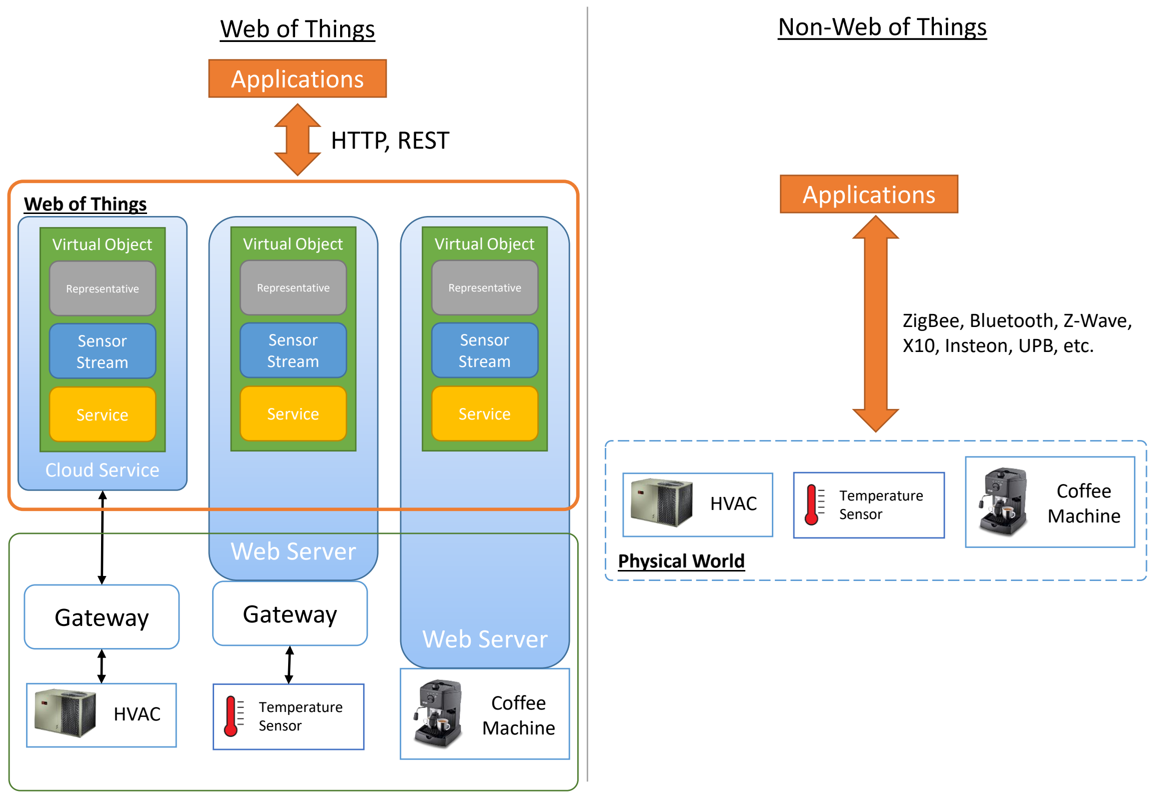


Fig. 2.2 Comparison between Web of Things and Non-Web of Things solution for accessing physical objects from an application

IoTSE appear in different usage scenarios with different forms and implementations in the literature:

- *Locating Physical Objects:* In early projects, IoTSE are commonly used to locate physical objects, which are tagged with passive RFID tags [35, 36] or sensor nodes [37–39].
- *Sensor Search:* GSN [40], CASSARAM [10] demonstrate the use of IoTSE for retrieving sensors based on their static meta-data and contexts, such as cost and reliability.

- *Finding Entity with Dynamic State:* Dyser [21] demonstrates an IoTSE that searches for physical objects (e.g., meeting room) based on their real-time states (e.g., “empty”) derived from their sensor readings.

- *Finding Actuation Services:* [22] demonstrates the use of IoTSE as a middle-ware for retrieving services offered by physical objects (e.g., changing lamp intensity).

- *Retrieving Data Records:* Prototypes of EPC Discovery Services [29] illustrate the use of IoTSE to retrieve data records relevant to an individual physical object. This problem was also investigated in Cooltown project [41].

Each form of IoTSE in the literature has its own characteristics and technical challenges. However, certain features are invariant between them. Therefore, we can build a common model to present majority of different IoTSE. We present this model in Section 2.3.1.

IoT Search vs Web Search

As IoT content is increasingly accessible via the Web, IoT Search Engines are sometimes considered a minor extension of Web Search Engines. However, this is not the case, due to the unique features of IoT (Table 2.1). Existing Web Search Engines face following four issues in IoT.

First, the IoT holds a vast amount of short, structured texts and non-text content (e.g., sensor streams, functionality) while Web Search Engines are optimized for long, unstructured texts. Therefore, text processing alone is not adequate for the IoT. Second, the IoT lacks the explicit links of the Web. Majority of relation between physical objects exists in the form of latent correlation [42]. Therefore, both crawling and link analysis mechanisms (e.g., Page Rank) are not directly applicable to the IoT. Third, the IoT has a varying but high dynamicity. For instance, sensors in the IoT update their content from once every several seconds to $1,000,000$ times per second [43]. Therefore, storage and indexing mechanisms of Web

Table 2.1 Differences between the Web and the Web of Things

	Web	**IoT**
Content Type	Long, unstructured texts (i.e., Web Pages)	Numerical data; Short structured texts
Link Structure	Extensive, explicit link structure between pages (i.e., URL)	Latent links
Dynamicity	Stable; Long lifetime; Slow changing	Volatile; High update rate (up to $1,000,000$ per second)
Scale	Over 1 billion Websites	Over 50 billion devices. Interactions happens in local areas

Search Engines that assume slow changing content cannot cope with the IoT. Finally, the IoT is both larger and smaller than the Web. It is expected to contain over 50 billion devices by 2020, while the Web currently holds only 1 billion Websites [1]. Yet, WoT applications interact with closely located resources for most of their life time. For instance, consider cyber-physical applications that interact with smart homes. Current Web Search Engines might not be able to scale up to serve over 50 billion devices. They are also not equipped to retrieve resources in the immediate vicinity of search users [26].

The stated issues imply that new techniques and mechanisms are required to realize IoTSE, despite the strong foundation laid by existing Web Search literature.

2.3 A Model for Internet of Things Search Engines

A model for IoTSE which provides succinct description of their operation and architecture is required to analyze their diverse literature. This model must fit naturally with majority of the existing projects and must be extensible to work with future, unseen types of IoTSE. We build our model base on over 200 existing works related to IoTSE in the literature. Section

[1]http://www.internetlivestats.com/total-number-of-websites/

2.3.1 presents "Meta-path" – our model for describing IoTSE. Section 2.3.2 presents our modular architecture for IoTSE.

2.3.1 Meta-path: The Signature of an IoTSE

The operation, usage purpose and implementation of an IoTSE are decided by the type of resources that it uses for assessing query (i.e., *"Query Resource"*), for deriving search results (i.e., *"Result Resource"*) and the chain objects linking them. For instance, a search engine working with sensor streams uses different indexing and query assessment schemes comparing to a search engine working with physical functionalities. Therefore, they have different technical challenges and usage purposes. Based on this observation, we decided to model IoTSE with the types of resource that they use and the path between these resources. We call this model *"Meta-path"*.

$$QueryResType[Feature] + ... \Rightarrow Obj \rightarrow ... \rightarrow Obj \Rightarrow ResultResType + ... \tag{2.1}$$

Equation 2.1 presents the pattern of a meta-path, which consists of three parts. The first part [$QueryResType[Feature] + ...$] describes the types of resources utilized by a search engine to assess queries. For clarity, a meta-path also presents features of resources that involve in the query assessment. The part [$\Rightarrow ResultResType + ...$] describes the types of resources used for building search results. The part [$\Rightarrow Obj \rightarrow ... \rightarrow Obj$] describes the chain of objects linking query resources and result resources. The first object in the chain provides query resources, while the last object in the chain provides result resources. Links between objects can be extracted via their correlation [42]. This path can be zero-length, which denotes that a same set of resource is utilized for both assessing queries and building results. For instance, Web Search and Document Search systems have zero-length paths.

The actual query resolution of an IoTSE involves multiple concrete paths between discovered resources that are instantiated from its meta-path. Consider the resolution process for a query for available meeting room in a smart building (Fig. 2.3) (e.g., Dyser system [21]). The result of this query is a set of digital representatives (e.g., Web page) of rooms in the building, which are of the type "meeting room" and have the state "available" reported by their sensors at the query time. The search engine has a set of sensor streams and digital representatives of physical objects created by a prior discovery process. The relations between sensor streams, digital representatives and objects are also recorded a priori. The first step of query resolution is matching the given query with content of sensor streams to find the ones reporting "available" state, and with metadata of representatives to find the ones belonging to meeting rooms. The second step is finding objects that have both matching sensor streams and digital representatives (i.e., "available meeting rooms"). Finally, representatives of these objects are selected as result resources to build search results. In this case, the search engine simply returns the list of Web pages. However, more complex processing such as aggregation or projection onto a map can be performed. The meta-path of our example search engine is $[D(Content) + R(Metadata) \Rightarrow Object \Rightarrow R]$. It is a common meta-path in the existing literature.

It should be noted that searching is more challenging in real world scenarios. For instance, sensor streams might not report the state "available" explicitly, and the metadata of a room might not show its type as a meeting room. These problems must be countered by specific mechanisms of the search engine. However, the whole process is invariant.

An IoT resource is a mapping from a reference to a specific content in IoT (e.g., sensor stream, actuation service) at a specific instance of time. Formally, a resource is a four-dimensional vector $Res = (ID, Metadata, Representation, Content)$. ID denotes the reference assigned to a WoT content, which is commonly a URI. Alternatives are Electronic Product Code (EPC), Ubiquitous ID (uID) [44] and IPv6. $Representation$ denotes forms that a resource can represent itself, such as an HTML or JSON document. *Content* denotes the content encapsulated by the resource. *Metadata* describes this content with key-value pairs [10] or textual tags [35]. Semantic description [31] is an emerging form of metadata.

We organize resources into eight types according to their origin and content. A resource that is related to an individual object in the real world has physical origin. Otherwise, it has digital origin. The encapsulated content has four types:

- ***R**epresentative* denotes the virtual representation of physical entities in the digital world. Web pages are the most relevant for WoT. However, other forms of documents (e.g., XML, JSON) and database records are also acceptable.
- ***S**tatic Information* denotes the rarely-changed data held by an object. It can be archived sensor readings [45], files loaded by human users [46], or records of events related to an object.
- ***D**ynamic Information* denotes the frequently-changed data held by an object. Sensor readings are the most prominent form of dynamic information in WoT [43].
- ***F**unctionality* denotes the actuation services provided by an object.

By relying on the type of resources and the link between them, which dominates the operation and implementation of a search engine, our meta-path model provides a succinct description for IoTSE. Our model is also extensible by introducing new relations between objects. For instance, by giving IoTSE the ability to link room objects with human users, it can be extended to resolve queries for staffs who are using meeting rooms in a specific building. Because of these ability, Meta-path is used to model IoTSE in our survey.

2.3.2 An Modular Decomposition of IoTSE

The query resolution process that we introduced is common in existing IoTSE projects. Its implementation changes depending on the meta-path of each search engine, but its activities and their arrangement are invariant. Therefore, we model these activities as standalone modules for building an architecture for IoTSE. Figure 2.4 presents our modular architecture.

Modules in our architecture are organized into layers. Two lower layers handle discovery activities. Two upper layers handle search activities. Storage modules for resource collections and indexes link two set of layers. The whole system is protected by security, privacy and trust assessment measures which are grouped into a vertical layer.

Discovery Layer interfaces IoTSE with resources in WoT. The *Discoverer* module detects resources specified in the meta-path of the search engine, in a certain physical scope. It can also be extended to discover relations between objects and resources. The *Retriever* module collects the discovered resources and passes them to the upper layer.

Index layer stores and indexes resources with its *Collection Manager* and *Indexer* modules. This layer also possesses *Query Independent (Q.I) Ranker* to rank resources according to their natural order, independent from user queries. For instance, Page Rank is a form of Q.I Ranking. Depending on the timing between discovery and search activity, an IoTSE can push resources directly to the query resolution process, skipping the index and storage layer. For instance, the MAX search engine [35] discovers relevant objects in its vicinity during query resolution process by broadcasting the query. The set of responded objects forms its query resource collection, which is dropped after the query is resolved. We consider these IoTSE having *"virtual resource collection"*. Majority of existing IoTSE actually have *"real resource collection"*.

Search layer carries out the query resolution process. The *Query Processor* module transforms raw user queries into the form processable by the system. The *Query Dependent (Q.D) Ranker* scores discovered query resources with respect to the user query and utilizes the recorded links between resources to find their corresponding result resources. A Meta-path with multiple types of query resources can be implemented by multiple Q.D Rankers. The *Ranking Aggregator* module is responsible for combining different Q.D and Q.I ranking results into a final score for each resource. Finally, the *Result Processor* extracts and aggregates the information from matching result resources and produces search results.

User Interface (UI) layer interfaces IoTSE with users. It provides *Query Interface* and *Result Interface* to receive queries and return search results, respectively. Their forms and implementations vary depending on the meta-path of an IoTSE. It also depends on type of users targeted by the IoTSE. Naturally, a system designed for software applications needs a different interface than a system designed for human users.

The modular architecture provides a reference framework assessing the diverse implementation of existing IoTSE. It assesses the support that each module receives from the existing works and how it is commonly implemented. Together with meta-path, the modular architecture forms our IoTSE model.

2.4 Analytical Framework of the Survey

The framework of our survey consists of three parts (Fig. 2.5). The data for our survey is collected from bibliographic data set of DBLP [2] (retrieved on Sep 14, 2016) and Scopus [3]. Works included in our dataset are either directly related to search and discovery in WoT, or highly referenced by directly related works. The preliminary selection is done by a tool that we developed (Alg. 1). The final selection is done manually to remove highly cited works that are not related to search engines, such as general surveys on WoT and the IoT. The complete list of works is available at [4].

The second part of our analytical framework is building a model for IoTSE based on the collected works. Results of this part are Meta-path and the modular architecture that we discussed previously. From these models, we identify 24 dimensions, organized into seven groups, to analyze existing works (Table 2.2). Dimensions from meta-path (1.1 and 1.2)

[2]http://dblp.uni-trier.de/xml/
[3]https://www.scopus.com/
[4]http://cs.adelaide.edu.au/~nguyen/publications.html

ALGORITHM 1: Prototype Selection Algorithm

```
input  : $DBLP$: Bibliographic Dataset from DBLP
         $SKW = \{\text{Discover}, \text{Search}, \text{Query}\}$: Keywords related to search and discovery activity
         $DKW = \{\text{WebofThings}, \text{WoT}, \text{InternetofThings}, \text{IoT}\}$: Keywords to limit the domain of an
         article
output : $Candidates$: Set of candidate papers for manual selection

initialize $Candidates$ list;
initialize $References = title : count$ dictionary;
foreach $article$ in $DBLP$ do
    if $article.name$ contains $SKW$ and ($article.name$ contains $DKW$ or $article.venue$ contains
      $DKM$) then
        append $article$ to $Candidates$;
    end
end
foreach $candidate$ in $Candidates$ do
    extract the list of references $refs$ from $candidate$;
    foreach $ref$ in $refs$ do
        increase $References[ref]$ by 1;
    end
end
foreach $reference$ in $References$ do
    if count of $reference$ is larger than 2 then
        append $reference$ to $Candidates$;
    end
end
return $Candidates$
```

assess the general operation of an IoTSE prototype. The remaining dimensions describe the way an IoTSE prototype implements modules in our architecture. We include additional dimensions that reflect non-functional requirements such as scalability and adaptability. We also include Experiment Type and Experiment Scale to assess the evaluation carried out by the prototype.

Table 2.2 Comparison Dimensions

Dimension	**Description**
1.1 Meta-path	Meta-path utilized by the WoT Search Engine under consideration.
1.2 Scope	The spatial range in which the WoT Search Engine can detect resources and interact with search users.
2.1 Discovery Scheme	Overall class of the discovery scheme utilized by the WoT Search Engine.
2.2 Mobility Support	Mechanisms utilized by the WoT Search Engine to detect and record the change in spatial locations of discovered entities.
2.3 Collector Type	Mechanisms to detect and collect resources.
3.1 Collection Type	The class of resource collections utilized by the WoT Search Engine.
3.2 Index Type	Mechanisms utilized by the WoT Search Engine to speed up the lookup process on resource collections.
3.4 Storage Scalability	Measures taken by the WoT Search Engine to ensure the scalability of its resource collections.
4.2 Query Model	The internal model of user queries utilized by the WoT Search Engine.
4.3 Result Model	The internal model of search results.
4.4 Q.D Ranking	Mechanisms utilized by the WoT Search Engine to assess the relevance of resources against a given query.
4.5 Adaptability	The ability of the WoT Search Engine to adapt its operations to different usage scenarios (e.g., different types of users).
4.6 Search Scalability	Mechanisms of the WoT Search Engine to ensure the scalability of its query processing.

Continuation of Table 2.2

Dimension	Description
5.1 User Type	Type of search users for which the WoT Search Engine is designed.
5.2 Interface Modal	The channel (i.e., “mode”) on which the communication between search users and the search engine takes place.
5.3 Query Interface	The form of interface through which search users express their queries.
5.4 Result Interface	The form of interface through which search results are presented to users.
6.1 Security	The measures of the WoT Search Engines to protect itself against being breached by malicious parties.
6.2 Privacy	The measures of the WoT Search Engines to preserve the privacy of search users, sensor owners and sensed persons.
6.3 Trust	The measures taken by the WoT Search Engine to assess the trustworthiness of the discovered resources.
7.1 Experiment Type	The type of experiment carried out to evaluate the search engine prototype.
7.2 Experiment Scale	The scale of the carried out experiment, in term of the number of data points or participants.

End of Table

The third part of our framework is the analyzing the growth of research around IoTSE and its current state, reflected by classical and latest works in the field. We use the number of publication and *in-field* citations (i.e., references among over 200 IoTSE works) each year to assess the growth of the field. For the detailed analysis, we map a subset of works against the dimensions that we built in the second part of the framework. This subset of work is selected manually, with the attention on balancing the “classical” works with highest in-field citation count and latest works.

2.5 Research Prototypes

2.5.1 Overview of Major Research Prototypes

MAX: a Human-centric Search Engine for the Physical World:

MAX [35] is among the earliest works on building a search engine for the physical world. It is a standalone system that allows human users to provide a set of descriptive keywords to locate their tagged physical objects. To reflect its human-centric nature, MAX returns the location of the matching objects as landmarks instead of coordinates. MAX is organized into a three tier architecture which is closely mapped into the organization of the physical locations in the real world. Located at the highest level are base stations, powered by the power line. These computers host the search application, host the security agent and act as gateways between the network of wireless tags and the backbone network (i.e., Internet). The middle tier consists of battery-powered RFID readers that are tied to large, rarely moved physical objects that represent landmarks. The bottom tier consists of passive RFID tags attached to small, mobile objects such as books and mugs. Queries are propagated through the network from the base stations, and the identity of the RFID readers detecting matching objects are returned as landmarks. To protect privacy of object owners, MAX allows them to specify their objects and physical spaces as private or off-limit to prevent unauthorized discovery and searches.

Discovery mechanism of Global Sensor Network (GSN) system:

GSN [40] is actually a platform for integrating wireless sensor networks over the Internet, not a search engine. However, it is highly referenced due to its sensor selection mechanism when processing sensor streams. GSN models each sensor node as a virtual sensor. Each virtual sensor is identified by a unique name and has a set of key-value pairs to represent its metadata. In other word, these virtual sensors can be consider the digital representative resource in our model. When processing sensor streams according to the declarative deployment description of users, GSN uses this information to retrieve sensors and perform the processing.

SenseWeb - an Infrastructure for Shared Sensing:

SenseWeb [47] is a system for building applications and services based on the shared sensor data streams. Sensors in this system are connected to a centralized coordinator component through sensor gateway devices, which map their proprietary communication scheme into a standardized Web service API. An application requiring to use shared sensor data stream will interact with the tasking module of the coordinator a Web service API to express its sensing requests. The tasking module then searches on the static description of sensor streams to assess their capability, sharing willingness and other characteristics and return the relevant streams. This is a key distinction between the search service of SenseWeb and other search engines that also work with sensor streams such as DIS [45] and Dyser [21].

Distributed Image Search in Camera Sensor Network:

DIS [45] is a system that performs general purpose image search on camera sensor network to recognize different types of objects. Its main difference comparing to the similar system is that it is for general purpose usage and can be used to recognize different types of objects instead of application specific, which will lock the whole camera sensor network into one task. It is one of the unique features of this search engine. To cope with the massive stream of captured images, which are enormous in scale and fast in the generation rate, DIS employs a distributed search scheme, in which the discovery and search activities are carried out directly on each sensor nodes and results are combined into a single set of search result instead of having each sensor to transmit their readings to a centralized server for processing. Each image, either captured or supplied by search users are transformed into a set of 128 dimensional vectors using the Scale-Invariant Feature Transform (SIFT). These features are further clustered into Visual Words (i.e., “Visterms”) to further reduce the space to represent these images. The matching between queried image and captured images are performed on visterms with TF-IDF score similarly to the matching between documents. DIS supports both adhoc and continuous query. It can search on the newly captured image, or the set of images stored in the camera sensor node.

Microsearch:

Microsearch [46] is a scale-down information retrieval system that runs on sensor nodes with very limited computing and storage resources. It indexes small textual documents stored in the sensor node and returns the top-k documents that are most relevant to the query terms given by a search user. Documents are scored and ranked with the traditional Term Frequency (TF) and Inverse Document Frequency (IDF) metrics. While not being completely comparable to other works on this list, the unique approach of Microsearch provides an interesting alternative perspective on the problem. Therefore it is included in our analysis.

Object Calling Home (OCH) system:

OCH [37] system utilizes its participating mobile phones as a sensor network to locate missing physical objects. Each physical object is attached with a battery-powered Bluetooth transmitter, which is discovered by the phone’s built-in Bluetooth discovery mechanism. To deal with the potential huge scale of the network, OCH utilizes a scoping mechanism which utilizes the association between things, humans and physical locations to reduce the number of sensors to pull during query resolution. The ideas proposed by OCH have been applied in commercial products (e.g., TrackR tag [5], Tile [6])

[5]TrackR: https://www.thetrackr.com/
[6]Tile: https://www.thetileapp.com/

Dyser - a Real-time Search Engine for Real-world Entities:

The Dyser search engine [21] assesses queries of users against the real-world state of Web-enabled physical entities, which is reported by their attached sensors. The key challenge of Dyser is the dynamic nature of real-world states, which greatly surpasses the existing Web pages, rendering any indexes on these states outdated as soon as they are created. To solve this problem, Dyser assumes that sensor readings have a periodic nature and utilizes Sensor Rank algorithm [20] to predict sensor readings based on this assumption. The prediction result is used to order and minimize the sensor pull activity.

Ubiquitous Knowledge Base (uKB):

uKB [48] is a distributed knowledge base whose assertion knowledge (i.e., knowledge about individual objects) are distributed over RFID tags attached to physical entities. The architecture of uKB consists of RFID readers that are inter-connected as a Mobile Ad-hoc Network (MANET). In this survey, we focus on the discovery process in uKB which discovers and gathers relevant pieces of assertion knowledge to a client to perform reasoning activities. The first step of discovery process is syntactical matching, in which syntactically relevant tags are detected based on their identity and the identity of the ontology that they use to describe themselves. The second step is semantic matching in which relevant tags are downloaded for further semantic-based assessment. Storing semantic description inside physical objects is an interesting and relevant idea for WoT. Therefore, discovery process of uKB is included in our analysis.

Snoogle - a Search Engine for Pervasive Environment:

This work [38] proposes that the pervasion of information stored in the networked sensors attached to physical entities will soon turn the world into a physical database. Snoogle is a search engine designed to look up information in such physical database. This search engine receives a set of keywords from a search user and returns a set of k objects having textually relevant description. To resolve query on a large number of sensors with limited computing and communication resources, Snoogle utilizes a distributed top-k query algorithm with pruning based on the characteristic of flash memory and Bloom filter to further reduce the transmission size. To preserve privacy of object owners, the textual content stored in private objects are encrypted with Elliptic Curve Cryptography (ECC).

DiscoWoT - Extensible Discovery Service for Smart Things:

The heterogeneity of thing and service description is among the biggest challenges of WoT. DiscoWoT [49] is a semantic discovery service that aims to return the common representation form of any resource description given by a search user. DiscoWoT provides the common representation form and relies on strategies contributed by the community to translate resource description into this common form. The community effort lowers the entry-barrier for new WoT companies and products, and ensures that DiscoWoT is always up-to-date. While DiscoWoT appears to be very different from other WoT search engines, it is still mapped naturally into our model. If we consider each translation strategy as a function from a set of resource descriptions to the set of descriptions in the common representation form, then the union of these domains represents the set of all resource descriptions that DiscoWoT knows at a given point of time. In other word, this is the query resource collection of DiscoWoT. This collection is virtual, as it is not explicitly stored in the memory of the search engine.

IteMinder:

IteMinder [36] is a search engine that allows users to locate their physical entities. Each entity participating in the system is attached with a passive RFID tag that stores its unique identity. Landmarks also attached with RFID tags for identification. IteMinder utilizes a physical robot, equipped with a laser rangefinder for navigation and an RFID reader for detection, to crawl the physical environment and record the location of physical objects into a database. Users are provided a Web interface mapping to look up in this database for their objects.

Searching the Web of Thing:

This work [22, 50] develops a system for finding physical entities which have matching inputs and outputs to compose new applications and services. This system acts as a component in a larger WoT application framework instead of a standalone system. To describe physical entities, this system utilize five different ontologies to describe their finite state machines, their input and output structures, locations and owners. The entities are ranked by the similarity between their input structure and the output structure of the queried entity. This system also identifies the type of the interacting search user and adjusts its algorithm correspondingly.

Web of Things - Description, Discovery and Integration:

This work [51] proposes to describe Web-enabled smart things with a common ontology, and register all smart things participating the network in a central Knowledge Base server for discovery purpose. The ontology describes entities based on their four basic capabilities: identity, processing, communication and storage. A user wishing to search for a smart thing sends his request to an Ambient Space Manager system, which in turn utilizes a Knowledge Base agent to query the Knowledge Base server for semantically matching entities.

Searching in a Web-based Infrastructure for Smart Things:

This work [39] presents a distributed management infrastructure for environments populated with smart things, and a search engine prototype that allows users to perform look up for things in this infrastructure. These systems are organized as hierarchies according to logical identifiers of places that they cover to utilize the locality of smart things (i.e., things frequently interact with other things in their immediate environment). A user queries these systems by sending an HTTP GET request to one of their querying interfaces and providing the information for identifying a corresponding resource, along with spatial information to specify the query scope. Each query is modeled as a Web resource and assigned a unique URL. This URL is propagated through nodes in the infrastructure to build search results and returned to the search user.

Context-aware Service Discovery for WoT:

This work [52] explores the use of contextual information collected from heterogeneous sources, including information about the physical world provided by networked sensors, to search for user-centric and situation-aware services to human and devices. This work models contextual information and relations among contexts with an ontology model that is extended to model uncertainty and temporal context. The contextual information is used to search on a service repository that contains both traditional Web services and real-world services provided by physical entities.

Context-aware Sensor Search:

CASSARAM [10] is a system that searches for connected-sensors using their contextual information, such as availability, accuracy, reliability, response time, etc. It is motivated by the increasing number of sensors with overlapping capabilities deployed around the world and the lack of search functionality for these sensors. CASSARAM utilizes an extension of the Semantic Sensor Network Ontology (SSNO) [53] to describe the contextual information. A search user would query this ontology for sensors with a SPARQL query generated by the graphical user interface of CASSARAM. This interface also captures the references of the search user. The Euclidian distance between matched sensors and the user reference in a multidimensional space built from different types of sensor contextual information is used for ranking purpose. Top ranking sensors are returned as search results.

Content-based Sensor Search for Web of Things:

[15] defines content-based sensor search as the search for sensors that produce measurements within a certain range for a certain time period prior to the query. It is applied in WoT to find WoT-enabled physical entities that are in the queried real-time state. This work utilizes time-independent prediction models (TIPM) constructed for each individual sensor to rank them on based on their probability of having the queried state. This ranking activity reduces the communication overhead from validating the readings of matching sensors. TIPM is constructed from the assumption that a sensor reading which is frequently and continuously reported by a sensor in the past has a higher probability to be its current reading. To cope with the dynamic of sensor measurements, TIPM are continuously rebuilt and integrated into prior TIPM via a weighted sum. This method is evaluated by a combination of prototyping and simulation on a dataset of 162 sensors.

Ambient Ocean:

Ambient Ocean [54] is a search engine that enables context-aware discovery of Web resources. Ambient ocean operates on Web resources attached with a data structure called Ocean Metadata that holds context metadata entities describing the current discoverability context of the Web resource. Each context metadata is backed by a context handler that provides mechanisms for comparison and indexing. Ambient Ocean relies on the community for the construction of context handlers and for adding context metadata to Web resources. A user interacts with Ambient Ocean server through a client application running on his mobile device. This client application utilizes readings from the local sensors as query terms to describe the current context of the user to the Ambient Ocean server. A list of URL pointing to Web resources having the relevant context is returned as the search result. Ambient Ocean can learn the association between context metadata entities to expand the given query.

Semantic Discovery and Invocation of Functionalities for the Web of Things:

This work [23] describes a search and discovery mechanism for functionalities of physical entities. It aims to discover and expose high-level functionalities of a physical entity that can be realized by a combination of its low-level physical capabilities and functionalities exposed by other entities in the immediate area. These functionalities and capabilities are described in a shared ontology. Each physical entity queries this ontology with a set of SPARQL queries encapsulated in Java functions. This work is a part of the avatar architecture from ASAWoO project, which aims to build an infrastructure for enhancing appliance integration into the Web and enable the collaboration between heterogeneous physical entities.

IoT-SVKSearch:

IoT-SVK [17] is a hybrid search engine for WoT which is capable of resolving queries for WoT entities based on their textual description, their real-time sensor values, with respect to spatial and temporal constraints. IoT-SVK utilizes a uniform format to model the sampling data from WoT entities, which is distributed over multiple raw data storage for scalability. IoT-SVK utilizes three set of indexes. The full-text search on the description of WoT entities is handled by a B+ tree index. The spatial-temporal constraints of the queries are handled by an R-tree index, which is modified to support mobile entities. The value-based queries on sensor readings are handled a modified B+ tree index. Queries in IoT-SVK are processed as boolean expressions. The filtered search results are ranked according to an unspecified ranking mechanism. Evaluation of IoT-SVK is performed on a combination of real and simulated data from 352,000 sensors.

Gander:

Gander [55] is a middle-ware and a search engine for pervasive computing environment, which is deployed directly on each node in the environment. It is designed for discovering and retrieving “datum", produced by these nodes, by propagating queries between Gander nodes in an ad-hoc communication manner. Its prototype, however, is designed to work with virtual ad-hoc networks deployed over the Internet. Gander allows users to fuse raw data into semantic, higher-level states in run-time with predefined rules in form of graphs. A query in Gander is modeled as a composition of three partial functions for filtering reachable, matching and constraint-satisfying nodes. Gander is evaluated with extensive case study and simulation.

Meta-Heuristic Approach for Context-aware Sensor Search in the Web of Things:

This work [56] proposes a swarm intelligence method called AntClust to cluster sensors based on their meta-data and contexts for improving the scalability and efficiency of the search activity. The AntClust algorithm is inspired by the behavior of ants. It scatters all available sensors randomly on a sparse, two dimensional matrix and utilizes a set of agents to randomly pickup and drop sensors at different locations in the grid, biased by the relation between the selected sensor and its potential neighbors at the drop-off location. Performed experiments on a dataset of $100,000$ sensors show that AntClust achieves notable gain in efficiency at the expense of accuracy, comparing to CASARRAM [10].

DNS as a WoT Search Engine:

This work [24] assumes that every WoT entity exposes their content as RESTful Web Services and proposes to use DNS to perform location-based search for these services. Each service is assigned with a URL in form of “sensorid.service.location.env". A user searching for sensors of a specific type at a specific location first queries the domain name “service.location.env" to retrieve a list of sensor URL, and then utilizes DNS to translate the collected URLs into IP addresses. Each service is assumed to return a self-description written in Web Application Description Language (WADL). Two experiments were performed on simulated data to assess the response time and storage requirement of this approach.

ForwarDS-IoT:

ForwarDS-IoT [57] resolves queries for sensors and actuators whose semantic description stored on a federation of repositories. Each repository in ForwarDS-IoT has a domain-specific ontology, which is extended from SSN ontology. Users interact with ForwarDS-IoT via either its GUI or its RESTful API. Queries in ForwarDS-IoT specify conditions on metadata of physical objects, which are translated into a SPARQL queries and assessed against the stored semantic descriptions. ForwarDS-IoT supports both synchronous and asynchronous queries.

Extract - Cluster - Select (ECS):

ECS [58] is a framework for producing relevant and diversified search results for the queries on physical entities in the IoT. It utilizes the “Things Correlation Graph (TCG)”, which represents a network of correlations between things, namely shared geographical locations and entity type. ECS consists of three steps. First, correlation between things are extracted to build TCG using $l1$-based graph construction method. Second, clusters of things are formed from TCG with spectral clustering techniques. Finally, things are selected based on the user’s query and specified trade off between coherence and diversity of search results.

Query Processing for the IoT:

This work [59] considers the process of searching for digital resources matching with real-world information reported by connected sensors. It utilizes statistical models to optimize the energy consumption of sensors and billing costs of cloud servers hosting these search applications analytically. The evaluation is carried out on a visual sensor network composed of multiple BeagleBone Linux embedded platforms and an application running on Amazon Web Service Elastic Compute Cloud (AWS EC2).

Context-aware Search System for the IoT:

This work [60] presents a system which utilizes the contextual information extracted from the IoT sensor data, namely user's identity, location, query time and current activity to search for physical entities and their related information. The activity recognition is performed by the combination of an online classifier based on Hidden Markov Model and pre-calculated probability distributions of different activities with respect to different time slots and locations. The detected user's activity, along with other contextual information, is used as the query to retrieve relevant entities and information from a pre-built context ontology.

ViSIoT:

The Visual Search for Internet of Things system (VisIoT) [61] bridges sensor applications with public IoT cloud platforms by transforming the sensor information provided by public IoT clouds into the format required by the applications and exposing this information as virtual sensors via RESTful API. ViSIoT selects and ranks sensors on six "context properties" (i.e., battery, price, drift, frequency, energy consumption and response time) using TOPSIS technique. The presented case study on Open Weather Map dataset shows that ViSIoT is capable of translating and deploying $100,000$ sensors within two minutes.

ThinkSeek:

The ThinkSeek system [62] consists of a WoT crawler and a WoT search engine. The crawler extracts data about the physical world from public IoT cloud repositories on the Web, with the focus on live maps. Collected data is fed into a search engine and a Web-based visualization system. The ThinkSeek search engine supports both human users and smart devices. Humans utilize structured queries in form of $(Location, \{Keywords\})$ to interact with the system, while smart devices utilize a CoAP RESTful API.

LHPM:

LHPM [19, 18] is a prediction model for enabling searching on sensor content. LHPM consists of three parts. First, sensor readings are approximated with polynomials to lower the energy transfer cost. Second, sensor content is predicted with a multi-step, SVM-based method in which the new predictions are fed back as input. Finally, sensors are mapped into a two-dimensional vector space and ranked according to their cosine similarity with the given query. LHPM is evaluated with simulations with data from 54 temperature sensors from Intel Lab dataset and 78 water sensor from NOAA dataset. The experiment shows encouraging results on the approximation accuracy and energy consumption reduction.

2.5.2 Publication and Citation Analysis

Figure 2.6 presents the number of published works, in-field cited works and in-field citations each year from over 200 selected works. The number of works related to IoTSE increases steadily each year since early 2000s, with a surge in 2009. Coincidentally, 2009 also marks the birth of the Internet of Things as the number of devices connected to the Internet becomes larger than World's population [7]. This surge of interest represents a gradual shift in focus of the community connecting things to the Web to finding and utilizing Web-enabled things. It also reflects the sharp perception of the community working with IoTSE.

However, the number of in-field cited works does not keep up with the number of publications. The gap between them expands at a steep rate from 2010. In fact, majority

[7]http://www.postscapes.com/internet-of-things-history/

of in-field citations are held by a small set of works appearing around 2010. While citation count is not a perfect metric to evaluate the impact of research works, it can show that the existence of a research work is acknowledged by the the community. The lack of in-field citation is a possible indicator that majority of works around IoTSE are not detected by their peers.

2.5.3 IoTSE Form and Implementation Analysis

Comparison Result

The mapping of selected prototypes into dimensions defined in our analytical framework is presented in Figure 2.7, 2.8 and 2.9.

Operating scope of an IoTSE is presented in form of $[DiscoveryScope] - [SearchScope]$. **L**ocal scope denotes that the IoTSE can only find resources and provide search services to users in its vicinity, while **G**lobal scope denotes that it can operate across the Globe via the Web. An IoTSE can be tailored to work with **H**uman or **M**achine users. It's evaluation can be carried with **Prot**otypes on real devices or **Sim**ulation (e.g., network simulation with NS2).

The scheme of discovery process carried out by an IoTSE can be either **A**ctive or **P**assive. Active discovery means the search engine seek resources, while Passive discovery means resources are registered to the search engine. Depending on the scope and discovery scheme, an IoTSE uses different types of collector, including Web **Crawl**ers, resource **Reg**istration mechanisms and Local Discovery (**LD**), which includes mechanisms to detect entities and resources in the immediate vicinity. The support for mobile objects by an IoTSE is organized into four groups. **T**imer denotes the continuous resampling of object's location after a predefined time period. **B**eacon denotes the mechanism in which the search engine continuously broadcasts beacons for receiving objects to register themselves. Ad-hoc Pull (**AHP**) denotes that the location of objects are pulled every time a query is processed. Mobile

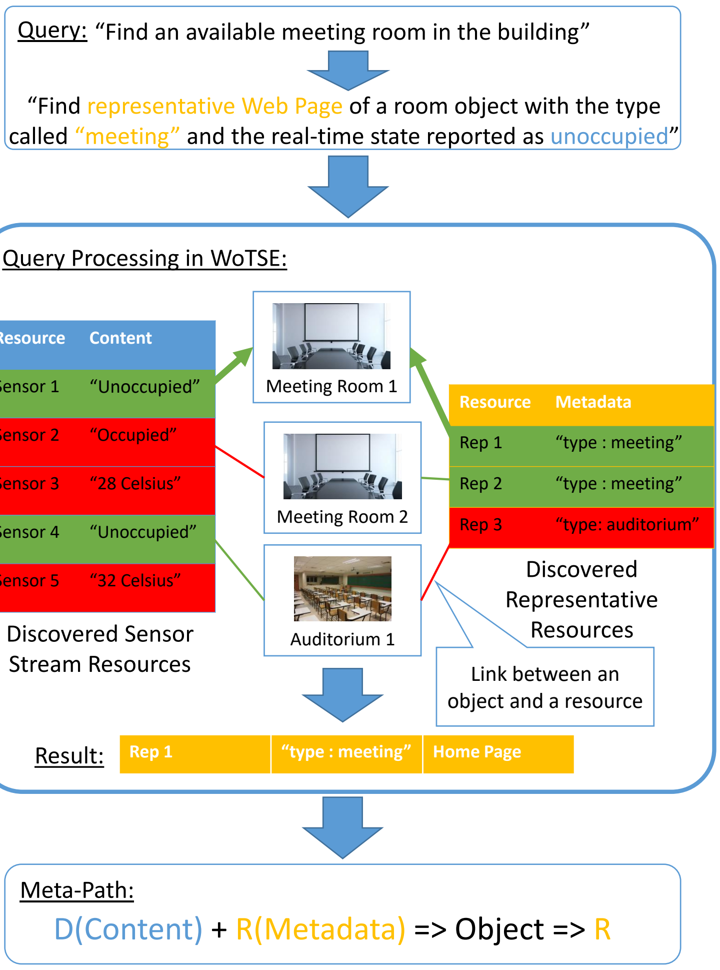

Query: "Find an available meeting room in the building"
"Find representative Web Page of a room object with the type called "meeting" and the real-time state reported as unoccupied"
Query Processing in WoTSE:
Resource
Content
Sensor 1
"Unoccupied"
Sensor 2
"Occupied"
Sensor 3
"28 Celsius"
Sensor 4
"Unoccupied"
Sensor 5
"32 Celsius"
Discovered Sensor Stream Resources
Meeting Room 1
Meeting Room 2
Auditorium 1
Resource
Metadata
Rep 1
"type : meeting"
Rep 2
"type : meeting"
Rep 3
"type: auditorium"
Discovered Representative Resources
Link between an object and a resource
Result:
Rep 1
"type : meeting"
Home Page
Meta-Path:
D(Content) + R(Metadata) => Object => R

Fig. 2.3 Assessment of a query for available meeting room in a smart building and its related meta-path

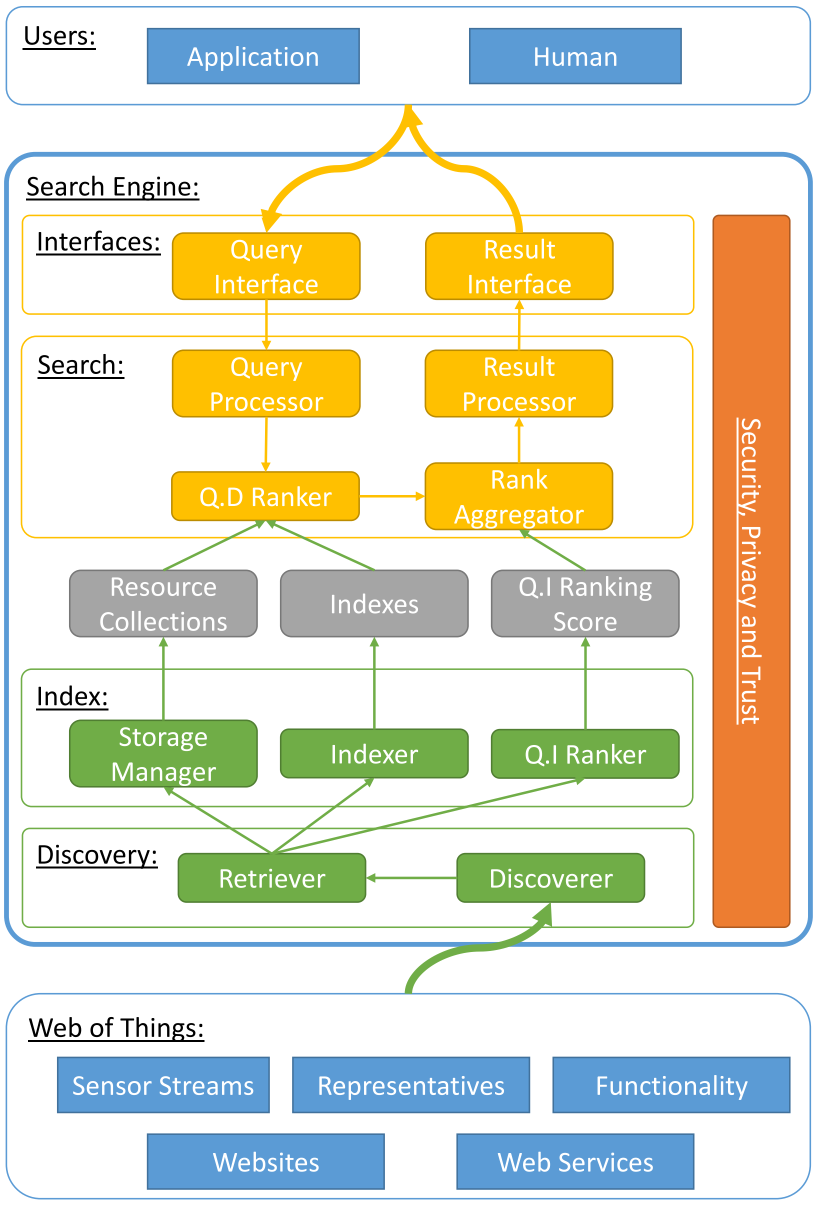


Fig. 2.4 A Modular Architecture for Web of Things Search Engines

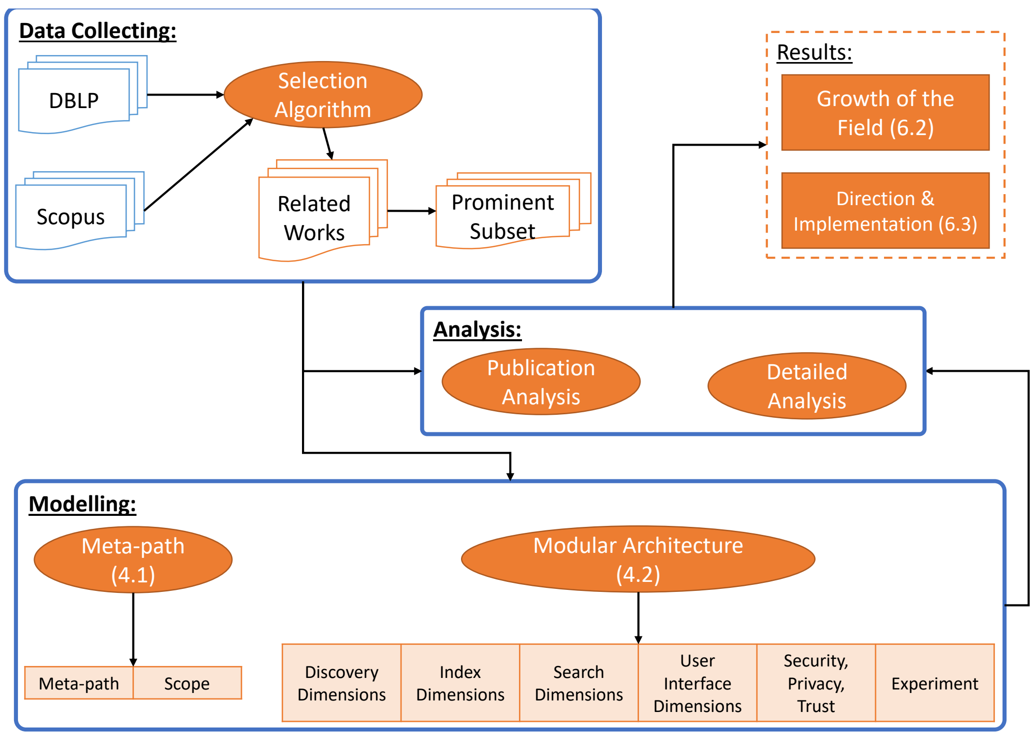
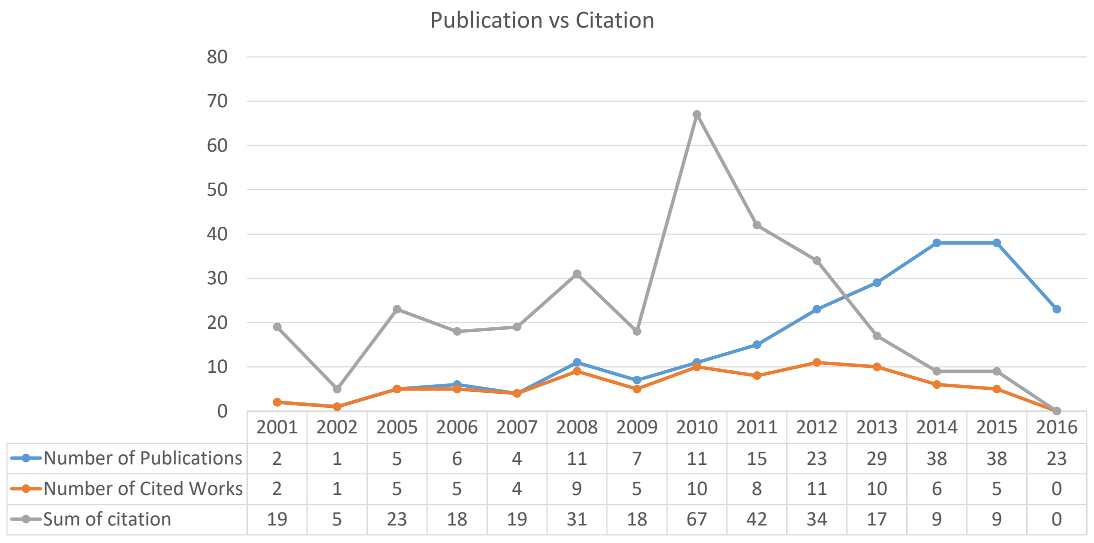
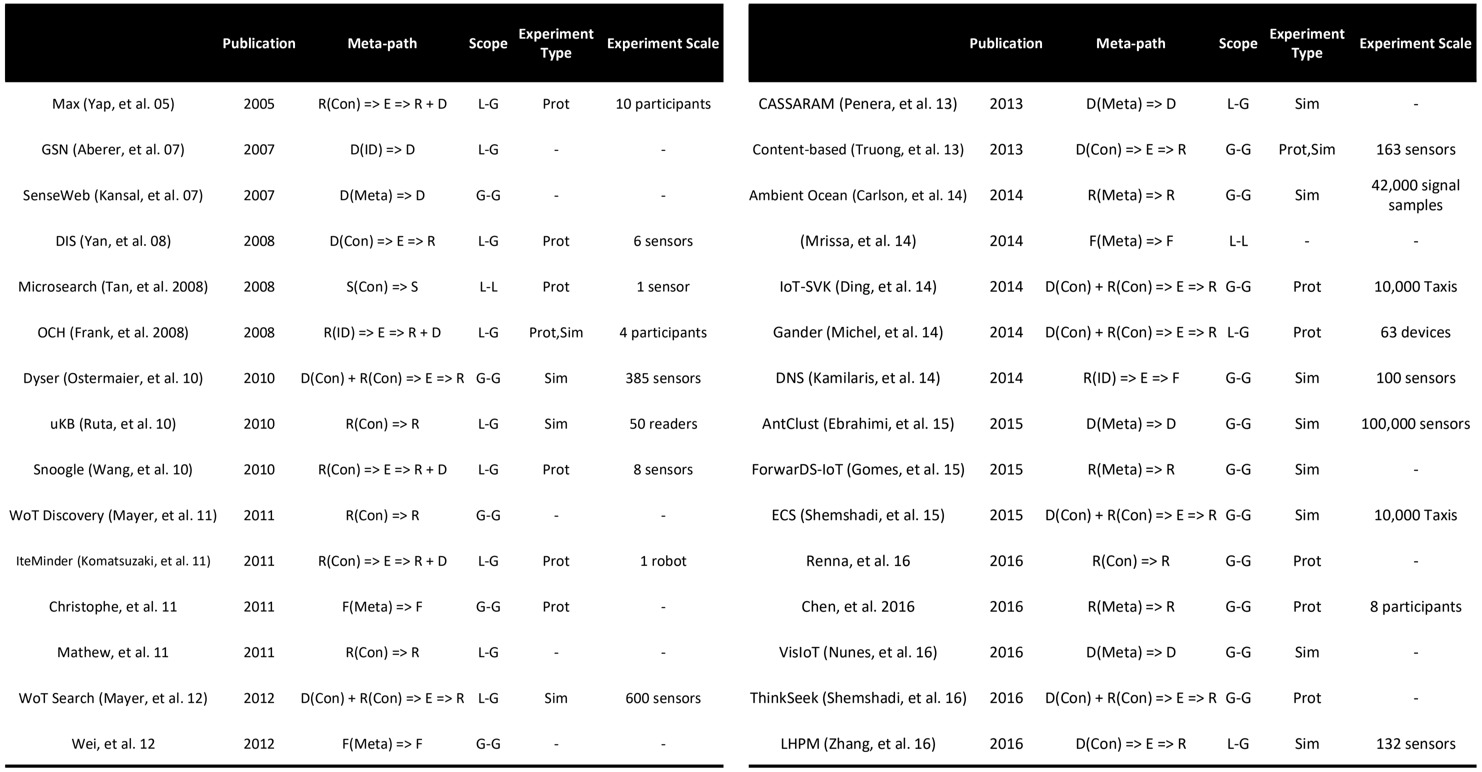


Fig. 2.5 Overview of the analytical framework. Oval objects represent components that we created. Dash arrows denote that the pointed object is derived from the pointing object. Solid arrows represent represents the link between inputs and outputs of our analysis.



	2001	2002	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Number of Publications	2	1	5	6	4	11	7	11	15	23	29	38	38	23
Number of Cited Works	2	1	5	5	4	9	5	10	8	11	10	6	5	0
Sum of citation	19	5	23	18	19	31	18	67	42	34	17	9	9	0

Fig. 2.6 Number of publications, in-field cited works and in-field citations



	Publication	Meta-path	Scope	Experiment Type	Experiment Scale
Max (Yap, et al. 05)	2005	R(Con) => E => R + D	L-G	Prot	10 participants
GSN (Aberer, et al. 07)	2007	D(ID) => D	L-G	-	-
SenseWeb (Kansal, et al. 07)	2007	D(Meta) => D	G-G	-	-
DIS (Yan, et al. 08)	2008	D(Con) => E => R	L-G	Prot	6 sensors
Microsearch (Tan, et al. 2008)	2008	S(Con) => S	L-L	Prot	1 sensor
OCH (Frank, et al. 2008)	2008	R(ID) => E => R + D	L-G	Prot,Sim	4 participants
Dyser (Ostermaier, et al. 10)	2010	D(Con) + R(Con) => E => R	G-G	Sim	385 sensors
uKB (Ruta, et al. 10)	2010	R(Con) => R	L-G	Sim	50 readers
Snoogle (Wang, et al. 10)	2010	R(Con) => E => R + D	L-G	Prot	8 sensors
WoT Discovery (Mayer, et al. 11)	2011	R(Con) => R	G-G	-	-
IteMinder (Komatsuzaki, et al. 11)	2011	R(Con) => E => R + D	L-G	Prot	1 robot
Christophe, et al. 11	2011	F(Meta) => F	G-G	Prot	-
Mathew, et al. 11	2011	R(Con) => R	L-G	-	-
WoT Search (Mayer, et al. 12)	2012	D(Con) + R(Con) => E => R	L-G	Sim	600 sensors
Wei, et al. 12	2012	F(Meta) => F	G-G	-	-
CASSARAM (Penera, et al. 13)	2013	D(Meta) => D	L-G	Sim	-
Content-based (Truong, et al. 13)	2013	D(Con) => E => R	G-G	Prot,Sim	163 sensors
Ambient Ocean (Carlson, et al. 14)	2014	R(Meta) => R	G-G	Sim	42,000 signal samples
(Mrissa, et al. 14)	2014	F(Meta) => F	L-L	-	-
IoT-SVK (Ding, et al. 14)	2014	D(Con) + R(Con) => E => R	G-G	Prot	10,000 Taxis
Gander (Michel, et al. 14)	2014	D(Con) + R(Con) => E => R	L-G	Prot	63 devices
DNS (Kamilaris, et al. 14)	2014	R(ID) => E => F	G-G	Sim	100 sensors
AntClust (Ebrahimi, et al. 15)	2015	D(Meta) => D	G-G	Sim	100,000 sensors
ForwarDS-IoT (Gomes, et al. 15)	2015	R(Meta) => R	G-G	Sim	-
ECS (Shemshadi, et al. 15)	2015	D(Con) + R(Con) => E => R	G-G	Sim	10,000 Taxis
Renna, et al. 16	2016	R(Con) => R	G-G	Prot	-
Chen, et al. 2016	2016	R(Meta) => R	G-G	Prot	8 participants
VisIoT (Nunes, et al. 16)	2016	D(Meta) => D	G-G	Sim	-
ThinkSeek (Shemshadi, et al. 16)	2016	D(Con) + R(Con) => E => R	G-G	Prot	-
LHPM (Zhang, et al. 16)	2016	D(Con) => E => R	L-G	Sim	132 sensors

Fig. 2.7 Meta-path Types, Scope and Experiment Scale of selected Prototypes.

	Discovery Scheme	Mobility Support	Collector Type	Collection Type	Index Type	Q.I Ranking	Storage Scalability	Search Scheme	Query Model	Result Model	Q.D Ranking	Adaptability	Search Scalability	User Type	Interface Modal	Query Interface	Result Interface	Security	Privacy	Trust
Max (Yap, et al. 05)	A	AHP	LD	V	-	-	V	AH	Txt	L	D(Rnk)	-	-	H	W	TBx	L	-	OAC,LAC	-
GSN (Aberer, et al. 07)	A	T	LD	R	-	-	-	AH	ID	Str	Ext	-	-	M	-	-	-	Crpt	OAC	-
SenseWeb (Kansal, et al. 07)	P	MP	R	R	U/S	-	-	AH	Cond	Str	D(Rnk)	-	Cch	M	API	API	-	-	OAC, Sum	R
DIS (Yan, et al. 08)	P	-	R	R	Txt	-	D	AH+C	Txt	L	D(Rnk)	-	D	H	App	-	L	-	-	-
Microsearch (Tan, et al. 2008)	P	-	R	R	Txt	-	-	AH	Txt	L	D(Rnk)	-	-	H	App	TBx	L	-	-	-
OCH (Frank, et al. 2008)	A	AHP	LD	V	-	-	V	AH	ID	S	Ext	-	Scp	H	App	Imp	S	-	OAC	-
Dyser (Ostermaier, et al. 10)	A	-	Crwl	R	PM	-	-	AH	Txt+ Cond	L	D(Rnk)+ Ext	-	Scp	H	W	TBx	L	-	-	-
uKB (Ruta, et al. 10)	A	AHP	LD	V	-	-	V	AH	Cond	L	Ext	-	-	M	-	-	-	-	-	-
Snoogle (Wang, et al. 10)	P	T+B	R	R	Txt	-	D	AH	Txt	L	D(Rnk)	-	D	H	App	TBx	L	Crpt	OAC	-
WoT Discovery (Mayer, et al. 11)	P	-	R	V	-	-	V	AH	ID	S	Ext	-	-	M	API	API	S	-	-	-
IteMinder (Komatsuzaki, et al. 11)	A	-	LD	R	-	-	-	AH	Txt	S	Ext	-	-	H	App	TBx	M	-	-	-
Christophe, et al. 11	P	-	R	R	U/S	-	-	AH	Cond	L	D(Rnk)	ReqT	-	H+M	App	Imp	-	-	-	-
Mathew, et al. 11	P	-	R	R	-	-	-	AH	Cond	L	Ext	-	-	H	App	TBx	L	-	-	-
WoT Search (Mayer, et al. 12)	A	AHP	LD	R	-	-	D	AH	Txt+ Cond	L	D(Rnk)	-	D	H+M	API	API	L	-	-	-
Wei, et al. 12	P	-	R	R	-	QoS	-	AH	Cond	L	Ext	-	-	M	API	API	L	-	-	-

Fig. 2.8 Implementation of selected prototypes. “-” denotes that the feature is not described, while “N/A” denotes that the feature is not implemented.

	Discovery Scheme	Mobility Support	Collector Type	Collection Type	Index Type	Q.I Ranking	Storage Scalability	Search Scheme	Query Model	Result Model	Q.D Ranking	Adaptability	Search Scalability	User Type	Interface Modal	Query Interface	Result Interface	Security	Privacy	Trust
CASSARAM (Penera, et al. 13)	A	-	LD	R	-	-	-	AH	Cond	L	D(Rnk)+ Ext	-	-	H+M	App+API	TBx+API	L	-	-	-
Content-based (Truong, et al. 13)	A	-	Crwl	R	PM	-	D	AH	Cond	L	P(Rnk)	-	D	H	W	F	L	-	-	-
Ambient Ocean (Carlson, et al. 14)	A	-	Crwl	R	U/S	R	-	AH	Txt	L	D(Rnk)	-	-	H	App	TBx+Sen	L	-	-	-
(Mrissa, et al. 14)	P	-	R	R	-	-	-	AH	Cond	L	Ext	-	-	M	-	-	-	-	F	-
IoT-SVK (Ding, et al. 14)	P	T	Reg	R	Txt,S,V	-	D	AH	Txt+ Cond	L	Ext	-	D	H	-	-	-	-	-	-
Gander (Michel, et al. 14)	A	AHP	LD	V	-	-	V	AH+C	Cond	L	Ext	-	D	M	API	API	L	-	-	-
DNS (Kamilaris, et al. 14)	P	-	R	R	-	-	D	AH	ID	L	Ext	-	D	M	API	API	L	-	-	-
AntClust (Ebrahimi, et al. 15)	P	-	R	R	Clst	-	-	AH	Cond	L	D(Rnk)	-	-	M	API	API	L	-	-	-
ForwarDS-IoT (Gomes, et al. 15)	P	-	R	R	-	-	D	AH+C	Cond	L	Ext	-	-	H+M	App+API	TBx+API	L	-	-	-
ECS (Shemshadi, et al. 15)	A	T	Crwl	R	Clst	-	-	AH	Txt+ Cond	L	D(Rnk)	-	-	H	W	TBx	M	-	-	-
Renna, et al. 16	-	-	-	R	-	-	-	AH	Txt	L	D(Rnk)	-	-	M	-	-	-	-	-	-
Chen, et al. 2016	-	-	-	R	Clst	-	-	AH	Txt+ Cond	L	D(Rnk)	-	-	H	App	TBx	L	-	-	-
VisIoT (Nunes, et al. 16)	P	-	R	R	-	-	-	AH	Cond	L	D(Rnk)	-	-	H	W	F	M	-	-	-
ThinkSeek (Shemshadi, et al. 16)	A	T	Crwl	R	Clst	-	-	AH	Txt+ Cond	L	D(Rnk)	-	-	H	W	TBx	M	-	-	-
LHPM (Zhang, et al. 16)	P	-	R	R	PM	-	D	AH	Cond	L	P(Rnk)	-	-	M	API	API	L	-	-	-

Fig. 2.9 (Cont) Implementation of selected prototypes.

Proxy (**MP**) denotes the use of spatially deployed proxies to query for resources at specific locations without having to keep track of their mobility [47].

The Collection Type includes **R**eal and **V**irtual collections. Index Type includes Text-based (**Txt**) indexes, which also include image-based indexes that treat images as a set of terms [45], **S**patial Indexes, numerical **V**alue Indexes, **Clust**ering mechanisms, Prediction Models (**PM**) (e.g., Sensor Rank [21]) and Unspecified Indexes (**U/S**) denoting indexing schemes that are mentioned but not described by the prototype. Q.I Ranking dimension includes the use of Quality-of-Service (**QoS**) and **R**atings from community. Storage Scalability support includes the use of **V**irtual resource collections to negate the need of actual storage and **D**istribution of resource storage over multiple instances of the search engine.

The Search Scheme dimension includes Ad-hoc (**AH**) and **C**ontinuous search, denoting whether the given queries are matched one time against the current snap shot of the resource collection or continuously assessed against the updating collection. The Query Model dimension includes Text-based queries (**Txt**), Logical **Cond**itions and **ID**entity. The Search Result can be a **L**ist of matching data records, a **S**ingle record, or a **Str**eam of dynamic information (e.g., sensor readings). Q.D Ranking includes ranking based on the value of prediction models (**P(Rnk)**), distance-based ranking (**D(Rnk)**), which can be expressed by Euclidean distance, Jaccard index or Cosine similarity, and exact matching (**Ext**). We consider Text-based ranking (e.g., TF-IDF) a form of D(Rnk). Search Scalability mechanisms include the **D**istribution of query processing, caching (**Cch**) search results to reduce number of query sending to sensors (e.g., SenseWeb [47]) and scoping (**Scp**) to reduce the number sensors to assess. The adaptability dimension includes only one value - **ReqT** - which denotes the ability of a search engine to detect and adapt its algorithm to the type of user making the request.

The Interface Modal denotes the channel of communication between a search engine and search users, including **W**eb Interface, Web **API** and specialized **APP**lication. The form of interface on this channel to receive queries from users includes structured **F**orms, text boxes (**TBx**), **Sen**sors on client device, and **Impl**icit queries invoked by the interaction between users and client application (e.g., [22]). Result interface includes the traditional **L**ist of records and the geographical **M**ap.

Security measures of IoTSE include encryption (**Crpt**). Privacy in IoTSE is protected by enforcing access control on objects (**OAC**) and spatial locations (**LAC**), **Sum**marizing sensor data and **F**iltering of search results to protect sensitive information of involved users. Finally, on Trust dimension, we have the value **R** denoting the use of ratings from the community.

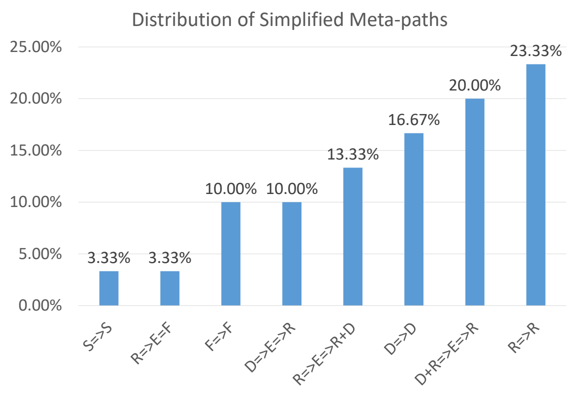
Form of IoTSE

Figure 2.10(a) presents the distribution of meta-paths supported by the selected prototypes. Searching for objects based on their ID or metadata ($R \Rightarrow R$) is the most common form of IoTSE, followed closely by searching for objects using their real-time state (e.g., sensor readings, location) and searching for sensor streams ($D \Rightarrow D$). Familiarity is a possible explanation for the popularity of these meta-paths. For instance, $R \Rightarrow R$ is similar to Web search, while $R + D \Rightarrow Obj \Rightarrow R$ comes naturally with the idea of feeding real-world states into software applications. Surprisingly, searching for real-world functionality is not commonly supported even though it is crucial in the interaction with IoT-enabled smart environments.

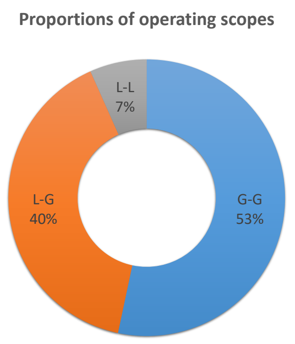
Figure 2.10(b) presents the distribution of operating scopes of selected prototypes. Local and global resource discovery are equally supported by the prototypes, which reflects the attention to both ends of the IoT scale. However, global scope dominates the search operation.

Implementation of IoTSE

Figure 2.11 presents the support that key dimensions receive from the selected prototypes. Q.D ranking and discovery enjoy the strongest support, as they are the core of an IoTSE. Scalability of query processing and storage capability is supported by about half of prototypes.



(a) Meta-paths



(b) Scopes

Fig. 2.10 Distribution of Meta-paths and Operating Scopes

Most supporting prototypes scale up by utilizing virtual resource collections and distributing processing and storage across multiple computers. Mobility of physical objects is considered by less than 40% of the selected prototypes. The weak support for indexing is a surprising result, considering its crucial role in resolving queries. A possible explanation for this phenomenon is the simplicity of usage scenarios and involving resources in the selected prototypes.

Most prototypes do not support adaptability which means that they cannot change their operations according to context, such as their current users. Query Independent (Q.I) ranking also lacks support, even though it plays a crucial role in the success of Web Search engines. It can be contributed to the lack of natural order of IoT resources. Security, privacy and trust are also not commonly addressed by prototypes.

Figure 2.12 presents the details of some interesting dimensions that receive high support. On discovery scheme, the active and passive schemes are equally utilized. This is a surprising result because both Web Search and Sensor Search systems, which are frequently considered predecessors of IoTSE, rely on active discovery scheme. On collection type dimension, real collections dominate because it is most straightforward and traditional solution in search engines. Search scheme is dominated by ad-hoc search scheme, which is carried over from Web Search Engines. On targeted user type dimension, human users have a slight edge over

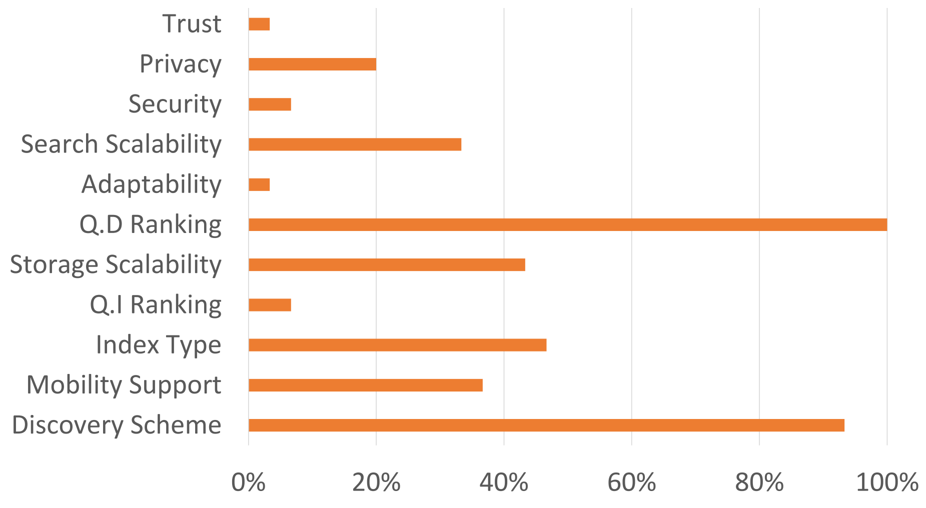


Fig. 2.11 Support of Prototypes on key Dimensions

machine users. Interestingly, some IoTSE are designed to support both types of users. On Query Dependent (Q.D) ranking dimension, distance-based ranking and exact matching are two most common forms of ranking mechanisms among the selected prototypes. On the query model dimensions, logical conditions is the most common form of query, while list of resources is the most common result model.

2.6 Industrial Works and Standards

2.6.1 Overview of Industrial Works and Standards

We define industrial works as publicly deployed and, optionally, commercialized products and services. We select two groups of industrial works for our evaluation based on references of research prototypes and IoT news sources. The first group is stand-alone IoT Search Engines. *Shodan* [8] proclaims to be "the world's first search engine for Internet-connected devices". It is designed and deployed by John Matherly in 2009. *Censys* [9] [63] search engine,

[8]https://www.shodan.io/
[9]https://censys.io/

Discovery Scheme	Count	Proportion
Active	13	48.15%
Passive	14	51.85%
Grand Total	**27**	**100.00%**

(a) Discovery Scheme

Collection Type	Count	Proportion
R	25	83.33%
V	5	16.67%
Grand Total	**30**	**100.00%**

(b) Collection Type

Search Scheme	Count	Proportion
AH	27	90.00%
AH+C	3	10.00%
Grand Total	**30**	**100.00%**

(c) Search Scheme

User Type	Count	Proportion
H	15	50.00%
H+M	4	13.33%
M	11	36.67%
Grand Total	**30**	**100.00%**

(d) User Type

Q.D Ranking	Count	Proportion
D(Rnk)	14	46.67%
D(Rnk)+Ext	2	6.67%
Ext	12	40.00%
P(Rnk)	2	6.67%
Grand Total	**30**	**100.00%**

(e) Q.D Ranking

Search/Result	L	S	Str	Grand Total
Cond	12		1	13
ID	1	2	1	4
Txt	6	1		7
Txt+Cond	6			6
Grand Total	**25**	**3**	**2**	**30**

(f) Query and Result Models

Fig. 2.12 Statistics of key Dimensions

deployed by the University of Michigan in 2015, and *Qadium*, which raised over 20 million dollars in fundings by 2016 [10], offer similar services. Essentially, these systems are tools for performing Internet-wide studies. However, they can be adapted to search for devices in the IoT. Shodan and Censys can detect and access a wide range of vulnerable network devices from Webcams, baby monitors, to ATM and medical devices [11]. *Thingful* search engine [12], on the other hand, is designed specifically for the WoT. Instead of pinging public IPv4 addresses, Thingful builds its dataset from sensor data sources on the Web. These resources are exposed for searching via a graphical map.

The second group is search mechanism offered within commercial IoT Cloud Platform (e.g., Amazon Web Service IoT Platform (AWS IoT) [13], IBM Watson IoT platform (Watson IoT) [14]). These search mechanisms operates on objects and resources of the searcher, linked

[10]http://www.forbes.com/sites/thomasbrewster/2016/06/05/qadium-iot-google-security-darpa-cia/#7289d6c42722
[11]https://blog.kaspersky.com/shodan-censys/11430/
[12]https://thingful.net/
[13]https://aws.amazon.com/iot/how-it-works/
[14]http://www.ibm.com/internet-of-things/

to the platform. The offered search capability is basic, such as filtering objects by their ID and metadata. It should be noted that while searched objects can be physically distributed across the globe, the scope of search capability offered to a user is still limited in his own "silo" of data.

We select standards for analysis based on the technical landscape of IoT Interest Group [64] and references of the research prototypes. We focus on standards that specify the whole discovery and search process, and select EPCglobal Discovery Service [15], BRIDGE Discovery Service (WP2) [65], and Afilias Extensible Supply-chain Discovery Service (ESDS) [66, 67]. These standards revolve around "Discovery Service", which finds Information Systems in a network (e.g., Internet) that hold the information corresponding to a given object identifier. Therefore, from Meta-path perspective, these standards are very similar.

2.6.2 Evaluation

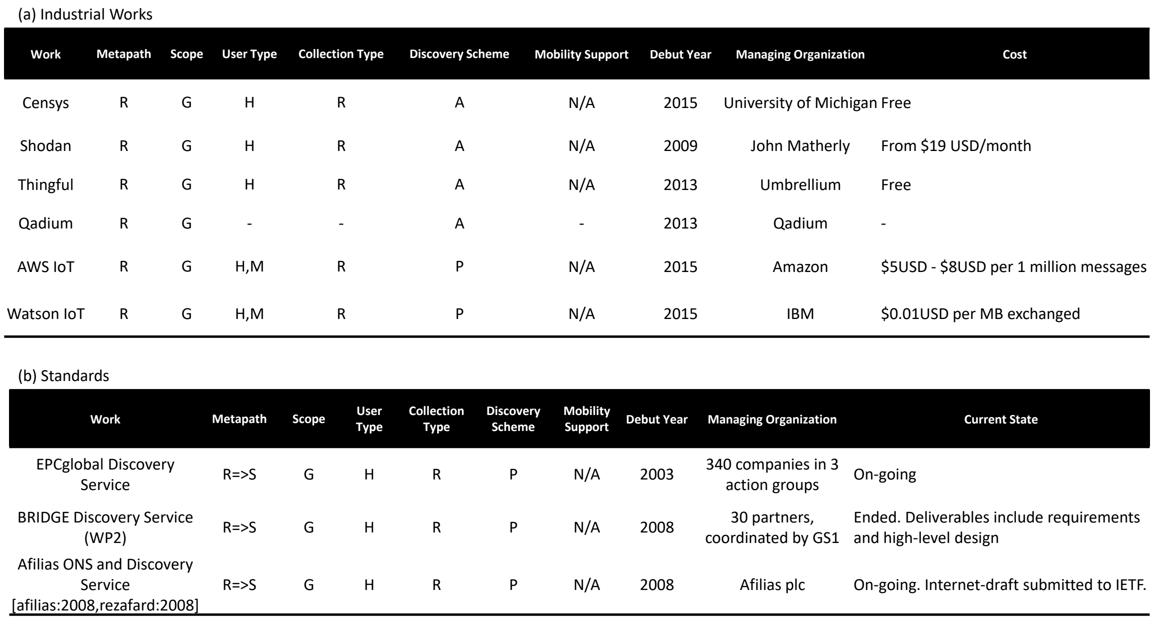
Figure 2.13 presents the evaluation result of industrial works and standards on a subset of our dimensions.

Comparing to the result of academic prototypes, industrial works and standards are considerably less "adventurous". They converge to searching for objects based on their static information such as ID and metadata, which is arguably the most natural step from the existing Web Search Engines. And, in the current state of IoT, this form might be all it takes to reap benefits from the emerging library of the real world.

2.7 Discussions

The goal of IoTSE is building an "ideal" search engine that can find *"anything"*, at *"anywhere"* and *"anytime"*. "Anything" means it can work with any meta-path, involving any combination

[15]http://www.gs1.org/epcrfid/epc-rfid-dci/1



(a) Industrial Works

Work	Metapath	Scope	User Type	Collection Type	Discovery Scheme	Mobility Support	Debut Year	Managing Organization	Cost
Censys	R	G	H	R	A	N/A	2015	University of Michigan	Free
Shodan	R	G	H	R	A	N/A	2009	John Matherly	From $19 USD/month
Thingful	R	G	H	R	A	N/A	2013	Umbrellium	Free
Qadium	R	G	-	-	A	-	2013	Qadium	-
AWS IoT	R	G	H,M	R	P	N/A	2015	Amazon	$5USD - $8USD per 1 million messages
Watson IoT	R	G	H,M	R	P	N/A	2015	IBM	$0.01USD per MB exchanged

(b) Standards

Work	Metapath	Scope	User Type	Collection Type	Discovery Scheme	Mobility Support	Debut Year	Managing Organization	Current State
EPCglobal Discovery Service	R=>S	G	H	R	P	N/A	2003	340 companies in 3 action groups	On-going
BRIDGE Discovery Service (WP2)	R=>S	G	H	R	P	N/A	2008	30 partners, coordinated by GS1	Ended. Deliverables include requirements and high-level design
Afilias ONS and Discovery Service [afilias:2008,rezafard:2008]	R=>S	G	H	R	P	N/A	2008	Afilias plc	On-going. Internet-draft submitted to IETF.

Fig. 2.13 Evaluation of IoT search engine industrial efforts (a) and standards (b)

of IoT resources. “Anywhere” means it can utilize the spatial information to objects located at any specific location, in any specific area. “Anytime” means it can utilize the whole range of IoT data, from the archived sensor readings to current sensing data to the data that will be produced in the future to match queries with resources. Resources returned by an ideal IoTSE not only have relevant content, but they also have that content at the relevant time, at the relevant place. An ideal IoTSE is the gateway to the Web of Things.

Moving toward this vision from the current state of the art requires us to address a wide range of issues. In this section, we discuss prominent ones.

2.7.1 Crawling IoT

Constructing resource collections automatically via crawling is desirable in IoTSE. However, this task is very challenging. The first issue is *detecting IoT data sources*. These sources can be organized into four groups [68]: cloud-based IoT platforms (e.g., Amazon Web Service IoT Platform, IBM Watson IoT platform), live-maps such as real-time transportation information services (e.g., FlightRadar24 [16]), urban crowdsensing services (e.g., Waze [17]) and public environmental sensing services. To detect these sources automatically, their features must be formally defined and mapped into machine-detectable traits of Websites. These criteria are not straightforward, even for human operators. For instance, should a live-map of lightnings [18] around the world be considered a IoT data source?

The second issue is *extracting resources automatically*. Resources are commonly transferred by XML HTTP Request (XHR) responses in form of XML or JSON documents. Currently, the URL pattern of these XHR must be detected manually [62]. The third issue is *automatic integration of resources*. High degree of overlapping in coverage is observed in the IoT data sources (e.g., flight data [62]). However, the data that they provide is not completely identical. Aggregating reports from different data sources can reveal a complete picture of collected resources. However, this automation is challenging due to the diversity of resource data fields and formats.

2.7.2 Supporting Location-based Search

Spatial information is crucial in searching IoT [28, 39]. The first challenge of providing location-based serach is *identifying locations*. Latitude, longitude and the height comparing to sea level together forms a potential location identifier. It is feasible in outdoor scenarios with sparse sensors. However, its granularity is challenged in indoor environments with dense distribution of objects. A potential solution is utilizing different coordinate systems with different granularity for different scenarios. However, this approach raises additional questions. For instance, how to recognize the utilized coordinate system? How to integrate different coordinate systems into a single index structure?

[16]http://flightradar24.com
[17]https://www.waze.com
[18]https://www.lightningmaps.org/?lang=en

The second problem is *associating coordinates with landmarks*. The role of landmarks to coordinates is similar to the role of domain names to IP addresses. For human users, landmark is preferable comparing to coordinates in both query and search results. For instance, a search result showing that the missing key chain is "under the desk in the dining room" is more intuitive than numerical coordinates. However, the query processing would be straightforward and unambiguous with numerical coordinates. Therefore, IoT Search Engines must be able to formally and semantically describe landmarks, and translate between coordinates and landmarks.

2.7.3 Supporting the Dynamic Nature of IoT

Mobility of physical objects and changing sensor readings reflect the dynamic nature of IoT. Detecting and storing changes are key problems. In the context of IoTSE, we consider the problem of storing changes. The critical issue is indexing the changing data.

The first issue of storing changes is *indexing*. As indexes on sensor measurements are outdated as soon as they are created [21], a balance must be achieved between the freshness of stored data and the communication overhead of pulling the latest sensor measurements. For instance, a naive solution is pulling readings from all detected sensors for every query received. This approach does guarantee the freshness of data, however the massive communication overhead negates any scaling up possibility. An emerging solution is indexing prediction models of sensors instead. These prediction models can be built on the assumption of the periodic nature of sensor measurements [21] or from the density and scalability of each sensor reading within a time frame prior to the query [15]. Based on the result of the indexed prediction model, a search engine contacts can limit the number of sensors to validate before building search results.

The second issue is *storing and purging the collected data*. A IoT Search Engine must find a balance between the number of old readings stored for resolving historical queries and building prediction models, and the scale resources that it manages, because each set of past measurements duplicates the whole resource collection. As a result, mechanisms for ensuring the scalability of the data storage such as distribution deployment and purging strategies must be investigated.

The final issue is supporting subscription-based queries and continuous query processing. These abilities allow search users to register their interest for a specific real-world state and receive relevant search results in the future when they are detected. Subscription and continuous query processing are discussed, but not implemented in the existing prototypes.

2.7.4 Supporting the Diversity

An ideal IoTSE must be able to work with many different types of resources and their combinations to resolve all given queries. Basic solutions are either building a search engine that is highly adaptable, or building a large number of specialized search engines for different resource combinations. The first solution might lead to "jack-of-all-trades" systems that are usable in many scenarios, but not particularly competent in any of them. In the second solution, the diversity of the search engines itself might become the problem.

A potential solution for this challenge is enabling modular construction of IoTSE, in which search engines are composed from a set of standardized modules according to the meta-path needed by a given query (Fig. 2.14). The analysis of existing works reveals that meta-paths of different IoTSE overlap to a certain degree. By turning the whole discovery and search process of IoTSE into standardized modules, we can reuse them in other IoTSE that has (partial) overlapping meta-paths. This method facilitates specialization. Involving parties can focus on only components that align with their expertise instead of having to build the whole system. These components are then easily shared with the community to leverage the improvement and development of other components to ensure that the global IoTSE is always optimal and ready to cope with any combination of resource types in IoT.

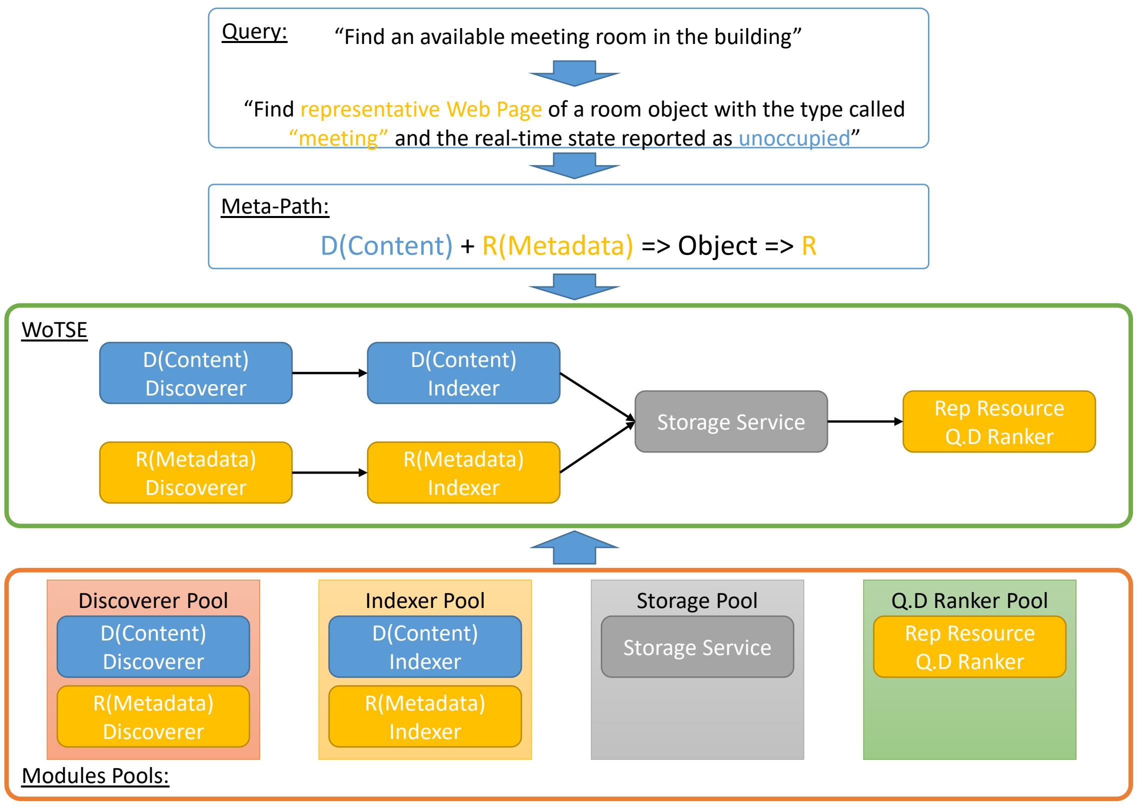


Fig. 2.14 Modular construction of Web of Things Search Engines

Two major issues in enabling modular construction are standards and security. For modules that are independently developed to work together, we must provide standards for interfaces between modules, their operations, characteristics and their arrangement as a system. We also need to ensure that modules actually do what they promise, and ensure that they are not bias in their operation. These are challenging endeavors.

2.7.5 Supporting Scalability

The scale of IoT extends to both extremes of the spectrum: it is expected to be 50 or 100 times larger than the existing Web, yet majority of its interaction would be in small sets of co-located objects and resources. Therefore, IoTSE must fit the search activity in local scale naturally, and at the same time, they must also be able to scale up to reach billions devices on the world. Scaling up centralized search engines is not a preferable solution because these

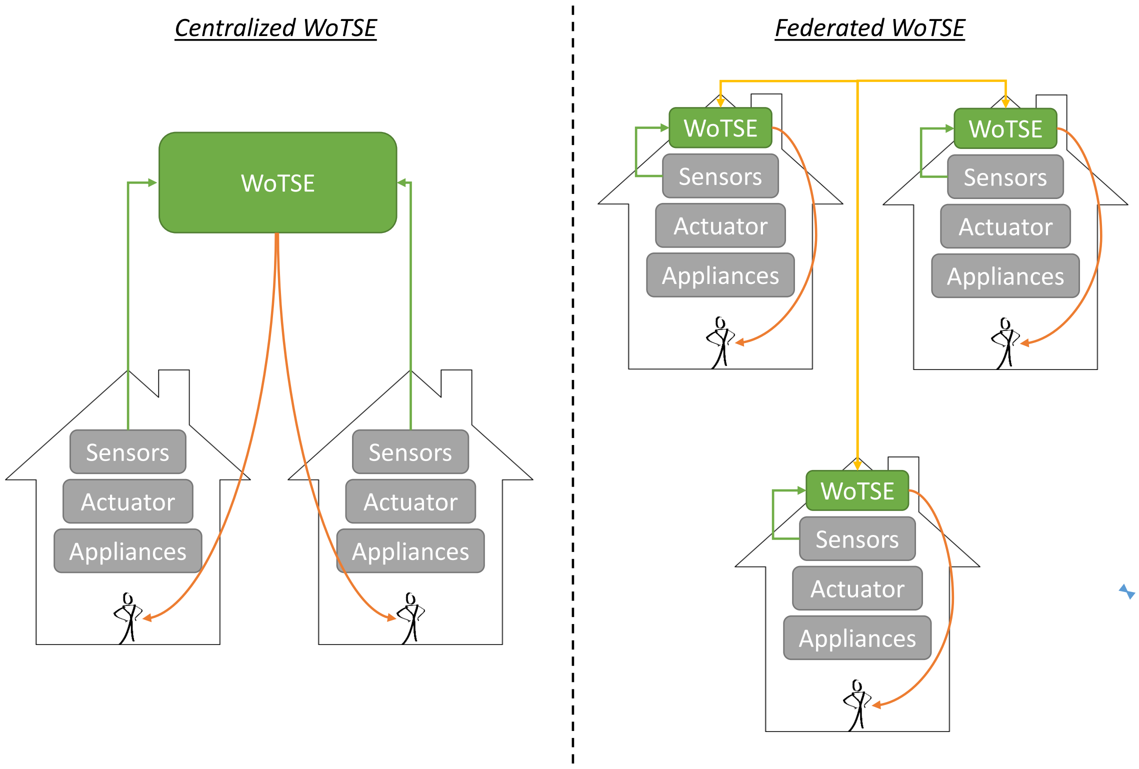


Fig. 2.15 Comparison between Federated and Centralized IoTSE

systems are too far from physical world, making them insensitive to changes. Moreover, these systems must be able to identify a massive number of private locations (i.e., rooms inside smart homes) and associate private objects with these locations, which is challenging both technically, socially and politically.

An alternative solution is distributing IoTSE closer to the edge of IoT and linking them into a federation to provide global coverage. The IoTSE can be hosted on a computer, or even a smart phone to provide services to all authorized applications in its immediate vicinity (Fig. 2.15). Distributing IoTSE to the edge of IoT makes them naturally fit for local search activity, while linking them together provides the coverage to address the upper ends of IoT scale. Two major issues in enabling federated IoTSE are building effective and efficient methods to manage this massive federation, and evaluating the trust of each member IoTSE in the federation.

2.7.6 Security, Privacy and Trust

As the gateway to IoT that is capable of finding “anything”, at “anywhere” and “anytime”, IoT Search Engines represent unprecedented security and privacy risks. For instance, IoTSE can be used to track a person over a broad area and time period for surveillance or other malicious purposes. It can also be used to find and attack unprotected Web-enabled vehicles and medical devices, as demonstrated by Shodan and Censys. Such a system can also be used to spy on the stockpile and the transportation fleet of a company, resulting in massive economical damage. On the national and international scale, IoTSE can be a dangerous tool for espionage and sabotage. Therefore, a key issue of IoT Search Engine is *protecting the privacy of searchers, information owners and sensed people*. This is a challenging task, because a person cannot opt out of being sensed by sensors. Moreover, we lack the mechanisms to define ubiquitously accessible privacy policies and mechanisms to enforce them.

The second issue is *validating the discovered IoT content*. As real-world information in IoT is provided by exposed electronic tags and sensors that can be breached and forged, a malicious party can inject false information into IoT, which would be distributed by IoT Search Engines. For instance, consider a restaurant recommendation system that infer the crowdedness of restaurants with public sensors retrieved via IoT Search Engines. Rivals of a restaurant can sabotage it by planting forged sensors that report extreme noise and movement in its vicinity, causing the restaurant to be inferred as full and removed from the recommendation list. This type of attack can drive the restaurant out of business and damage the trust of users in both the recommendation system and the IoT Search Engine. A potential solution for this issue is validating the information received from sensors against past patterns and readings of their neighboring sensors. Another potential solution is building the audit-ability into IoTSE. Ensuring that one would be held accountable for his malicious activities is a powerful preventive mechanism.

2.8 Related Work

A range of surveys on the WoT and its closely related concept – the IoT – exist in the literature. Early surveys [9, 2, 6, 7] serve as road-maps to realize the IoT and the WoT. They cover visions, definitions, enabling technologies and propose potential research directions. Later surveys focus on more specific usages. [4] reviews IoT from the cloud computing perspective; [69] reviews the enabling technologies to extend the IoT into Cognitive IoT; [70] surveys the integration of humans' social network into the IoT to form a Social IoT; [8] explores different use cases of the IoT in smart cities and their enabling technologies; and [71] approaches the IoT from the politics and policy perspective. In these surveys, IoTSE either receives a brief discussion as a potential research topic [6] or a short introduction presenting some representative works [43].

A small number of surveys specifically on IoTSE exist in the literature. They either focus on one type of IoTSE or listing potential research problems without considering the state of the field. [27] analyze seven prototypes on nine dimensions that focus on the ability to handle real-time, local sensor queries. [28] performs a similar analysis with six prototypes on 14 dimensions. [29] approaches IoTSE from perspective of EPCglobal's Discovery Service. They analyze five works on nine dimensions. Finally, [30] evaluates 49 works, including both IoT-specific prototypes and results from other fields that are expected to be applicable to IoT, on nine dimensions. Selected works are analyzed on their basic operating principles, data representation and type of searched content.

Our survey addresses the limitations of the existing surveys. We retrieve over 200 related works to build a flexible model for describing IoTSE and a modular architecture for assessing their implementation. The resulting analytical framework from our models is used to assess the growth and the state of all major types of IoTSE in the literature. Table 2.3 presents the comparison between our survey and the existing ones.

Table 2.3 Comparison between our work and existing surveys on IoT search engine

	Prototypes	Dimensions	Assessment Focus
Romer, et al. 2010	7	9	Ability of IoTSE to handle real-time sensor data and perform local search
Zhang, et al. 2011	6	14	Ability of IoTSE to handle real-time sensor data and perform local search
Evdokimov, et al. 2010	5	9	Maturity of Discovery Service architectures and prototypes
Zhou, et al. 2016	49	9	Technologies and techniques transferable to IoTSE
Our Survey	**214**	**24**	**Forms, Implementation of IoTSE and the current state of the field**

2.9 Summary

The World is becoming a library of resources for software applications, thanks to the Internet of Things. The Internet of Things Search Engines enable the optimal utilization of this emerging library. The diversity of the solution space and the scale of the IoT are the main challenges facing IoTSE.

Our survey on over 200 academic and industrial works related to IoTSE confirms the continuous expansion of the field. It also reveals skewness in the attention that these works receive from their contemporaries. Searching for real-world objects, based on their real-world state is by far the most popular form of IoTSE.

Bridging the gap from the state-of-the-art to an ideal IoTSE that can “find anything, at anywhere and any time” requires addressing various open issues. Supporting the scale of the IoT is the first notable issue. The IoT is larger yet at the same time smaller than the World Wide Web. The total amount of the IoT content might be 50 to 100 times larger than the Web. However, a majority of interactions in the IoT would be among small sets of co-located objects and resources. A potential solution to this problem is engineering IoTSE as an extensive collection of instances, which are deployed near the edge of the IoT. Each instance monitors a small set of co-located objects and works with other instances to scale up and cover the anticipated 50 billion devices in 2020.

Supporting the diversity of the IoT is the second notable issue. An ideal IoTSE must be able to discover and query most types and combinations of IoT content. Such vision might be achieved by finding a “magical algorithm” that allows assessing any IoT content and developing an IoTSE instance around it. The other solution, which is arguably more realistic, is engineering a large number of IoTSE instances that specialize in different types and combinations of IoT content.

The potential solution to both the scale and the diversity problem point toward engineering a vast number of IoTSE instances. The analyses in this survey suggested the existence of significant overlaps in functionality and internal operation of IoTSE instances, which can be leveraged to enable reuse-centric engineering of IoTSE. These points motivate the research presented in this thesis.

Chapter 3

An IoTSE Reference Architecture

In the previous chapter, we have conducted an extensive survey of IoTSE literature and identified the diversity of IoTSE solution space one of the primary open issues in IoTSE research. We have also found notable overlaps regarding internal operations of IoTSE instances, demonstrated by the mapping of different IoTSE classes into the same modules. In this chapter, we leverage those insights to propose an IoTSE reference architecture, comprising 18 component types, 13 composition patterns, and 6 deployment patterns. We also introduce a framework for instantiating IoTSE instances based on the proposed reference architecture. For the evaluation, we map two representative IoTSE prototypes, which utilize complex composition and deployment patterns, to the constructs of the reference architecture to demonstrate their utility. By presenting a blueprint for engineering IoTSE instances, this chapter addresses the second research objective of this thesis: to propose a reference architecture that captures the commonalities of IoTSE.

3.1 Introduction

The aim of the research reported in this chapter is compiling the commonalities of IoTSE into a *Reference Architecture for the Internet of Things Search Engine*. This reference architecture captures functional elements and architectural patterns necessary for engineering IoTSE instances.

In developing the reference architecture, we limited our scope to only building blocks and patterns that are necessary for the core functionality of an IoTSE instance – to discover and resolve queries on IoT content. We also avoided the dependency on particular architectural styles and implementation technologies to ensure the broad applicability of the reference architecture. Due to this independence, the proposed architecture, for instance, can instantiate a Service-oriented Architecture (SOA) solution for IoTSE by combining with a Service-oriented Architecture Reference Architecture (e.g., ISO/IEC 18384 [72], S3 [73]) to incorporate building blocks necessary for enabling, monitoring, and securing a service-oriented IoTSE solution. The proposed architecture can also be mapped to a modular system such as OSGi (Open Services Gateway initiative) or even to a set of shared libraries.

The reference architecture serves four purposes. First, it offers a vocabulary comprising building blocks and patterns to enable the communication about IoTSE between involving parties [74]. Second, it helps to align the research and engineering effort on IoTSE across multiple parties by specifying the common architectural building blocks of these systems. Third, the reference architecture offers a framework for consolidating existing knowledge on designing IoTSE architectures to guide the engineering of new IoTSE instances. Finally, building blocks and patterns specified by the reference architecture lay a foundation for enabling the reuse-centric engineering of IoTSE.

The research for an IoTSE Reference Architecture presented in this chapter addresses the following three objectives:

Identifying Functional Elements Functional elements capture common activities across different IoTSE instances. They represent architectural building blocks, which are then realized by software components in a concrete IoTSE instance. The reference architecture specifies 18 functional elements, which are organized into 8 logical layers. They represent different areas of concern of IoTSE. Section 3.4.2 presents these results.

Identifying Composition Patterns of IoTSE A composition pattern is a named abstraction of interactions between functional elements to realize different tasks of an IoTSE instance. Section 3.4.3 presents a taxonomy and details of 13 composable composition patterns specified in the reference architecture.

Identifying Deployment Patterns of IoTSE A deployment pattern is a named abstraction of the organization of computing nodes hosting an IoTSE instance and the arrangement of functional elements on these nodes. While reference architectures generally focus only on the logical aspect of a software system, we include deployment patterns due to the increasingly important role of deployment information in describing an IoTSE instance. Section 3.4.4 presents a taxonomy and details of six patterns observed in the IoTSE literature.

3.2 Methods for Developing the IoTSE Reference Architecture

Reference architectures emerge from prior, proven concepts and experiences captured by previous systems and architectures [74]. Inputs of our reference architecture come from building blocks, patterns, and design decisions extracted from existing primary studies and the prototypes of IoTSE in the literature. The process utilized by our research consists of three phases.

3.2.1 Phase 1: Study Selection

The first phase is selecting primary studies relevant to our research. This phase has been conducted in the survey, presented in Chapter 2. To help readers follow the discussion, we introduce it again in more detail.

To ensure a comprehensive and unbiased selection, we systematized the study selection into a process, which was inspired by the study selection phase of the Systematic Literature Review (SLR) method [75]. To identify the potential primary studies, we performed a boolean search on the XML dataset of DBLP with the following query: "search OR discovery AND internet of things OR web of things". Different from the SLR method, we also carried out the "snowballing search", in which we retrieved references of the potential primary studies and built a graph of citations between them. The information from this graph helped us to identify the relevant studies that were published before the emergence of IoT and therefore missed by our boolean search. To ensure the completeness and integrity of these effort-intensive activities, we automated them with an in-house developed software tool.

We utilized the following inclusion and exclusion criteria to select the relevant studies:

- Including any study on a software system that is capable of both detecting and resolving queries on IoT content. The considered types of content include physical objects, smart appliances, sensors, actuators and content generated by these objects.
- Including any study on a software system that is only capable of resolving queries for physical objects, smart appliances, sensors, actuators and content generated by these objects.
- Excluding any primary study that addresses the content discovery problem on the physical and network layer exclusively, without considering query assessment.
- Excluding any primary study that utilizes sensing data to extend Web search.
- Excluding any secondary study (i.e., other reviews)

- Excluding any information retrieval study that does not involve the IoT or the WoT, *unless it is referenced by at least two included primary studies.*

The selection process results in 210 relevant work. We narrow this set to a representative subset of 36 work for the detailed analysis, data extraction and synthesis. These works are chosen in a way that balances high-cited work with new work to capture both the "norm" and evolution of the IoTSE research.

3.2.2 Phase 2: Information Extraction

The second phase of our process is extracting the relevant information from the chosen primary studies to address our research objectives. For identifying IoTSE building blocks and composition patterns, we extracted workflows – sequencing and timing of activities – from the representative IoTSE prototypes. Activities that overlap among workflows represent potential building blocks of IoTSE, while their sequencing and timing are possible composition patterns.

For identifying the deployment patterns of IoTSE, we extracted from each IoTSE prototype the arrangement of computing nodes that host its components, and the placement of its components on those nodes. The extracted information formed the basis for identifying and classifying the deployment patterns of IoTSE.

3.2.3 Phase 3: Architecture Design

In the last phase, we synthesized the extracted data into an IoTSE reference architecture iteratively. In each iteration, we processed the data from a batch of five studies and identified the possible building blocks, composition patterns, and deployment patterns of IoTSE. By the end of each iteration, we assessed the potential architecture against the IoTSE prototypes from the previous iterations and refined it. We continued this process until consuming all the extracted data. The architecture that emerged at the end of this process became the IoTSE reference architecture.

For identifying IoTSE building blocks, in each iteration, we derived possible common activities of IoTSE, each of which was described by its inputs, outputs, and its functional scope. A set of potential activities is relevant to an IoTSE prototype if the scope of every activity of the prototype is smaller than that of the potential activities. After each iteration, we adjust the set of common IoTSE activities and their functional scope. The remaining activities at the end of the process became the building blocks of IoTSE.

For identifying IoTSE composition patterns, we maintained a catalog of the sequencing of the IoTSE building blocks and updated it with the newly identified IoTSE building blocks by the end of every iteration. After processing all primary studies, we analyzed the catalog and decomposed these system-wide patterns into combinable sub-patterns, and composed them into a taxonomy of IoTSE composition patterns.

For identifying IoTSE deployment patterns, we maintained a classification of the deployment structure of IoTSE instances, regarding the type and the position of their involving computing nodes. We updated this classification by the end of every iteration. The one that emerged by the end of this process became the taxonomy of IoTSE deployment patterns.

3.3 Data Extraction Results

As described previously, we updated the results of data extraction after every iteration. The first version of the extraction results contains the verbatim text from the IoTSE prototypes, which describe their workflows and deployment structure using a variety of terminology. After every iteration, we refined the potential reference architecture and updated the extraction results with new terminologies from the refined reference architecture. In this section, we present the summations drawn from the final version of data extraction results.

Table 3.1 Distribution of interaction patterns extracted from IoTSE prototypes

Interaction Pattern	**Frequency**
Detect - Collect - Store // Receive Query - Search - Return Result	12
Detect - Collect - Store - Index // Receive Query - Search - Return Result	11
Detect - Collect - Process - Store - Index // Receive Query - Search - Return Result	5
Detect - Collect - Process - Store // Receive Query - Search - Return Result	2
Receive Query - Search - Return Result	2
Receive Query -Detect - Collect - Search - Return Result	2
Detect - Collect - Store // Receive Query - Process - Search - Return Result	1

3.3.1 Interaction Patterns

Table 3.1 presents the distribution of activity sequences utilised by the processed IoTSE prototypes. The $-$ symbol links activities that are carried out sequentially. The $//$ sign denotes that two sequences of activities before and after it are carried out in parallel. For instance, $A-B-C//D-E$ describes a pattern consisting of two sequences $A-B-C$ and $D-E$ which run in parallel.

3.3.2 Deployment Patterns

From each IoTSE prototype, we extracted the topology of its computing nodes physical proximity each computing node (i.e., edge, fog, and cloud) to the sensing and actuating infrastructure, and the arrangement of its components on computing nodes. Table 5.1 presents the distribution of topology and physical proximity of computing nodes utilized by processed IoTSE prototypes.

Table 3.2 Distribution of deployment patterns extracted from IoTSE prototypes

	Cloud	Cloud - Edge	Edge
Multiple peer nodes	1		7
One central node - Multiple leaf nodes	3	7	
One node	16		1

3.4 A Reference Architecture for IoTSE

3.4.1 Meta-model

The meta-model is a "model of model" that defines concepts and relationships appearing in a reference architecture [72]. Before diving into the details of the IoTSE reference architecture, it would be helpful to discuss its underlying meta-model. We utilize a meta-model derived from one underlying the Service-oriented Solution Stack (S3) [73] and the reference architecture for SOA solution specified in ISO/IEC 18384 standard [72]. Figure 3.1 depicts the meta-model underlying our reference architecture.

In our meta-model, a reference architecture comprises logical layers and patterns. Each *layer* represents a different area of concern in the system and has a set of functional requirements, which are represented by the set of capabilities that it offers. Each *capability* is specified in terms of the *inputs* that it utilizes and the *outputs* that it generates.

A layer comprises multiple *Architectural Building Blocks (ABB)*, which represents reusable functional elements of a system. ABBs realize capabilities that are offered by their encapsulating layers. They offer these capabilities either by themselves or by interacting with other ABBs both within and across layers.

Patterns are abstraction of the interaction between ABBs. Our meta-model utilizes two types of patterns. *Composition pattern* captures logical interactions between ABBs either

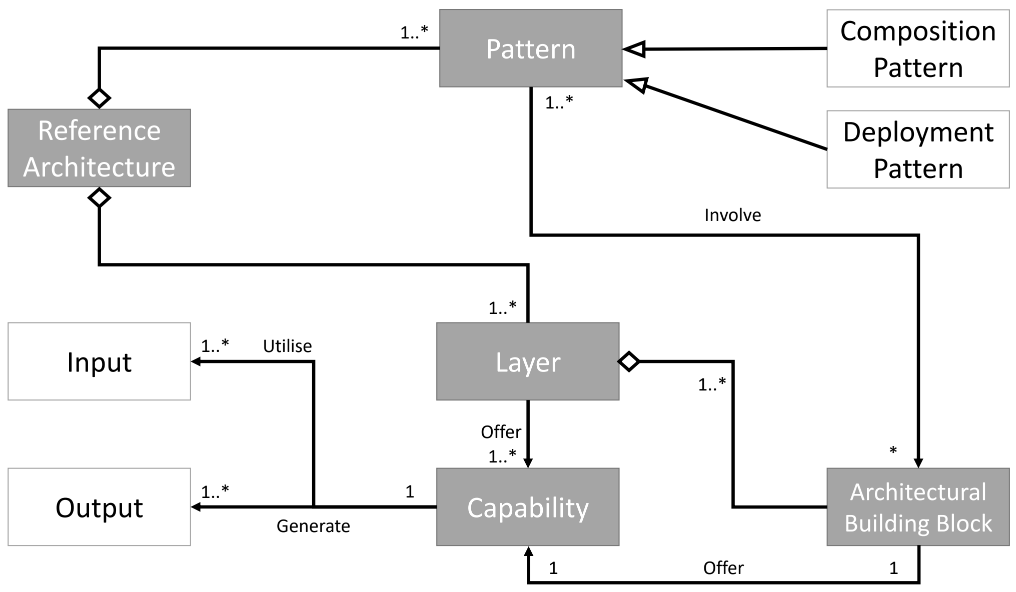


Fig. 3.1 Meta-model describing the proposed reference architecture.

within or across layers. *Deployment pattern* captures the deployment of ABBs on physical computing nodes, as well as the arrangement and interaction between these nodes.

3.4.2 Layers and Architectural Building Blocks

The proposed reference architecture for IoTSE consists of 18 architectural building blocks. By their area of concerns, we cluster these building blocks into five horizontal layers: *computing resource*, *discovery*, *storage*, *search*, and *interface*. The reference architecture also includes three vertical layers to capture cross-cutting concerns in IoTSE development and operation: *integration*, *management*, and *security*. Figure 3.2 depicts layers and architectural building blocks in the reference architecture. Table 3.3 presents the capabilities of each layer, and the building blocks offering those capabilities.

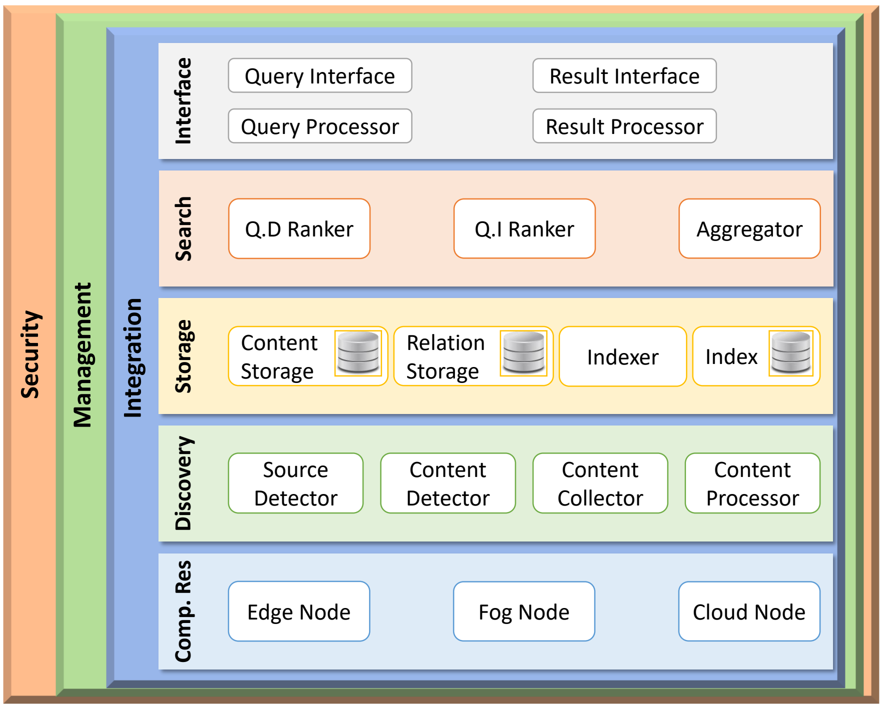


Fig. 3.2 Layers and architectural building blocks making the reference architecture for IoTSE.

Table 3.3 Specification of discovery, storage, search, and interface layer by their offering capabilities

Layer	**Capability**	**Input**	**Output**	**ABB**
Discovery	D1. Detect IoT content sources	Instructions	ID / URL of sources	Source Detector
	D2. Detect IoT content	ID / URL of sources	ID / URL of IoT content	Content Detector
	D3. Collect IoT content	ID / URL of IoT content	IoT content	Content Collector
	D4. Process IoT content (Clean, Transform, Derive)	IoT content	Processed content	Content Processor
Storage	S1. Generate index structures for efficient lookup	IoT content	Index	Indexer
	S2. Store IoT content	IoT content	N/A	Content Storage

Continuation of Table 3.3

Layer	**Capability**	**Input**	**Output**	**ABB**
	S3. Store relations between IoT content and things	Relations between content and things	N/A	Relation Storage
	S4. Store generated indexes	Index	N/A	Index
Search	Se1. Score and rank IoT content by their relevance to queries	Query, IoT content	Search Results	Q.D. Ranker
	Se2. Score and rank IoT content by their natural order	IoT content	Ranked list of IoT content	Q.I. Ranker
	Se3. Aggregate search results	Search results	Aggregated result	Aggregator
Interface	I1. Accept query	Interaction with external actors	Query	Query Interface
	I2. Process query into correct format	Raw query	Processed query	Query Processor
	I3. Return search results	Search Results	Interaction with external actors	Result Interface
	I4. Process search results into appropriate format	Search Results	Processes results	Result Processor

End of Table

Discovery Layer

Discovery layer concerns with detecting and collecting IoT content to build up collection of contents for answering queries. This layer offers four capabilities (Table 3.3).

D1. Detect IoT content sources Content sources denote cloud platforms, websites, as well as IoT-enabled things that offer IoT content. This capability receives instructions (e.g., seed URLs) and returns identifier or URL of sources as output. *Source Detector* building block encapsulates this capability.

D2. Detect IoT content Locating IoT content in a given source is non-trivial. The task of finding identifiers or addresses of specific pieces of IoT content in a source is encapsulated in this capability. It receives identifiers or URLs of IoT content sources as inputs and returns identifiers or addresses for accessing IoT content as outputs. It is handled by *Content Detector* building block.

D3. Collect IoT content Many IoTSE instances in the literature retrieve IoT contents as materials query assessment. This task is encapsulated in this capability. It receives identifiers or addresses and returns IoT content as outputs.*Content Collector* building block encapsulates this capability.

D4. Process IoT content Raw content retrieved from IoT infrastructures might be subjected to further processing for cleaning, denoising, transforming, or deriving higher-level information. Capability D4 encapsulates these activities. It receives raw IoT content and returns processed IoT content. *Content Processor* building block offers this capability.

Storage Layer

Storage layer concerns with storing the collected IoT content and generating data structures to support efficient lookup. Both of which are utilized in the query assessment that follows. This layer offers four capabilities (Table 3.3).

S1. Generate indexes for efficient look up Achieving sub-second query response time over an extensive collection of IoT content depends on data structures for supporting efficient lookup (i.e., indexes). Capability S1 encapsulates this activity. It receives the collected IoT content as inputs and generates indexes as outputs. *Indexer* building block offers this capability.

S2. Store IoT content Many IoTSE instances in the literature store the collected IoT content for future processing. Storage layer encapsulates this activity in the capability S2. It receives collected a IoT content as inputs and does not generate outputs. The *Content Storage* building block offers this capability.

S3. Store relations between IoT content and things Besides collected content, an IoTSE instance can also maintain the detected relations between IoT contents and things offering them, as well as thing-to-thing relations. This capability receives relations as inputs and does not generate outputs. The *Relation Storage* building block offers this capability.

S4. Store generated indexes This capability encapsulates the storage of data structures generated by the capability S1. It receives indexes as input and does not generate outputs. The *Index* building block offers this capability.

Search Layer

Search layer captures concerns of scoring and ranking IoT contents to generate search results. This layer offers three capabilities (Table 3.3).

Se1. Score and rank IoT content by their relevance to queries IoT content can be subjected to two forms of scoring and ranking. Capability Se1 captures the *query dependent (Q.D.)* scoring and ranking of IoT content. It receives queries, collected IoT contents, and, optionally, indexes as inputs. It generates search results which are generally a ranked list of scored IoT content. *Q.D. Ranker* building block offers this capability.

Se2. Score and rank IoT content by their natural order The second form of scoring and ranking, defined here as query independent (Q.I.) ranking, does not rely on queries, but only on the natural order of content. Page Rank algorithm is an example of Q.I. ranking. This capability receives IoT contents as inputs and generates a ranked list of IoT contents as outputs. *Q.I. Ranker* building block offers this capability.

Se3. Aggregate search results Query assessment in an IoTSE instance might involve multiple Q.D. and Q.I. rankers. The ability to combine multiple sets of search results, therefore, is crucial. This activity is captured in capability Se3. It receives search results as inputs and generates aggregated results as outputs. *Aggregator* building block offers this capability.

Interface Layer

Interface layer offers the ability to interact with external parties, including human users, software systems, and other IoTSE instances, to receive queries and return search results. This layer offers four capabilities (Table 3.3).

I1. Accept Query This capability captures the interaction of an IoTSE instance with external actors to receive intents of lookup for IoT content. It returns raw queries as outputs. *Query Interface* building block offers this capability.

I2. Process query into correct format Raw queries must be transformed to the format that can be processed by an IoTSE instance. An example would be transforming a string in natural language into a feature vector. This capability receives raw queries as inputs and returns processes queries as output. *Query Processor* building block offers this capability.

I3. Return search results This capability captures the return of relevant IoT content to search engine users. It receives aggregated search results as inputs and interacts with external actors to communicate results as outputs. *Result Interface* building block encapsulates this capability.

I4. Process search results into appropriate format This capability transforms the search results generated by the search layer into the form required by search engine users. A simple example would be filtering IoT content so that only relevant information is returned. This capability receives raw search results as inputs and generates processed results as outputs. *Result Processor* building block offers this capability.

Computing Resource Layer

Computing Resource layer offers the computing, storage, and networking capability necessary for an IoTSE instance to operate. While the reference architecture generally concerns with abstract logical entities instead of implementation and deployment details, we include various types of computing nodes in the IoTSE reference architecture due to the emphasis on physical deployment aspect observed in many existing works on IoTSE. In the reference architecture, we organize computing resources into three building blocks. *Edge nodes* denote low-power computing devices such as wireless sensor nodes and Raspberry Pi devices [5, 76, 77]. *Fog nodes* denote major, elastic computing, and storage capability placed physically close to IoT infrastructures [76]. *Cloud nodes* denote remote, significant, elastic computing and storage capability accessible via the Internet [4].

Integration Layer

Integration layer captures the concerns related to the interoperation of components in an IoTSE instance. Its capabilities include offering communication channels, protocols, and message formats to enable the interaction between IoTSE components. Details on building blocks and patterns of this layer are left to the lower-level instantiation of this reference architecture. For instance, an SOA solution for IoTSE can fill this layer with necessary service buses and message formats to achieve interoperation between IoTSE component services.

Management Layer

Management layer captures the concerns related to monitoring the workflow of an IoTSE instance and performance of its comprising components. It offers monitoring capabilities on the IoTSE instance-wise workflows, as well as performance and quality of services offered by individual components. Details on building blocks and patterns of this layer are left to the lower-level instantiation of this reference architecture.

Security Layer

Security layer captures cross-cutting concerns on securing IoTSE. This layer has four main capabilities. First, it protects an IoTSE instance from external intrusions by users and other IoTSE instances. Second, it assesses the trust and provenance of discovered IoT content. Third, it authorizes access to an IoTSE instance and the contents that it holds. This capability can be realized via query interfaces and processors in the interface layer. Finally, it controls the degree of exposure of IoT content in search results. This capability can be realized via result processors in the interface layer. Details on building blocks and patterns of this layer require further research which is out-of-scope of this work.

3.4.3 Composition Patterns

A composition is a system generated from a collection of elements to serve a specific purpose [78]. Each IoTSE instance is a composition of component instantiated from architectural building blocks in this reported IoTSE reference architecture. A *composition pattern* describes the types of involved components, timing, and sequencing of their interactions to realize a task in the operation of an IoTSE. Composition patterns are named and described

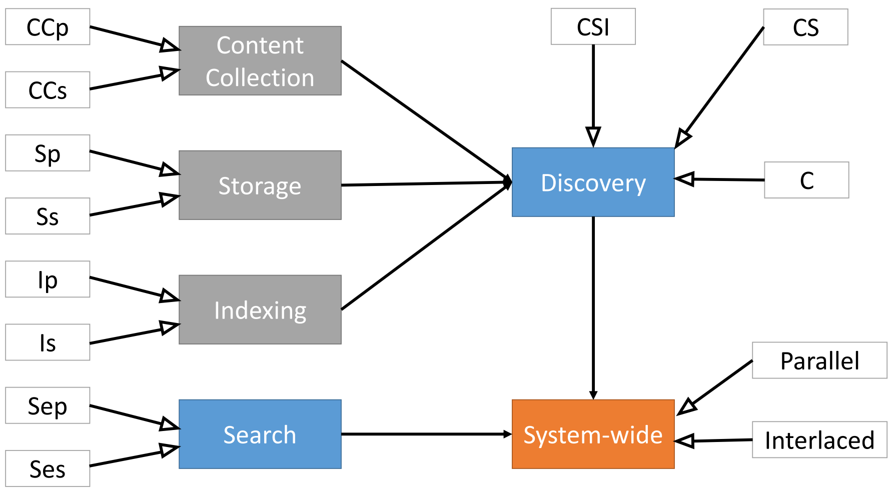


Fig. 3.3 A taxonomy of IoTSE composition patterns.

by their capabilities, which are represented by inputs and outputs. Patterns discussed in this section provide the vocabulary for communicating the logical composition of IoTSE instances and offer a guide for combining ABBs to design a concrete IoTSE instance.

Composition patterns described in this section have been identified from workflows of IoTSE prototypes in the literature. These *system-wide* patterns are decomposed into *search* and *discovery* patterns. The later set is further classified into three subsets: *content collection*, *storage*, and *indexing* (Fig. 3.3). Reaching this level of granularity allows our reference architecture to capture a wide variety of composition structure with the same set of "pattern vocabulary". For instance, our reference architecture can describe both IoTSE instances carrying out every task, as well as the ones omitting storage and index generation. It can also describe variations in timing between components involved in a certain task.

Our description of patterns assumes that an ABB can be implemented as functionally equivalent software components that process different types of IoT content, and these components are utilized together in an IoTSE instance to resolve queries. For instance, Q.D. ranker ABB can be realized as "Q.D. ranker for metadata" and "Q.D. ranker for streaming sensing data" components, which are utilized together in an IoTSE instance. In the following

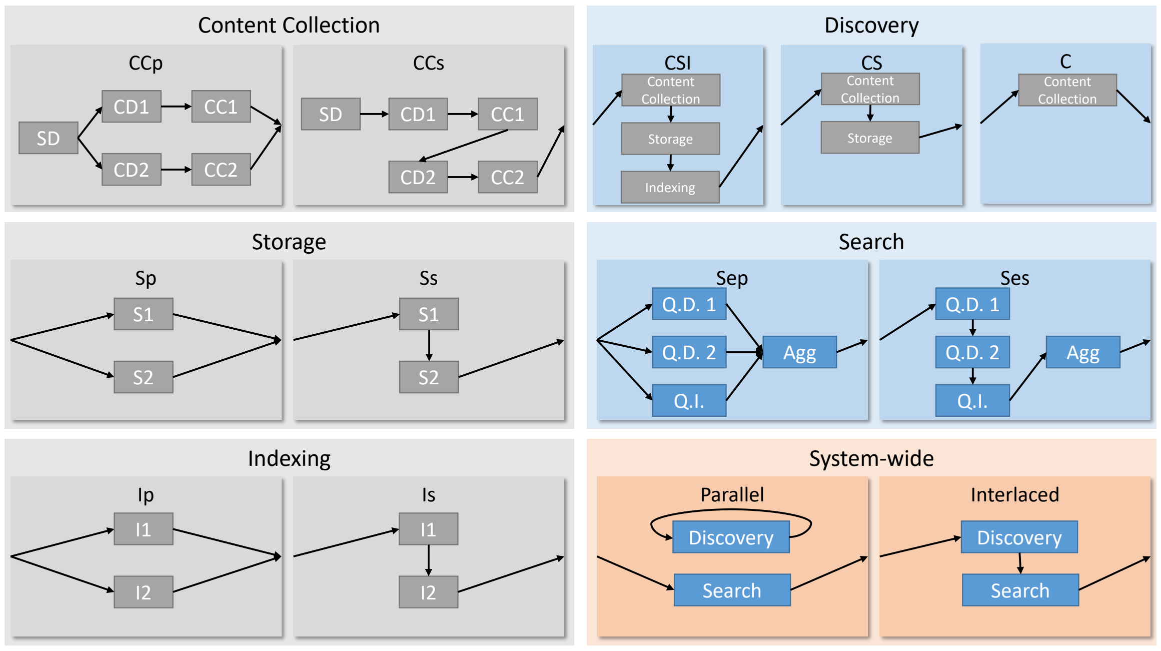


Fig. 3.4 Detail of IoTSE composition patterns. Abbreviations used: SD - source detector, CD - content detector, CC - content collector, S - storage, I - indexer, Q.D. - query dependent ranker, Q.I. - query independent ranker, Agg - aggregator.

description, we depict the parallel patterns in the context of only two types of IoT content for clarity. The IoTSE components with the index 1, such as Content Detector 1, process the first type of IoT content, while the components with the index 2 process the other type. The depicted patterns, however, can cover an arbitrary number of IoT content type.

It should also be noted that the patterns described in this section concern only with the parallelization of components on a logical level. They do not capture, for instance, duplication and parallelization of a component in runtime to achieve scalability.

Content Collection Patterns

The patterns in this group(*CC*) concern with detecting and retrieving IoT contents necessary for resolving queries. These patterns involve source detectors, content detectors, and content collectors. They receive instructions for detecting IoT content and return collected content as outputs. Two patterns in this group differ by the parallelization of content detection and collection.

Storage Patterns

This group *(S)* contains patterns for storing collected IoT content. These patterns involve storage building blocks. They receive IoT content as inputs and are not required to generate outputs. Two patterns in this group differ the parallelization of storage activities for different types of IoT contents.

Indexing Patterns

The patterns in this group *(I)* concern with generating and storing indexes for different types of IoT contents. These patterns involve indexer and index building blocks. They receive IoT content as inputs and return indexes as outputs. Two patterns in this group differ in the timing between index generation for different types of IoT content.

Discovery Patterns

The patterns in this group are generated by combining patterns from the three groups mentioned above. They aim at generating collections of IoT contents to support query assessment. This group has three patterns: *CSI* pattern involves content collection, storage, and indexing; *CS* pattern omits index generation; and *C* pattern further omits the storage of collected IoT contents. CS and C patterns are generally utilized by IoTSE instances which avoid storing massive data generated by IoT infrastructure.

Search Patterns

This group *(Se)* contains patterns for resolving queries. They involve Q.D. ranker, Q.I. ranker, and aggregator building blocks. The patterns in this group receive queries, IoT contents and, optionally, indexes as inputs, and generate search results as outputs. The patterns in this groups differ by the timing of ranking components.

System-wide Patterns

Two patterns in this group capture the macro composition of an IoTSE instance, specifically the timing between discovery and search processes. In the *parallel* pattern, content discovery is carried out continuously and independently from query assessment. This pattern is common among search systems not concerning with computing and storage resources. The *interlaced* pattern, on the other hand, are utilized by IoTSE instances facing constraints on resources. In this pattern, content discovery is only carried out when a query is received.

3.4.4 Deployment Patterns

A *deployment pattern* describes the deployment structure of an IoTSE instance, specifically *(i)* types and arrangement of utilized building blocks in the computing resource layer, and *(ii)* deployment of ABB on computing resource blocks. We decided to capture deployment patterns in the IoTSE reference architecture due to two observations. First, some work on IoTSE is driven by the deployment structure, specifically to bring IoTSE instances to the edge and remove centralized control on such systems. Second, the deployment of an IoTSE instance plays a crucial role not only in its performance but also in whether or not it can operate. For instance, due to the sensitivity of data collected by IoT infrastructure, the search engine might only be deployed in the vicinity of the infrastructure itself. These observations highlight the role of deployment structures and emphasize the need for the reference architecture to capture this information. The patterns described in this section provide a vocabulary for communicating the deployment aspect of an IoTSE instance. These patterns also guide the deployment of concrete IoTSE solutions.

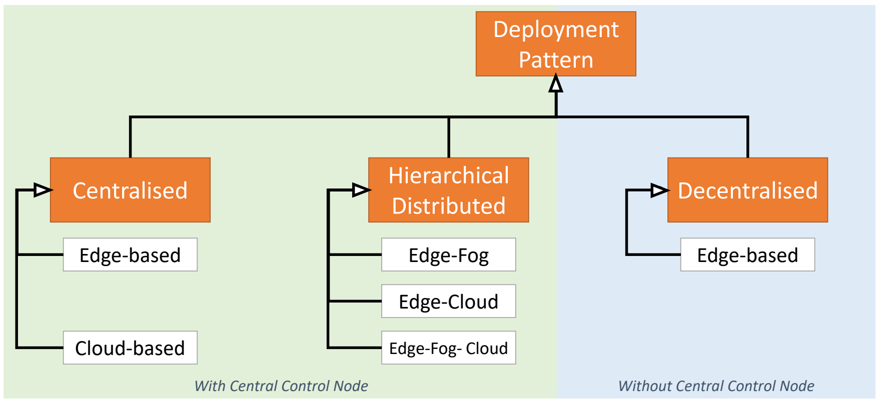


Fig. 3.5 A taxonomy of IoTSE deployment patterns.

Patterns in this section have been identified from IoTSE prototypes in the literature. Depending on the existence of centralised control and location of utilised computing nodes (i.e., edge, fog, and cloud), we organise these patterns into three classes: *centralised*, *hierarchical distributed*, and *decentralised* (Fig. 3.5). We limit the discussion to only patterns that we have detected in the IoTSE literature. We also limit the discussion on the arrangement of ABB on computing nodes only on a layer basis, focusing on discovery, storage, and search layer. We recognize that, in practice, components of the same layer can be deployed across multiple nodes, and an IoTSE instance can employ different patterns for each involving content type. It is not our aim to identify an exhaustive list of patterns.

Centralized Patterns

Centralized patterns capture IoTSE instances that have a central control node and limit their deployment to one location (i.e., only edge, fog, or cloud). In these patterns, all three layers are deployed on the same computing resource node. In the edge-based pattern, an IoTSE instance is deployed entirely on one edge node [46]. In the cloud-based pattern, an IoTSE instance is deployed on a remote cloud node. It should be noted that even if the cloud is made available by multiple distributed computing nodes, if it is available as a single (logical) node, then we still consider the deployment centralized.

Hierarchical Distributed Patterns

Hierarchical distributed patterns describe IoTSE instances which have a central control node and are distributed across multiple computing nodes, forming a hierarchy. In the *Edge-fog* pattern, the control node resides on a computing node with strong capability in the vicinity of the IoT infrastructure [35, 46]. In the *edge-cloud* pattern, the control node resides on a remote cloud, generally handling storage, indexing, and query assessment, while edge nodes are generally in charge of detecting and collecting IoT content [17]. In the *edge-fog-cloud* pattern, the control node resides on a remote cloud and generally handles global query assessments, while remaining components are distributed across edge and fog nodes [21, 14, 19].

Decentralized Patterns

Decentralized patterns describe IoTSE instances that do not have a central control node. In the literature, we observe mostly *edge-based* decentralized systems. Each node in a decentralized IoTSE instance possesses all components necessary to resolve queries independently, and collaborates with other nodes in a peer-to-peer fashion to increase its coverage.

3.5 A Framework for IoTSE Instantiation

Instantiation denotes a series of transformation steps to turn a reference architecture into a single or a family of software systems [74]. In this section, we introduce an instantiation framework, which supports both generating an IoTSE instance and an ecosystem that support engineering multiple IoTSE instances. This framework consists of four phases (Figure 3.6). Each phase represents a transformation which brings abstract building blocks and patterns of the reference architecture closer to the implementation.

Phase 1: The first phase transforms the IoTSE reference architecture into more concrete architectures that are closer to implementation. The key design decision in this phase is on the

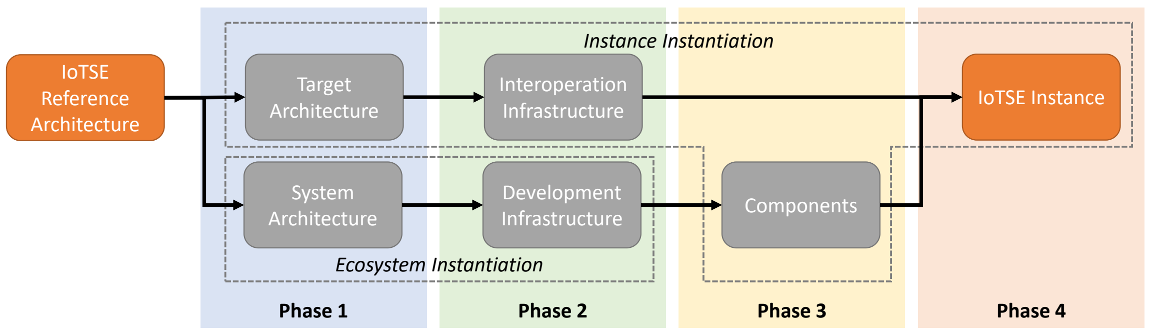


Fig. 3.6 Four phases of the instantiation framework.

type of software component (e.g., software library, module, service) that encapsulates architectural building blocks of IoTSE. Following this decision, the IoTSE reference architecture can be mapped onto another reference architecture that supports the chosen type of software component. This mapping activity helps identify and incorporate the necessary building blocks to support the interoperation of components, the management, and the operation of the whole system. For instance, if we decide to design and implement building blocks of IoTSE as software services, we would map the IoTSE reference architecture to a Service-oriented Architecture (SOA) reference architecture (e.g., S3 [73] or ISO/IEC 18384 [72]).

This first phase creates either a *target architecture* that is specific to the IoTSE instance being instantiated, or a *system architecture* which is more general than the target architecture and applicable to multiple IoTSE instances.

Phase 2: The second phase instantiates the software infrastructure necessary for running IoTSE components and controlling their interactions. For instance, assuming that in the previous phase, we decided to engineer the IoTSE instance as a set of Web services, each of which implements an IoTSE component. Then, in this phase, the software infrastructure will consist of Web servers running on the involved computing nodes to serve the Web services, a service registry, and a service orchestration engine to control the interaction between services.

If the goal of this phase is instantiating only an IoTSE instance, we can conclude it here. However, if we aim to engineer multiple IoTSE instances, then we also need to instantiate an additional infrastructure that supports the development of IoTSE components according to the system architecture that we instantiated in the previous phase. Continuing our example, the development infrastructure might include software libraries to simplify the development of Web services that encapsulate IoTSE components. It might also include repositories to accumulate these component services for future engineering of IoTSE instances by external parties. Together with the system architecture in the first phase, the development infrastructure forms an ecosystem for engineering IoTSE instances.

Phase 3: The third phase involves building algorithms and mechanisms corresponding to different IoTSE building blocks, based on the capabilities and scope specified in the reference architecture, and encapsulating them into components to be used by the infrastructure built in the second phase.

Phase 4: The final phase involves deploying and composing IoTSE components into operational IoTSE instances. The first step in this phase is selecting the components that are necessary for resolving the targeted type of query from the components developed during the previous phase. If the goal of the instantiation is engineering multiple IoTSE, then this step is relevant as the third phase might have produced more IoTSE components than necessary. If the goal of the instantiation is a single IoTSE instance, this step might be skipped. When an extensive set of components is available, this step is not trivial and can be benefited by search tools (e.g., [79]).

The second step involves constructing composition patterns for the IoTSE instance from sub-patterns defined in the reference architecture. The third step is selecting a deployment pattern. Finally, the infrastructure constructed in the second phase is utilized to realize these patterns and generate complete IoTSE instances.

3.6 Evaluation

A reference architecture is “good” if it is accessible and understandable by a majority of the involved parties, possesses satisfactory qualities, is up-to-date and maintainable, and adds value to the business [74]. The “goodness” of a reference architecture is generally assessed based on feedback from the users (i.e. architects and developers). This form of assessment – sometimes referred to as validation [80] – presents a paradox: a reference architecture must be “good” to be published, utilized, and receive feedbacks; however, to be published, utilized, and receive feedbacks, the reference architecture must first be assessed as “good”.

To avoid the presented loop, we instead assess the “goodness” of the IoTSE reference architecture on the ability of its building blocks and patterns to model IoTSE instances. In this section, we present a mapping of the reference architecture onto two representative IoTSE prototypes as an assessment. In the following chapters, we will conduct case studies, which involve engineering IoTSE instances based on the reference architecture, as further assessment. This approach allows us to assess the reference architecture when the external feedback is not available. The conducted mappings and case studies can also increase the confidence of architects and developers to adopt the reference architecture and provide feedback in future evaluation when the reference architecture has been released.

3.6.1 Methods

Assessment of the IoTSE reference architecture was carried out by mapping architectures of IoTSE prototypes in the literature to its building blocks and patterns. Each mapping represents a test. The reference architecture passes the test if it can satisfy the following criteria. First, the reference architecture can map the conceptual components of the prototype into its building blocks. Second, the reference architecture can model the operation of the prototype, represented by interactions between its components, with its composition patterns

Finally, the reference architecture can model the deployment structure of the prototype with its deployment patterns.

We selected a sample of two prototypes from over 200 existing work based on following criteria: (i) they must be representative, meaning their approaches and architectures must appear in multiple studies; and (ii) they should utilize complex composition and deployment architecture. By applying these two criteria, we selected a sample comprising Dyser [21] and IoT-SVK [17].

3.6.2 Results

This section presents mappings of IoTSE prototypes to the reference architecture. For each prototype, we present an overview of its features, the mapping from its functional components and hardware nodes to architectural building blocks, and the mapping from its workflows to the composition patterns specified in the reference architecture.

Mapping to IoT-SVK: IoT-SVK [17] is a multi-modal, real-time IoTSE. It resolves queries on IoT-enabled sensors based on their keyword descriptions, locations, trajectories, and sensing values. IoT-SVK collects sensor readings and relies on an efficient index update to enable near real-time querying. Notably, IoT-SVK lacks sensor discovery capability. Instead, it relies on pre-configured sensor networks.

IoT-SVK utilises 15 types of functional components, which are instantiated from 7 of 18 defined architectural building blocks. These components comprises three storages *(S)*, three indexers *(I)*, three indexes *(Idx)*, and three query dependent rankers *(Q.D)* corresponding to three types of content utilised in query assessment (i.e., keyword description *(K)*, spatio-temporal data *(S)*, and sensing value *(V)*); one aggregator *(Agg)*, one query interface *(Q.In)* and one result interface *(R.In)*. These components are distributed across a hierarchy of fog and cloud nodes, following the *hierarchical distributed* deployment pattern specified in the reference architecture.

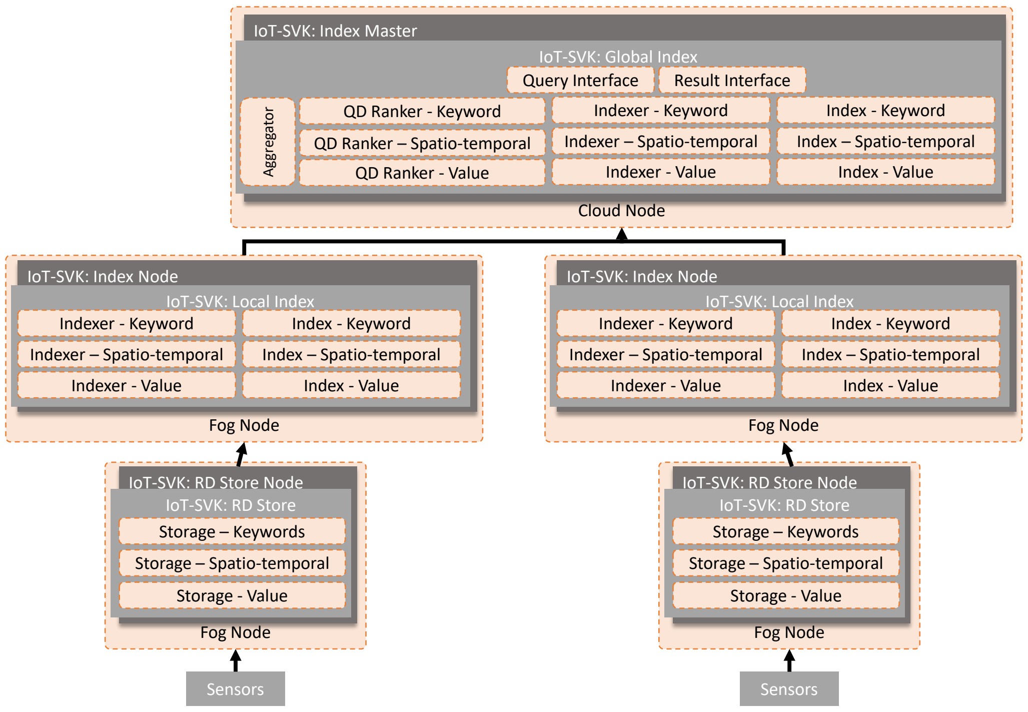


Fig. 3.7 Mapping of IoT-SVK components and deployment structure to the reference architecture. Blocks with dashed border are component instantiated from building blocks in the reference architecture.

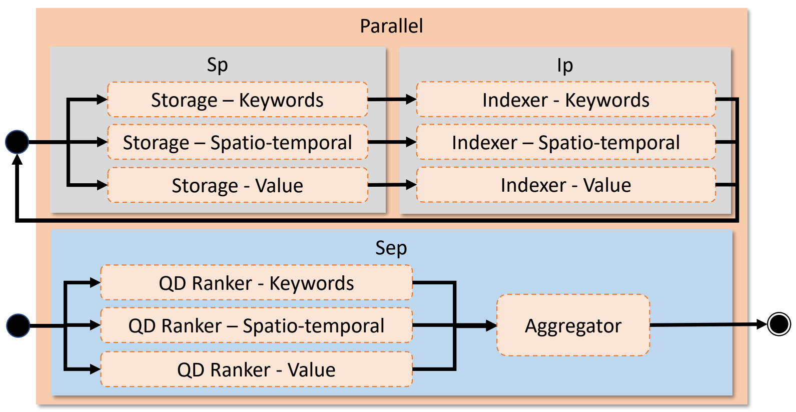


Fig. 3.8 Mapping of IoT-SVK workflows into interaction patterns defined in the reference architecture.

Figure 3.8 depicts the mapping of workflow by IoT-SVK to interaction patterns described in the reference architecture. On the system-wide level, this prototype utilises *parallel* pattern. Discovery-wise, IoT-SVK is mapped to pattern *SI* as it includes only storage and indexing activity. Search-wise, IoT-SVK is mapped to pattern *Sep*. In each activity, IoT-SVK processes three types of content (i.e., S, V, K) in parallel.

Mapping to Dyser: Dyser [21] is also a real-time IoTSE. It searches for physical entities whose real-world states, reflected by their embedded sensors, match queries given by search clients. Different from IoT-SVK, Dyser does not store and index sensor readings. Instead, it contacts and validates the values of sensors during the query assessment. Prediction models are utilized to prioritize the time-consuming sensor communication.

The scope of Dyser is a challenging design decision that we had to make early in the mapping. Specifically, we had to consider whether model construction is a component of Dyser. Sensor content prediction models and model construction have a central role in Dyser, and related prototypes (e.g., [13–15]), and therefore should be included as a part of the search engine. However, the architecture described in these works does not include model construction. Instead, this component locates in remote IoT gateway systems and is maintained by sensor owners instead of search engine operators. In this mapping, we respect the scoping decision presented in the original study and exclude model construction from the search engine.

Dyser utilizes 19 types of functional components, which are instantiated from 9 of 18 defined architectural building blocks. These components comprises three content detectors *(CD)*, three content collectors *(CC)*, three indexers *(I)*, three indexes *(Idx)*, and three query dependent rankers *(Q.D)* corresponding to three types of content utilised in query assessment (i.e., entity page *(E)*, sensor page *(S)*, and prediction model *(P)*); one source detector *(SD)*, one aggregator *(Agg)*, one query interface *(Q.In)* and one result interface *(R.In)*. The query-dependent ranker for prediction model (Q.D-P) encapsulates both sensor ranking based on prediction models and the validation of real-time sensor readings. These components are deployed on a single remote computing node, following the *cloud-based centralised* deployment pattern specified in the reference architecture. Figure 3.9 depicts the description of mapping between Dyser's components and building blocks in the reference architecture. Note that the reference architecture has finer granularity, and uses different terminologies (e.g., Dyser uses "indexer" to denote content discovery).

Figure 3.10 depicts the mapping of the workflow by Dyser to the composition patterns described in the reference architecture. On the system level, this prototype utilizes *parallel* pattern. Discovery-wise, Dyser is mapped to pattern *CI* as it includes only content collection and indexing activity. In each activity, Dyser processes three types of content in parallel. Search-wise, IoT-SVK is mapped to pattern *Ses*. Sequencing of ranking reflects the arrangement of query assessment into a filtering pipeline utilized by Dyser.

3.6.3 Threats to the Validity

Threats to internal validity The first threat to the internal validity of our assessment is the selection of IoTSE prototypes to which the reference architecture is mapped. Specifically, as chosen IoTSE prototypes have been used as inputs to the reference architecture, one

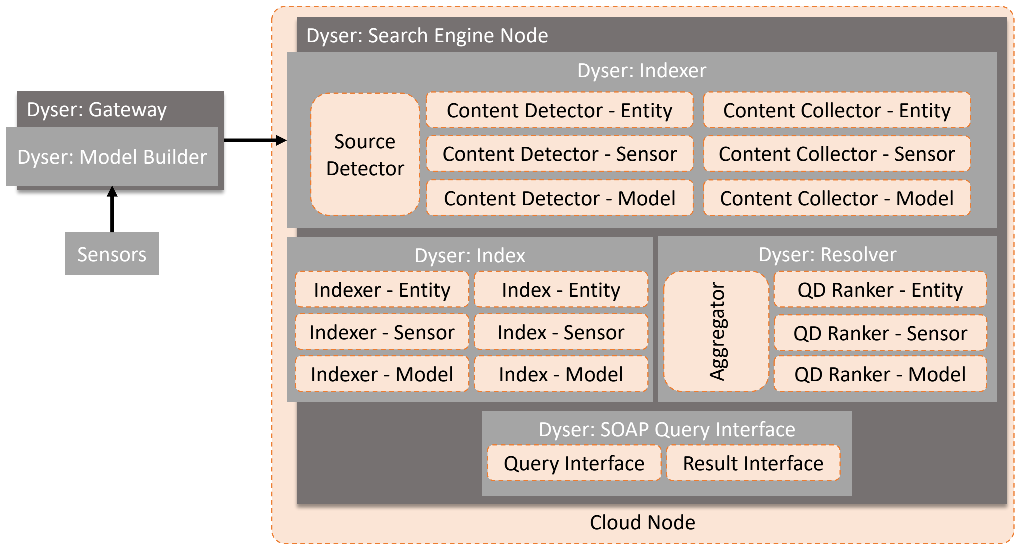


Fig. 3.9 Mapping of Dyser components and deployment structure to the reference architecture. Blocks with dashed border are component instantiated from building blocks in the reference architecture.

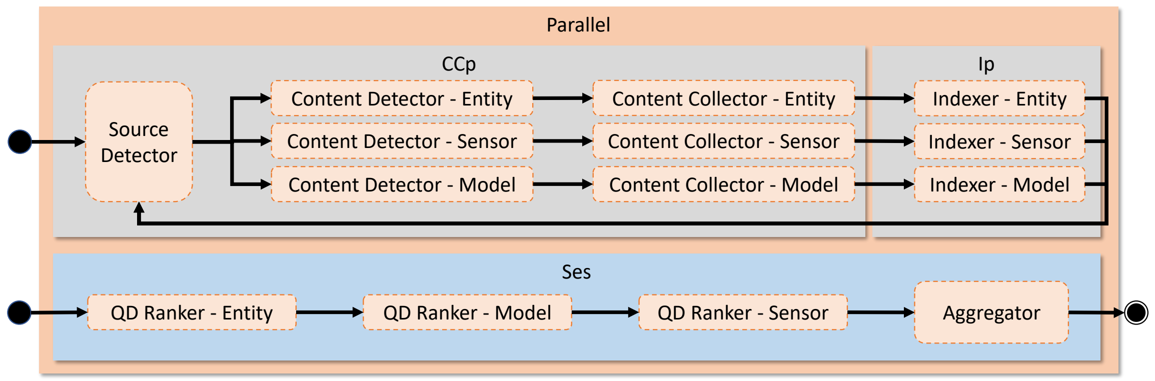


Fig. 3.10 Mapping of Dyser workflows into interaction patterns defined in the reference architecture.

would argue that the reference architecture is capable of modeling these prototypes by design. Therefore, the assessment is not valid. We argue that this is not the case, as the IoTSE reference architecture emerges from the inputs of many prototypes besides our Dyser and IoT-SVK in an iterative design process. Due to multiple transformation and aggregation in this process, the resulting architecture might no longer match Dyser and IoT-SVK. Therefore, mapping the IoTSE reference architecture back to these works is valid to verify whether the architecture was built right.

The second threat to the internal validity of our assessment is the correctness of our mapping from Dyser and IoT-SVK to the reference architecture. To negate this threat, we derive the architecture of chosen prototypes from their textual descriptions, architectural models, and described experimentations; and did our best to ensure faithfulness of the mapping from these architectures to the IoTSE reference architecture.

Threats to external validity A threat to the generalisability of our assessment is size and representativeness of chosen IoTSE prototypes for testing the reference architecture. Regarding the sample size, we emphasize that the architecture patterns of Dyser and IoT-SVK are common among IoTSE prototypes. Therefore, our assessment covers not only two systems but, arguably, two *classes* of IoTSE prototypes. Regarding the representativeness of our sample, we emphasize that the sampling has been carried out on a representative subset of IoTSE works, comprising both highly referenced and latest studies. Finally, we emphasize that the architectural patterns utilized by chosen IoTSE prototypes are, arguably, among the most complex ones in IoTSE literature. Therefore, they are valid to test the modeling capability of the reference architecture.

3.7 Related Works

Our work falls at the intersection of the Internet of Things Search Engine and the Software Architecture of Search Engine systems. As we have conducted an extensive survey on IoTSE literature in Chapter 2, in this section, we focus on comparing and contrasting the proposed IoTSE reference architecture with other efforts on developing an architecture for search engine systems.

As literature on architecture of IoTSE is limited, we extend the scope of our related work to include studies on architecture of non-IoT search engines (e.g., content-based image search engine [81], distributed document search engine [82, 83], semantic web search engine [84], academic document search engine [85], spatial-textual search engine [86], social search engine [87], and Web search engine [88]). We present an overview of these related works in chronological order and discuss the position of our work with respect to these works.

One of the earliest works on search engine architecture that we found is Virage – an open framework for content-based image search engines [81]. In these systems, images are transformed from data-rich matrices of pixels to semantic-rich primitives, and query assessment is carried out on these primitives. Virage is a library of algorithms for processing images to primitives and resolving queries on these primitives. External parties can offer different algorithms for building primitives from images and ranking based on the provided primitives as plugins.

AgentRAIDER [82] presents the architecture of an agent-based search engine that helps users finding information hosted in distributed heterogeneous data sources. This architecture comprises five agents, each of which handles a specific task in the query processing, and a database holding user profiles for customizing queries and filtering information. This architecture is instantiated from a reference architecture called I3, developed by ARPA and Stanford.

P2P-IR architecture [83] is a layered “generic” architecture for peer-to-peer information retrieval (P2P-IR) system. It focuses on enabling separation of concern in a P2P-IR system into abstract layers. This architecture comprises four layers – transport, structured overlay network, document and content management, retrieval models. Each layer concerns with a specific type of object, and support a specific set of operations on this object.

Swoogle [84] presents the architecture and implementation of a semantic web search engine. This work defines a semantic web search engine as a three layer system. It crawls semantic web document (i.e., written in RDF) from the search results of Google, and resolves queries on these documents. It is capable of querying for ontology, for a semantic web document, and answering questions on the structure of a semantic web search engine.

CiteSeerX architecture [85] describes the new architecture of CiteSeer – an academic document search system. The monolithic architecture of CiteSeer faced the limitation on scalability to process a large number of queries and documents. This monolithic architecture also limits the ability to add and modify the functionality of the system. Therefore, CiteSeer was redesigned into CiteSeerX. The new architecture is a service-oriented system, in which majority of its functional components are redesigned as autonomous Web services. This architecture allows independent scalability and ease of modification.

STEWARD [86] presents the architecture of a spatial-textual search engine. The presented system is capable of tagging spatial location to Web documents and resolving queries on them based on both their location and textual content. Tagging is done by a combination of natural language processing and external gazetteers for location-information look up. This system is structured according to seven phases of operation that it carries out. The motivation of this decision is the distribution for scalability. The authors claim that each phase can be mapped onto a different computing node.

Horowitz et al. [87] present the architecture of a large scale social search engine. This type of system resolves a query, not for documents but human in the extended social network of the query client. It finds a human who can resolve the query for information specified by the query client, and facilitate the communication between them. The reported architecture comprises the following components: importer, database, index, routing engine, question analyzer, and conversation manager. The importer is responsible for importing social network of search clients who register to join the system, identify and index topics that that user discuss based on public posts. Question analyzer and routing engine are responsible for finding a human in the social network that might be able to answer the query. Conversation manager facilitates the message exchange between the query client and potential human users that have the answer.

Brin, et al. [88] present the architecture of Google, such that it supports large-scale collection and storage of HTML documents, extraction of links between documents, maintains various information about documents, link structure, and anchor texts, and resolves queries on these documents in a highly scalable way. The architecture presented in this work is closely related to its utilized retrieval models.

Our work is different from these works in many ways. First, our work has a different purpose and is constructed differently. Majority of the reviewed work, except Virage [81] and P2P-IR [83], focuses on explaining the inner architecture of a specific search engine prototype, instead of offering an architectural foundation to instantiate new systems. These architectures can be closely tied up to the specific algorithms and mechanism that they use, and therefore might not be generalized. Second, the listed works discuss only the building blocks of their system. Our reference architecture, on the other hand, discusses not the only building blocks and their capabilities, but also patterns of their interaction and deployment, and a framework for using the reference architecture to instantiate new systems. Third, the listed works assume a specific interaction pattern between building blocks. Our work, on the other hand, provides a taxonomy of combinable patterns to describe and allow instantiating different IoTSE instances. Finally, the listed work either do not address or assume only one pattern of deployment. Our work, on the other hand, provides a taxonomy of deployment pattern to describe different situation faced by IoTSE systems.

3.8 Summary

The overlap among IoTSE instances of different classes regarding their internal operation and architecture suggests a potential for leveraging reuse in the engineering of these systems. In this chapter, we have identified the commonalities of IoTSE from an analysis on representative IoTSE prototypes, and compiled this knowledge into a reference architecture to guide the reuse-centric engineering of IoTSE. The reference architecture specifies 18 types of components, 13 composition patterns, and 6 deployment patterns, which together can model IoTSE instances of various classes. This reference architecture offers the necessary blueprint for developing IoTSE components and engineering IoTSE instances.

Future research enabled by the reported reference architecture can be organised into three directions:

Instantiating a close-to-implementation architecture template for IoTSE. The reference architecture describes abstract building blocks and patterns of their interactions in IoTSE solutions. More concrete architecture templates simplifies the instantiation of concrete IoTSE solutions from the reference architecture. These concrete architecture templates concern with *(i)* the type of software components (e.g., services, plug-ins, libraries) into which ABBs are mapped, and *(ii)* supporting software components to enable the interoperation and composition of ABBs into an IoTSE solution. Architecture templates can be instantiated by applying an architectural style, such as Service-oriented Architecture, to the reference architecture.

Bringing architectural-level security support to IoTSE. Security protects an IoTSE solution against destruction, corruption, removal, disclosure of its data, and interruption of its service. It encompasses privacy and trust concerns. Security has been established as crucial to the success of an IoTSE solution because this system holds sensitive information about people and businesses, and it serves other IoT software systems, and therefore its compromise has a ripple effect across the whole IoT ecosystem. Existing efforts on securing IoTSE have been limited to policies, algorithms and encryption mechanisms. The reference architecture allows addressing security at a system level by identifying the security-related building blocks and the patterns, which can be shared and integrated into IoTSE instances.

Accumulating architecture knowledge resulting from applying the reference architecture. IoTSE architects and operators make various design decisions on the types of components and patterns to be used for a specific type of query and operation context of an IoTSE instance. The body of knowledge resulting from the accumulation of these decisions represents a "recipe book" for engineering future IoTSE instances. It provides a source of data for mining new insights on how IoTSE solutions are composed and deployed in practice. It will also enable machines to learn and resemble the design decisions of IoTSE architects and operators, which eventually might lead to the automatic engineering of these systems.

Chapter 4

A Kernel-based Approach to IoTSE Engineering

In the previous chapter, we have proposed a reference architecture, which offers a blueprint for developing IoTSE components and engineering IoTSE instances. However, the reference architecture by itself is inadequate for enabling the reuse-centric IoTSE engineering. Component developers need a software infrastructure that helps them transform their algorithms and mechanisms into reusable, composable IoTSE components. Search engine operators need a software infrastructure that helps them engineer IoTSE instances by accumulating IoTSE components, controlling their interactions, and deploying them on given computing infrastructure. This chapter proposes a kernel-based approach to IoTSE engineering as a solution to the stated problem. It addresses the third research objective of the thesis: to propose a software infrastructure to support the reuse-centric engineering of IoTSE.

4.1 Introduction

The IoTSE reference architecture is the first step toward the reuse-centric IoTSE engineering. By committing to the same reference architecture, independent component developers can engineer IoTSE components that are compatible with each other and fit into the bigger picture of an IoTSE engineering effort. Search engine engineers and operators can use the patterns in the reference architecture to communicate and assess the design of an IoTSE instance and engineer it with components that have been contributed by independent component developers. The reference architecture provides the blueprint to enable this vision of reuse-centric IoTSE engineering.

However, the reference architecture by itself is inadequate. As we have pointed out in the framework for IoTSE instantiation (Section 3.5), regardless of whether our goal is engineering one IoTSE instance or multiple instances, software infrastructure is necessary. This infrastructure runs software components that encapsulate the functional elements of an IoTSE instance and controls their interactions so that they can work together to discover and resolve queries on IoT content. If our goal is engineering multiple IoTSE instances, then a software infrastructure to support the development and accumulation of IoTSE components is also necessary. Such infrastructure might encourage the engineering of reusable, composable IoTSE components by reducing the overhead associated with transforming algorithms and mechanisms related to IoTSE operations to IoTSE components.

In this chapter, we propose a *Kernel-based Approach to IoTSE Engineering* to enable a supporting software infrastructure for IoTSE engineering. This approach was inspired by the design of modern operating systems. In these systems, a majority of their functionality and operations are developed as plug-able modules around a microkernel that provides essential utilities. Similarly, algorithms and mechanisms for realizing different internal operations of an IoTSE instance, such as detecting IoT data sources and generating indexes, can be engineered as plug-able modules around an IoTSE kernel that offers them a runtime environment and control their interaction. Moreover, this kernel might also provide templates for developing IoTSE components that are compliant with the reference architecture. By utilizing these templates, a component developer can transform their algorithms and mechanisms into reusable, composable IoTSE components with minimal effort. These modules, then, can be shared and acquired via package managers, similarly to APT of the Ubuntu OS and PIP of Python.

In order to facilitate the development and evaluation of the proposed IoTSE kernel, and also to clarify the discussions in this chapter, we limit the scope of the reported study to a subclass of IoTSE, which we will denote as the Web Sensor Retrieval System (WSR). Instances of WSR discover and resolve queries on metadata and sensing streams to retrieve IoT-enabled sensors, which are accessible from the Web. While this scoping might impact the generalizability of the proposed approach, it has three favorable trade-offs:

First, the type of query that WSR instances resolve belongs to the meta-path $[D+R => E => R]$, which is the second-most popular type of meta-path and much more complex compared to the other types detected in the literature. Therefore, by focusing on WSR, we can bring the specificity to the discussion, which has been revolving around generic IoTSE instances, without giving up a significant amount of generalizability. Second, as the IoT has been increasingly accessible from the Web, focusing on *Web Sensor* Retrieval System is timely. Finally, the evaluation would be more straightforward as sensory datasets are more accessible compared to other types of IoT-related datasets, such as the ones on actuation services.

In the following sections, we will provide more background information of WSR (Section 4.2) and then present details of the kernel-based approach (Section 4.3 and 4.4). Section 4.5 presents an evaluation of the proposed kernel via a case study, which involves engineering an IoTSE prototype and assessing the reduction regarding the development effort. In Section 4.7, we discuss the strengths and weaknesses of the kernel-based approach and potential future works to improve it, which partially inspired the platform-based approach proposed in the following chapter.

4.2 Background

4.2.1 Web Sensor Retrieval System

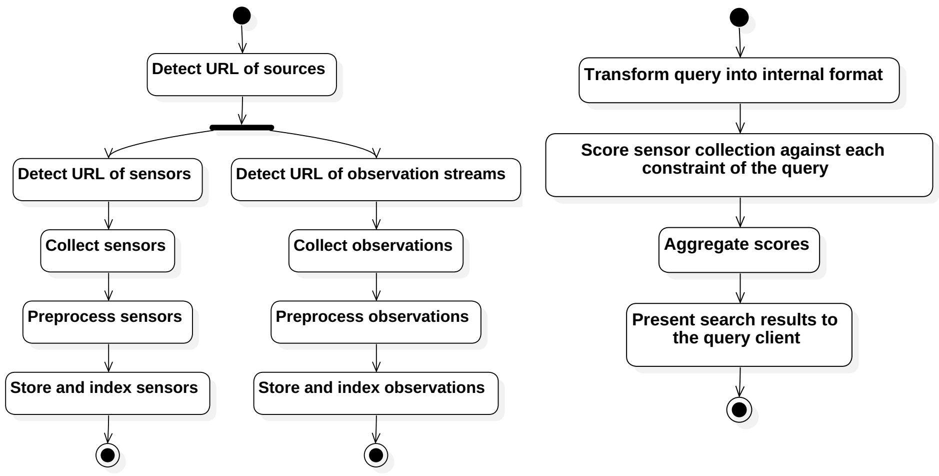
Software systems require the ability to retrieve needed sensors and actuators from the Web in order to adapt themselves to different sensing and actuating infrastructure with minimal users' intervention. For instance, to list all empty meeting rooms near a user, the building management system requires access to passive infrared sensors that detect motion in meeting rooms to deduce their availability. While sensors can be pre-configured, the building management system is more flexible with the ability to detect and retrieve sensors in run-time. We define *Web Sensor Retrieval system (WSR)* as a system providing the ability to discover and resolve queries on Web-enabled sensors (i.e., *Web Sensors*). In a smart building, instances of WSR can be deployed on each floor to manage its sensors, or on a remote cloud service to manage all Web sensors in the building. WSR instances can also be integrated into the building management systems to interact with Web sensors in the vicinity of users.

As WSR is a type of IoTSE, it faces similar problems including the scalability in both direction, the *diversity of sensors* and the *diversity of queries imposing on them.* In a smart building, different types of sensors such as temperature, humidity, luminance, power consumption, noise and image are available. They have different forms of descriptions and different quality metrics, such as refresh rate and power consumption. They produce observations with different data types and formats. Even if they produce observations with the same syntax, the real world feature that they observe might be different. For instance, the temperature in a meeting room has different meaning and behavior than the operational temperature of an HVAC system. Queries on these sensors are also diverse. They vary from simple ID look-up to combinations of textual description, spatial location and numeric value of sensors [17] (e.g., finding all available parking spots that are less than 500 meters away from the campus). Queries can also be imposed on features and characteristics of observation streams produced by sensors, such as finding power consumption sensors that show abnormalities in the last 24 hours. Each type of sensor and sensor query requires different techniques to detect, index and query. Despite this diversity, a functional WSR instance must be able to retrieve any Web sensors needed by client applications.

While the diversity challenge can be addressed by equipping WSR instances with every available mechanisms and algorithms to work with any sensor and query, this solution is not appropriate for WSR instances deployed in resource-constrained scenarios (e.g., [35, 38]). Similarly to IoTSE, reuse-centric engineering is also a potential solution for WSR. By accumulating an extensive selection of components for discovering and querying different types of sensors, and then compose them selectively into WSR instances, these systems can be adapted to different usage scenarios without over-engineering. For example, a WSR instance that queries temperature-humidity sensors (e.g., DHT22) on seasonal feature of their observations can be composed from modules that detect, parse, index DHT22 sensors and resolve queries on seasonal feature of time series. Extending the WSR instance to support other types of sensors and queries can be done by changing its modules.

4.2.2 Discovery

Discovery and Search are two primary processes in a WSR instance. *Discovery* is the process of *detecting* sensors and, optionally, *collecting* them into *sensor collections* (Fig. 4.1a). This process varies by its scope, trigger, and collection activity. The scope of discovery process either extends across the globe via the Internet (i.e., *global scope*) [62, 21] or limits to the wireless communication range of the device hosting the IoTSE instance (i.e., *local scope*) [38, 35]. Global discovery in WoT can be considered WoT crawling [62]. The discovery


Detect URL of sources
Detect URL of sensors
Detect URL of observation streams
Collect sensors
Collect observations
Preprocess sensors
Preprocess observations
Store and index sensors
Store and index observations
Transform query into internal format
Score sensor collection against each constraint of the query
Aggregate scores
Present search results to the query client

(a) Discovery (b) Search

Fig. 4.1 Discovery and search activities of IoTSE.

process can be executed continuously as a background task, or triggered by a user's request. This feature affects the implementation of sensor collection activity. Discovery processes implemented as background tasks tend to collect and store the information of detected sensors. On the other hand, processes triggered by users tend to return the detected sensor information to users without storing.

For small sensing infrastructure, discovery process alone is sufficient. WoT applications can filter the small list of detected resources by themselves. However, larger sensing infrastructure and more complex queries, such as finding "power consumption sensors on the forth floor of the campus building that recorded irregularities in the last 24 hours", require more sophisticated query mechanisms.

4.2.3 Search

The *Search* process resolves queries on discovered sensors (Fig. 4.1b). Let s be a sensor, and S be the collection of discovered sensors. For each sensor query q, a set of sensors $r_q \subseteq S$ that are relevant to the query exists. The task of search process is constructing the result set $\widetilde{r_q} \subseteq S$ that approximates the unknown r_q. This task is done by evaluating the relevance of each discovered sensor s against the given query q with a relevance function $f(s,q)$. If the relevance function produces binary result, the result set comprises of sensors scored positive and the process is considered *"sensor selection"* or "lookup" (Eq. 4.1). In case the relevance function produces a real number, the result set contains sensors whose score higher than a predefined threshold α (Eq. 4.2). This process is called *"sensor scoring"*.

$$\text{Selection: } \widetilde{r_q} = \{s \in S | f(s,q) = 1\} \text{ where } f(s,q) \in (0,1) \tag{4.1}$$

$$\text{Scoring: } \widetilde{r_q} = \{s \in S | f(s,q) \geq \alpha\} \text{ where } f(s,q) \in \Re \tag{4.2}$$

While the sensor search process shares the formalization with text information retrieval, it is different in three ways. First, sensor search relies on rich metadata and descriptive information that varies according to sensor's context (e.g., spatial location, interacting users). Second, content of sensors (observation streams) is unbounded in size and constantly changing. On the other hand, content of text documents are commonly fixed. Finally, text content is small and infrequent in sensors. Therefore, sensor search process relies on the sensor models in addition to text descriptions.

Sensor Model:

Sensor models define constituting components and metadata of sensors, such as real-world features that they observe. This information determines types of query that a IoTSE instance can resolve. Notable sensor models in the existing literature includes W3C's Web Thing model [32] and OGC's Sensor Thing API model [89]. In our project, IoTSE is designed based on a subset of the Sensor Thing API model. Each sensor (s) has an unbounded number of datastreams (str), which store time-stamped observations (obs) (Eq. 4.3). Each datastream observes a property of a real-world feature. Consider a temperature sensor deployed in a meeting room. Its datastream observes the “apparent temperature” property of the “meeting room” feature. Each observation is a time-stamped measurement of the feature’s property.

$$s = \{str | str = \{obs | obs \text{ is a time-stamped observation}\}\} \tag{4.3}$$

It should be noted that this sensor model is not restricted to real-world, physical phenomenons such as temperature and humidity. It can model any measurable properties and features, such as daily performance of a software development team or daily revenue of a business.

4.3 Anatomy of WSR

In order to propose a kernel that provides the essential utilities for operating modules in an WSR instance and the templates to develop those modules, we need to identify modules in an WSR instance. This activity corresponds to the first phase in the instantiation framework that we have introduced in Section 3.5. While most of the identified modules have direct correspondence to the building blocks specified in the reference architecture, some modules such as the preprocessing chains and scorer chains have been instantiated specifically for WSR.

Figure 4.2 presents modules comprising an WSR instance. *Detector* and *collector* modules encapsulate detection and collection activities of the discovery task. Each collector possesses a chain of preprocessing modules to perform optional cleaning and transformation on sensor data before inserting it into *sensor collections* via *DB managers*. *Indexer* modules construct and maintain indexes of sensor collections. They can either be invoked one time or continuously in response to specific events. The search process is encapsulated into chains of *scorer* modules, which are linked in parallel or sequence. A scorer can be either a *retriever* which generates search results, or a *filter* which only reduces the set of results from other

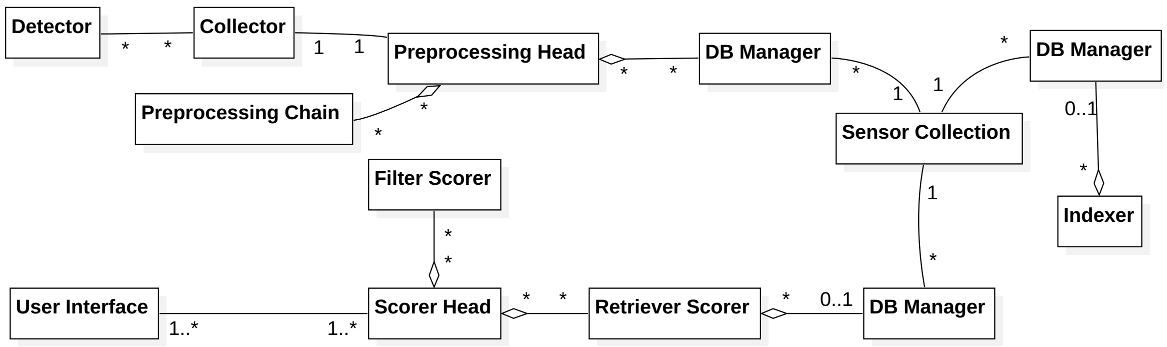
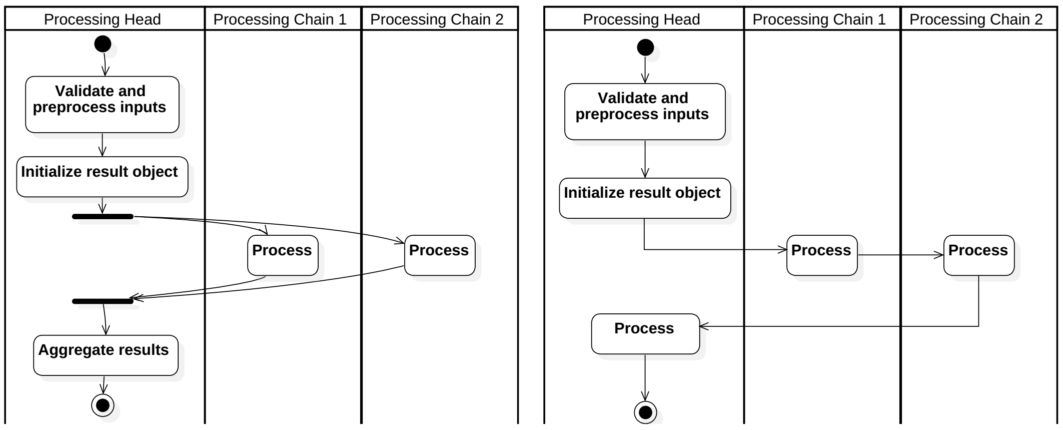


Fig. 4.2 Architecture of an adaptable and reusable Web Sensor Retrieval system.

scorers. The *user interface* module passes queries from users to scorer chains and displays the search results. It utilizes different forms of interaction, from RESTful API to Web Socket to Augmented Reality (AR).

4.3.1 Reusable Processing Chain

A notable feature of our architecture is reusable processing chains, which model the preprocessing of collected sensors and query scoring. Each processing chain consists of a chain head and unbounded number of chain members. Each member encapsulates an independent and reusable processing such as denoising, accumulating data and scoring. Chain heads arrange the execution of members in either sequence or parallel. They invoke the chain and perform the final processing step. In a sequential chain, the head acts as the last member of the chain. In a parallel chain, the head acts as an aggregator or results from chain members. Figure 4.3 presents parallel and sequential arrangement of a processing chain with two members. By organizing complex processing into independent modules, the reusable processing chain model facilitates the systematic reuse of processing mechanisms and algorithms across IoTSE instances.



(a) Parallel Processing (b) Sequential Processing

Fig. 4.3 Parallel and Sequential arrangement of user-defined processing.

4.3.2 Adaptability of the Modular Architecture

A majority of the IoTSE prototypes belonging to the $[D+R => E => R]$ class can be mapped into the reported architecture. IoTSE instances that have explicit discovery tasks, search tasks and sensor collections, such as ThinkSeek [62], Dyser [21] and IoT-SVK [17], have nearly 1:1 mapping. For instance, WoT crawlers in ThinkSeek and Dyser are mapped into detectors and collectors. Three forms of sensor scoring on textual description, spatial location and numerical values of IoT-SVK [17] can be mapped into a parallel scorer chain consisting of three chain members. IoTSE instances that operate without sensor collections (e.g., Disco-WoT [49]) or on predefined sets of sensors (e.g., CASSARAM [10]), can be mapped into a subset of our architecture that excludes detector and collector. For IoTSE prototypes that distribute the processing over multiple sensor nodes (e.g., MAX [35] and Snoogle [38]), each node can be mapped onto our architecture as a IoTSE instance.

4.4 Kernel-based Approach to Developing WSR

To support the development of modules and the engineering of WSR instances, we propose introducing a WSR kernel that lies at the core of every WSR instance. This kernel automates

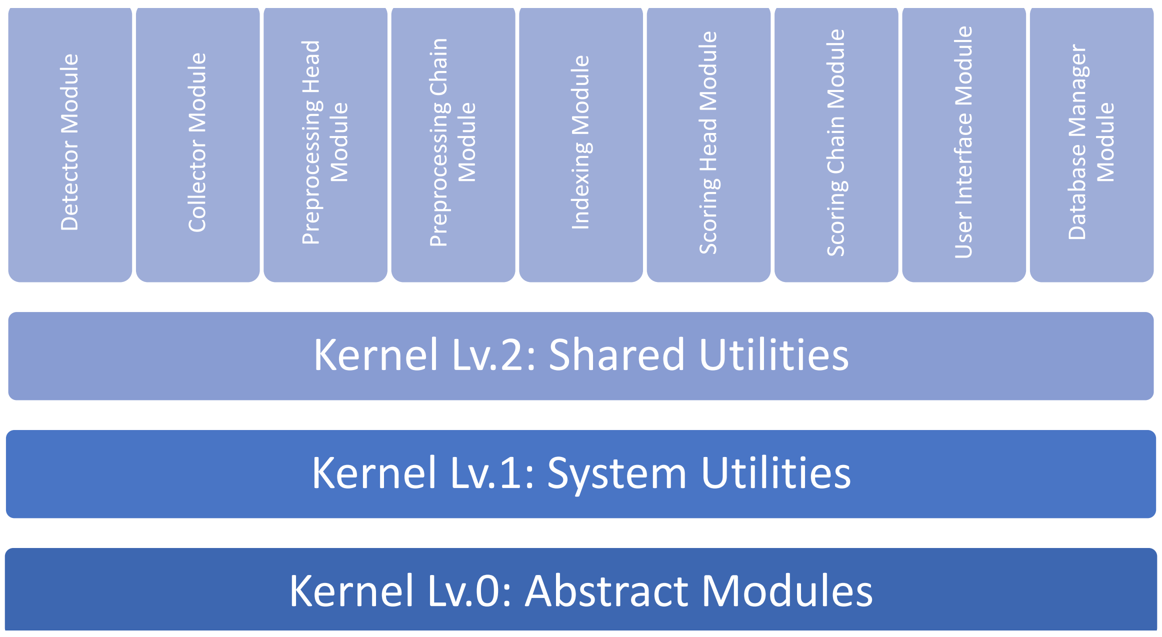

Detector Module
Collector Module
Preprocessing Head Module
Preprocessing Chain Module
Indexing Module
Scoring Head Module
Scoring Chain Module
User Interface Module
Database Manager Module
Kernel Lv.2: Shared Utilities
Kernel Lv.1: System Utilities
Kernel Lv.0: Abstract Modules

Fig. 4.4 Kernel of a Web Sensor Retrieval system

the composition of modules into a functional system. It also defines interfaces, attributes and key tasks of modules, which would be implemented by WSR developers. It simplifies the development of modules by separating system management logic from the processing defined by users, such as scoring sensors and recognizing context from observations. By providing the definitions and facilities to develop modules, the kernel also improves their interoperability. An implementation of our kernel in Python is available on our repository

The WSR kernel has three levels (Fig. 4.4). Level 0 contains abstract modules which provide the foundation to develop reusable WSR modules. Level 1 contains system utilities for composing modules into WSR instances. They validate configuration scripts given by users, confirm the existence of required modules, initialize threads to accommodate modules and setup the communication between them. Figure 4.6 presents the initialization process carried out by the Lv.1 kernel. Level 2 provides basic utilities that modules might need, such as HTTP client and server. This level is more prone to change than the lower ones.

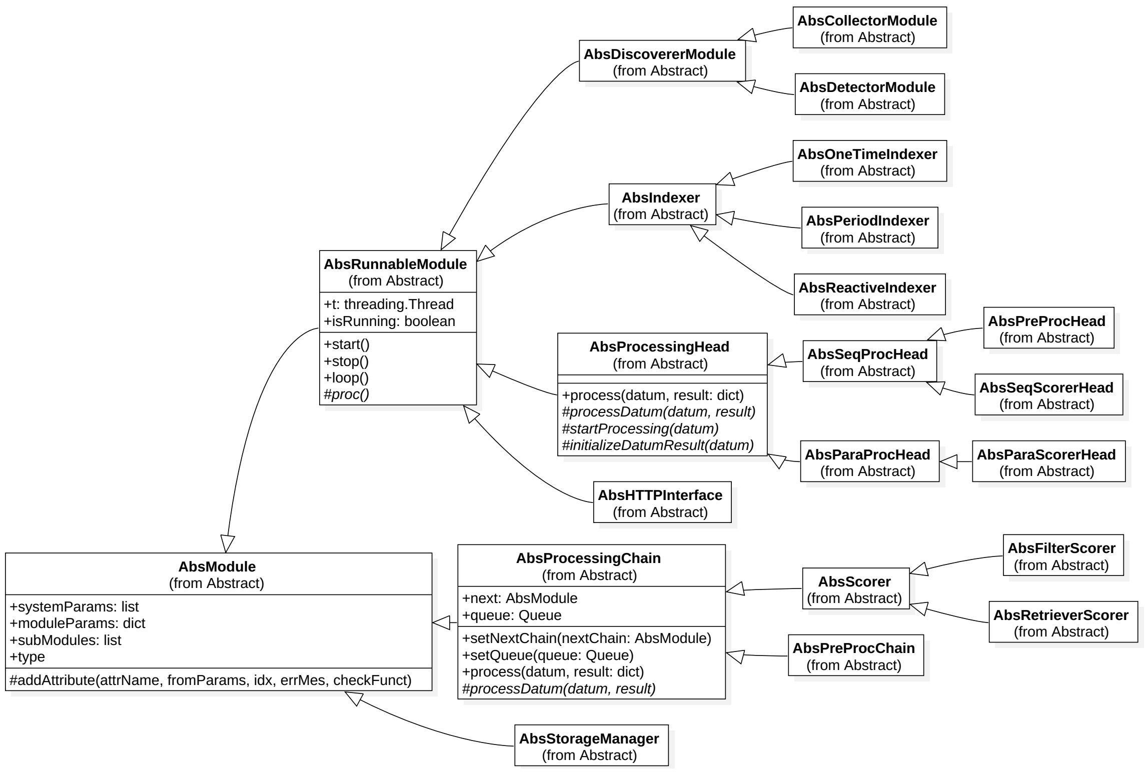

AbsCollectorModule (from Abstract)
AbsDiscovererModule (from Abstract)
AbsDetectorModule (from Abstract)
AbsOneTimeIndexer (from Abstract)
AbsIndexer (from Abstract)
AbsPeriodIndexer (from Abstract)
AbsRunnableModule (from Abstract)
+t: threading.Thread
+isRunning: boolean
+start()
+stop()
+loop()
#proc()
AbsReactiveIndexer (from Abstract)
AbsPreProcHead (from Abstract)
AbsProcessingHead (from Abstract)
+process(datum, result: dict)
#processDatum(datum, result)
#startProcessing(datum)
#initializeDatumResult(datum)
AbsSeqProcHead (from Abstract)
AbsSeqScorerHead (from Abstract)
AbsParaProcHead (from Abstract)
AbsParaScorerHead (from Abstract)
AbsHTTPInterface (from Abstract)
AbsModule (from Abstract)
+systemParams: list
+moduleParams: dict
+subModules: list
+type
#addAttribute(attrName, fromParams, idx, errMes, checkFunct)
AbsProcessingChain (from Abstract)
+next: AbsModule
+queue: Queue
+setNextChain(nextChain: AbsModule)
+setQueue(queue: Queue)
+process(datum, result: dict)
#processDatum(datum, result)
AbsFilterScorer (from Abstract)
AbsScorer (from Abstract)
AbsRetrieverScorer (from Abstract)
AbsPreProcChain (from Abstract)
AbsStorageManager (from Abstract)

Fig. 4.5 Class diagram of abstract modules defined in Lv.0 kernel. Only attributes and methods of essential classes are presented.

4.4.1 Abstract Modules

Figure 4.5 presents the hierarchy of abstract modules defined at the Lv.0 kernel. All concrete modules utilized in WSR instances are developed from the leaves of this hierarchy. The root `AbsModule` defines three sets of parameters that are used in the initialization of all modules. *System parameters* contain shared memory objects, queues and pipes for communicating with other modules. They are specified by the Lv.2 kernel in the bootstrapping process. *Module parameters* and *sub-modules* are specified by WSR instance developers in configuration scripts. These parameters are accessible to the user-defined processing methods.

Each abstract module contains concrete methods for performing system management tasks and invoking *abstract methods* which contain user-defined processing. For instance, in the `AbsRunnableModule` that represents a module running on a separated thread, the `loop()` concrete method repetitively invokes the abstract method `proc()` after every fixed

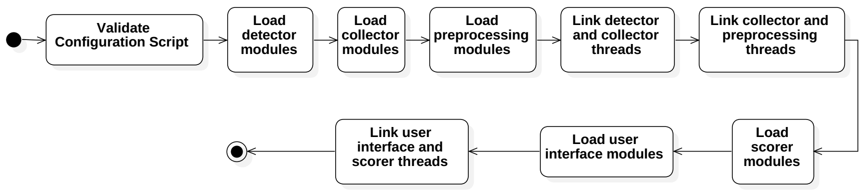

Validate Configuration Script
Load detector modules
Load collector modules
Load preprocessing modules
Link detector and collector threads
Link collector and preprocessing threads
Load scorer modules
Load user interface modules
Link user interface and scorer threads

Fig. 4.6 The bootstrapping process carried out by WSR kernel.

number of milliseconds specified in the configuration script. When developing a module based on `AbsRunnableModule`, developers only need to implement `proc()` method. The underlying thread management and looping are managed by the LV.1 kernel and the `AbsRunnableModule`.

4.4.2 Bootstrapping Process

Every WSR instance is defined by a configuration script. It lists modules comprising the instance, provides locations of their executables and declares their parameters.

The bootstrapping process composes executables of modules into a functional WSR instance. The process starts with validating the given configuration script and the existence of listed modules. Figure 4.6 presents the bootstrapping process of an ordinary WSR instance. During this process, each module performs its own validation to ensure that all of its attributes are provided in the configuration file. At the end of a successful bootstrap, a WSR instance is composed and started.

4.5 Evaluation

Due to the lack of quantitative measurements to assess the ability of a architecture in supporting diverse sensor types and queries, we assess our solution with a case study. In this case study, we assume the role of researchers who are developing a complex WSR instance to support a personal building management system. This WSR instance must interact with eight types of sensors deployed across two sensor platforms. One of which belongs to the building, the other belongs to the city. The instance must resolve five types sensor query: *(i)* searching by ID, *(ii)* by metadata, *(iii)* by observed property, *(iv)* by feature of interest (e.g., spatial location) and *(v)* by current observation. It must be deployed on a Raspberry Pi 3 (RPi3), which also acts as a Web gateway for sensors in the building, and accessible by module devices (Fig. 4.8).

Two key requirements of our WSR instance are the adaptability to work with new types of sensors and modify scoring mechanisms in the future. Our WSR architecture and kernel address these demands. The effectiveness of our approach is assessed via the degree that it meets our expectation as WSR instance developers. To be specific, we expect to be able to focus on major tasks of WSR (e.g., scoring a sensor) without having to consider the underlying mechanisms of the system, such as how query are passed from the user interface and how scorer threads are synchronized. In the next section, we present major modules that we have developed and the support that the WSR kernel provides in each one.

4.5.1 Reference IoTSE Instance

Major software and hardware components are presented in Figure 4.8. To emulate Web sensors of the building and the city, we replay sensor datasets from Intel Lab[1] and City Pulse project[2] with a Web Sensor platform (WSP) that we developed. This platform models and presents each sensor as a Web resource according to OGC Sensor Thing API standard. We deploy one instance of WSP on RPi3 to emulate the sensor platform of the building. The other one is deployed on a workstation to emulate the sensor platform of the city. We utilize an off-the-shelf REST client application, deployed on a tablet PC, to emulate a IoTSE client.

Constituting modules and their composition in the prototype is presented in Figure 4.7. Our WSR instance utilizes Mongo DB for storing, indexing and querying Web sensors. The kernel provides `AbsStorageManager` class to encapsulate sensor collections. We develop the

[1]http://db.csail.mit.edu/labdata/labdata.html
[2]http://ict-citypulse.eu/

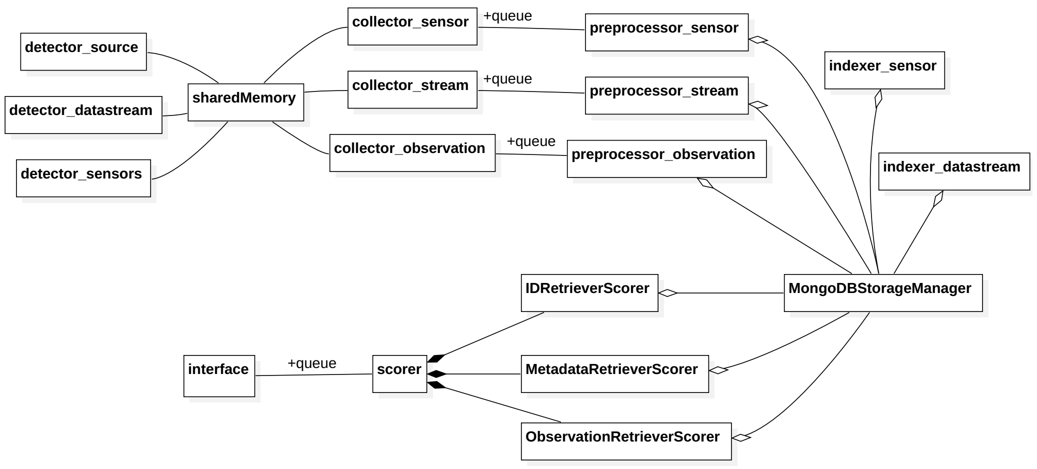


Fig. 4.7 UML class diagram of the prototype. "Queue" type associations indicate that instances of involving classes are connected with thread-safe queues. Normal associations indicate connection via function-calls.

`MongoDBStorageManager` module from this abstract class. Abstract methods for connecting to the database and performing CRUD operations are implemented using `PyMongo` library[3].

The discovery process is performed by three detectors, three collectors and three preprocessor chains. The `detector_source` module extracts URLs pointing to WSP instances from module parameters and puts them into the shared memory object, provided by the kernel. Remaining detectors extract URLs of sensors and datastreams with the assumption that sources are compliant to OGC Sensor Thing API. Collectors are extended from `AbsCollectorModule`. We implement the abstract method `collect()` to download the JSON document at the detected URL with the REST client provided in LV.2 of the kernel. The shared memory object is made available, thread-safe, to the `collect()` method by the kernel without requiring inputs from the developer. Each collector module is connected to a preprocessing chain via a thread-safe queue. We implement a chain member from `AbsPreProcChain` module to remove symbols that violate the syntax of Mongo DB from collected JSON documents. The chain head utilizes `MongoDBStorageManager` module to insert processed data to the sensor collection.

[3]https://api.mongodb.com/python/current/

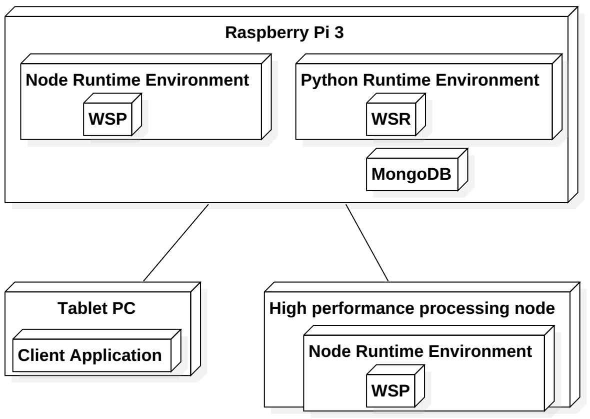
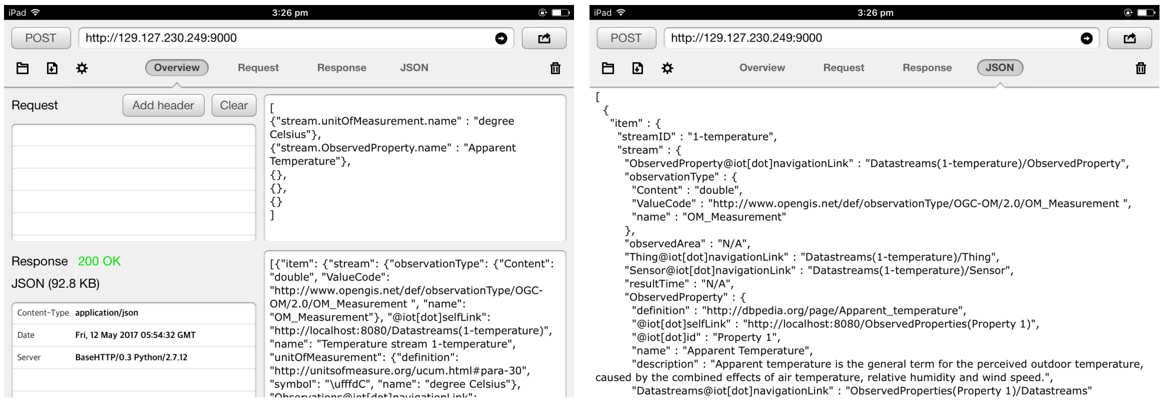


Fig. 4.8 Deployment of our WSR prototype.



```
[
{"stream.unitOfMeasurement.name" : "degree
Celsius"},
{"stream.ObservedProperty.name" : "Apparent
Temperature"},
{},
{},
{}
]
```

Response 200 OK

JSON (92.8 KB)

Content-Type	application/json
Date	Fri, 12 May 2017 05:54:32 GMT
Server	BaseHTTP/0.3 Python/2.7.12

```
[{"item": {"stream": {"observationType": {"Content":
"double", "ValueCode":
"http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_Measurement ", "name":
"OM_Measurement"}, "@iot[dot]selfLink":
"http://localhost:8080/Datastreams(1-temperature)",
"name": "Temperature stream 1-temperature",
"unitOfMeasurement": {"definition":
"http://unitsofmeasure.org/ucum.html#para-30",
"symbol": "\ufffdC", "name": "degree Celsius"},
```

```
[
 {
  "item" : {
   "streamID" : "1-temperature",
   "stream" : {
    "ObservedProperty@iot[dot]navigationLink" : "Datastreams(1-temperature)/ObservedProperty",
    "observationType" : {
     "Content" : "double",
     "ValueCode" : "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement ",
     "name" : "OM_Measurement"
    },
    "observedArea" : "N/A",
    "Thing@iot[dot]navigationLink" : "Datastreams(1-temperature)/Thing",
    "Sensor@iot[dot]navigationLink" : "Datastreams(1-temperature)/Sensor",
    "resultTime" : "N/A",
    "ObservedProperty" : {
     "definition" : "http://dbpedia.org/page/Apparent_temperature",
     "@iot[dot]selfLink" : "http://localhost:8080/ObservedProperties(Property 1)",
     "@iot[dot]id" : "Property 1",
     "name" : "Apparent Temperature",
     "description" : "Apparent temperature is the general term for the perceived outdoor temperature,
caused by the combined effects of air temperature, relative humidity and wind speed.",
     "Datastreams@iot[dot]navigationLink" : "ObservedProperties(Property 1)/Datastreams"
```

(a) Query (b) Result

Fig. 4.9 Screen captures from the REST client application.

The search process is implemented as a parallel chain of retriever scorers. For scorers, we extend `AbsRetrieverScorer` module from the kernel and implement its `retrieve(query, result)` with the query capability of Mongo DB. For scorer head, we extend `AbsParaScorerHead` and implements its aggregation method. IoTSE kernel manages the passing of queries to `retrieve()` and returning of result lists to aggregation method in the scorer head. The user interface is implemented using `AbsHTTPInterface`. Figure 4.9 demonstrates the interaction with our IoTSE instance from a REST client.

From these modules, we compose a WSR instance by declaring its components in a JSON file and invoking the bootstrapping utility. This utility connects modules based on their name.

For instance, modules `collector_[type]` and `preprocessor_[type]` are connected via a queue if their types are identical. Figure 4.7 presents the architecture of the reference IoTSE instance. Its source code in Python is available on our repository

4.5.2 Discussion

The WSR kernel simplifies the development of WSR instances by hiding irrelevant system management tasks from developers to reduce the cognitive loads and the amount of codes to develop. In the prototype, developers only implement **36 methods (500 lines of codes)** in the total **162 methods (1655 lines)** making up the system. In other word, developers using the IoTSE kernel only need to work on around *30%* of their WSR instance, lines of codes wise. This figure will be *even lower* if most modules are already available from other projects. Because of loose coupling of modules and their independence from the composition process, the resulting WSR instance is reusable and adaptable. It can be extended with additional modules. For instance, WoT crawlers [62] can be integrated as a source detector module. Influx DB can be added to provide more efficient stream storage. Its integration can be done similarly to the existing Mongo DB.

4.6 Related Works

Effective management of large-scale things over the Web relies on the discovery and search capability provided by retrieval systems [90]. Most existing research prototypes and industrial solutions are built specifically for a predefined scenario. These systems can be organized into three groups based on the type of entity being sought. The first group, commonly referred to as Discovery Service, consists of systems that *retrieve data records* of physical objects which are scattered across supply chains. Discovery Service is a component of the EPCglobal architecture [91]. BRIDGE project (funded by European Commission) investigated Discovery Service [65], and a part of its results contribute to the IETF draft on the Extensible Supply-Chain Discovery Service (ESDS) [67]. The second group of retrieval systems *works with actuation services*. These systems provide semantic models for things [22] and their functionality [23]. Based on these models, they retrieve actuation services that are semantically relevant to users.

The third group retrieval systems works with Web sensors. Early research prototypes, including MAX [35], Microsearch [46] and Snoogle [38], focus on searching for sensor nodes based on their embedded textual content. They investigate the distribution of information retrieval techniques over a network of low-power sensor nodes. As WoT and Web sensor emerge, more works focus on querying sensors based on their observations at the query time. The key challenge of these projects is predicting sensors' observations to avoid costly sensor pull operations. It is solved with regression using SVM [19], fuzzy sets [13] and patterns in observation streams [21]. Other works focus on integrating spatial information (e.g., ThinkSeek [62], IoT-SVK [17]) and quality metrics [10] into sensor queries. In general, each research prototype exceeds at a different aspects of IoTSE. However, reusing and composing them into a more optimal WSR instance are not supported.

Industrial solutions relevant to Web sensor search consist of open-source Information Retrieval (IR) systems and databases. IR systems such as Elasticsearch[4], Apache Solr[5], Xapian[6] and Indri[7] are efficient in processing large corpora. However, they lack efficient processing for streams and spatial data, which are crucial for Web sensors. Time series databases such as InfluxDB[8] and Warp10[9] are more suitable. They provide efficient storage for observation streams and query capability for their metadata. However, most stated systems require hardware with strong computing capability, which prevents them from being

[4]https://www.elastic.co/
[5]http://lucene.apache.org/solr/
[6]https://xapian.org/
[7]https://www.lemurproject.org/indri/
[8]https://www.influxdata.com/
[9]http://www.warp10.io/

deployed at the edge of WoT. Moreover, substantial amount of development effort is required to build WSR instances around these systems. This effort might not be reusable.

4.7 Summary

In this chapter, we have proposed a kernel-based approach for engineering WSR, which is a major subclass of IoTSE. In this approach, an WSR instance is engineered as a set of modules, which are plugged into a kernel. Each WSR module is a code package developed independently by component developers, based on the skeleton defined in the kernel. The interaction patterns and the format of messages exchanged between WSR modules are also defined in the kernel, which allows independently developed WSR modules to interoperate without any prior coordination other than using the same kernel. The kernel also includes bootstrapping utilities to validate and compose given modules into an operational WSR instance. The conducted case study revealed that an the kernel-based approach can reduce the amount of new source lines of code in a multi-modal WSR instance to just 30%.

Even though we proposed the kernel-based approach in the context of WSR engineering, this approach is generalizable to the IoTSE engineering because WSR is a representative subclass of IoTSE and is notably more complicated compared to other subclasses. Moreover, because a majority of abstract modules defined in the innermost level of the kernel corresponds to the building blocks defined in the IoTSE reference architecture we might not even need to modify the kernel when adapting it to a different class of IoTSE.

Future works can improve the kernel-based approach in the following ways. The first direction is standardizing the query interface of an IoTSE instance, which will allow instances to operate together as a federated search engine for broader coverage and more capable query assessment. The second direction is incorporating security and privacy protection into modules and inter-module communications to support these critical requirements from an architectural level.

Chapter 5

A Platform-based Approach to IoTSE Engineering

In Chapter 4, we have proposed a software infrastructure for engineering IoTSE instances in the form of a kernel, which provides a skeleton for developing interoperable IoTSE components and utilities for composing IoTSE instances from those components. The conducted case study has shown that the kernel-based approach can reduce the amount of new code when engineering a multi-modal IoTSE instance to only 30%. In this chapter, we propose a platform-based approach to IoTSE engineering, which extends upon the kernel-based approach to support emerging classes of IoTSE and prepare for features that might emerge in the future reuse-centric IoTSE engineering, such as automatic composition and deployment. This chapter addresses the third research objective of the thesis: to propose a software infrastructure to support the reuse-centric engineering of IoTSE.

5.1 Introduction

The kernel-based approach reported in Chapter 4 resulted in a software infrastructure to support the engineering and operation of IoTSE instances, according to the IoTSE blueprint specified in the reference architecture. For independent component developers, it helped to transform algorithms and mechanisms for different internal operations of IoTSE into reusable, composable components for engineering IoTSE instances. For search engine engineers and operators, the kernel provides the necessary software infrastructure to run IoTSE components on a computing node and control their interactions to realize the discovery and search functionality of an IoTSE instance. With these capabilities, the kernel-based approach enabled the engineering of IoTSE instances that leverage prior components and architectural patterns.

The kernel-based approach can support engineering a wide variety of IoTSE instances. Deployment wise, the kernel-based approach is compatible with centralized IoTSE instances, which represents a majority in the IoTSE literature. It also supports decentralized (peer-to-peer) IoTSE instances because each node of these systems is generally a fully functional centralized IoTSE instance itself. Operation wise, the kernel-based approach codifies an extensive workflow that covers both content discovery and query assessment activities and allows search engine operators to indicate either parallel or sequential invocation of various steps in this workflow. By supporting both centralized and decentralized IoTSE instance and codifying an extensive workflow, the kernel-based approach IoTSE engineering covers most of the prominent IoTSE classes in the literature.

Despite the already broad coverage, there are still potentials for enhancing the kernel-based approach, allowing it to support even more types of IoTSE and prepare for features that might emerge in the future IoTSE engineering. The first potential enhancement is supporting the distribution of the components of an IoTSE instance across various computing nodes (e.g., [21, 35, 17]). The second potential enhancement is supporting arbitrary workflows instead of only certain variations of a workflow. The third potential enhancement does not target the kernel but the ecosystem around it, which concerns with the accumulation of components and architectural knowledge generated from utilizing the kernel in engineering various IoTSE instances. Such an ecosystem might support future features of IoTSE engineering, such as automatic composition and deployment.

The stated enhancement potentials motivated us to extend the kernel-based approach into the platform-based approach to IoTSE engineering. This approach revolves around a software platform – *Internet of Things Search Engine Platform (ISEP)* – that acts as a hub for conducting both component development and instance engineering. This platform can either be hosted and used by an organization and its trusted partners or can be opened to component developers and search engine operators across the world.

ISEP offers developers a component framework that fulfills the role of the component templates in the innermost level of the reported kernel. ISEP offers IoTSE operators and engineers a runtime environment that they can install on their computing nodes to support the IoTSE components. It also provides tools for operators to specify the components that constitute an IoTSE instance, control the placement of the components on the computing nodes, and control the interactions of components in workflows. Different from the kernel-based approach, both the composition and deployment pattern of an IoTSE instance are independent of the component implementation, and operators can declare these patterns explicitly. Finally, to link developers and operators, ISEP includes component repositories, which can be enhanced in future works to run automatic tests on the components and accumulate architectural design decisions from the instances that the platform supports.

Our experimentations with Microservice architecture and Software Container technologies inspired the reported platform-based approach and drove the development of its reference implementation. To increase the specificity and help readers follow the discussion on ISEP, we present it in the context of engineering IoTSE instances as a collection of containerized Web services. However, the applicability of ISEP is not limited to this context.

We would like to emphasize that the platform-based approach reported in this chapter does not invalidate nor render the kernel-based approach obsolete. For large projects that aim to reach a global scale and span a significant time, the platform-based approach would be beneficial and necessary. However, there would also be projects that do not require maintaining an extensive repository of IoTSE components nor engineering a lot of IoTSE instance. For these projects, the kernel-based approach would be sufficient, as it provides a minimal infrastructure that offers all the necessary functionality.

In the following section, we first present a motivating scenario for the platform (Section 5.2). Then, we present our study to identify the functional and architectural requirements of ISEP (Section 5.3). In Section 5.4, we present the logical architecture of ISEP, comprising 13 components. To assess the feasibility of the reuse-centric IoTSE engineering with the platform-based approach, we conducted 3 case studies, which involved engineering 16 IoTSE instances of increasing complexity. The Section 5.5 presents the results from these case studies.

5.2 Motivating Scenario

The sensors and actuators embedded in “smart buildings” form IoT infrastructures, which continually generate content such as sensing data, actuating services, and virtual representatives of real-world entities. Human users and software applications consume this content to address their information and functionality requirements. IoTSE systems help users to resolve queries and retrieve IoT content from an IoT infrastructure. Whilst simple lookup can be carried out directly on an IoT infrastructure, IoTSE systems are critically important to enable an end user to raise and handle complicated queries due to the unpredictable availability of IoT devices in an infrastructure. Moreover, the resolution of complex queries might require further aggregation and transformation of IoT content. As we have discussed earlier in this thesis, we anticipate IoTSE systems to be a collection of instances, each of which covers a specific IoT infrastructure and resolves a particular type of query.

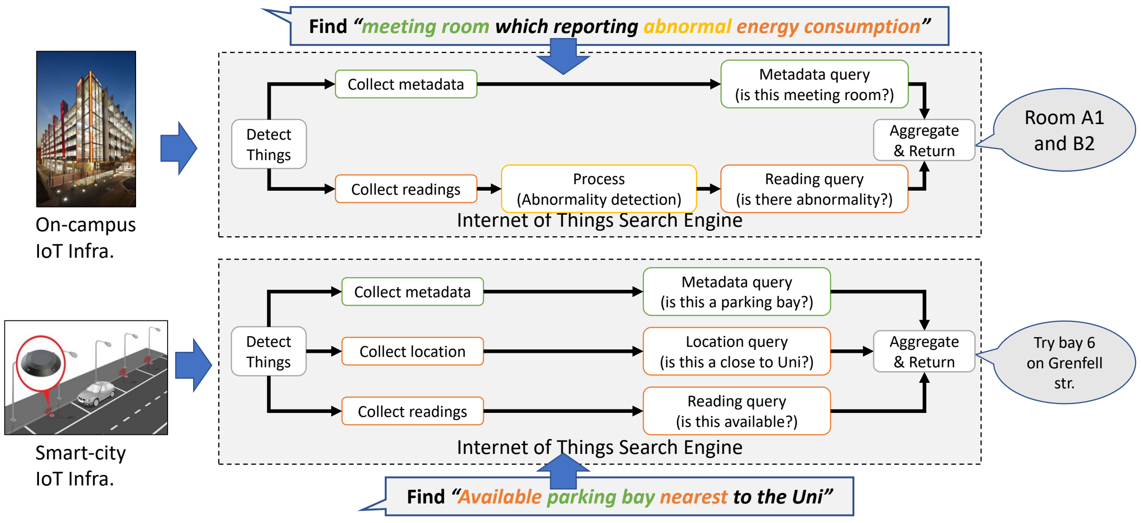

Find "meeting room which reporting abnormal energy consumption"
On-campus IoT Infra.
Detect Things
Collect metadata
Metadata query (is this meeting room?)
Aggregate & Return
Collect readings
Process (Abnormality detection)
Reading query (is there abnormality?)
Internet of Things Search Engine
Room A1 and B2
Smart-city IoT Infra.
Detect Things
Collect metadata
Metadata query (is this a parking bay?)
Collect location
Location query (is this a close to Uni?)
Aggregate & Return
Collect readings
Reading query (is this available?)
Internet of Things Search Engine
Try bay 6 on Grenfell str.
Find "Available parking bay nearest to the Uni"

Fig. 5.1 Query and an excerpt of the search result. Matching criteria are highlighted.

Let us consider two scenarios as per (Figure 5.1) for which two IoTSE instances are required. The first instance resolves queries on a smart-city infrastructure, such as finding "available parking bays that are closest to work". The second instance can monitor a smart building infrastructure to query for "conference rooms that are reporting abnormal energy consumption". Both IoTSE instances operate on virtual representatives of real-world entities (e.g., Web pages representing parking bays and conference rooms) and sensing streams (e.g., pressure sensors at parking bays, energy consumption sensors in conference rooms). Both IoTSE instances detect available virtual representatives and sensing streams from their respective IoT infrastructures, generate indexes, assess the relevance of the detected content to the given queries on different aspects, and aggregate the relevance scores to form the final search results (Figure 5.1).

The overlap exhibited by the two instances indicates a potential to share and reuse components, logical structure, and physical architecture between them. However, the current literature on IoTSE shows that these instances are engineered independently from scratch due to the lack of a suitable reference architecture for guidance and a software infrastructure. This approach of developing every IoTSE from scratch has many disadvantages. For example, the first instance might be engineered for an indexing scheme that relies on decentralized index nodes located at the network's edge, which allows it to be more sensitive to changes in real-world phenomena. The second instance, on the other hand, might be engineered as a more accurate and efficient relevance assessment mechanism. The engineering efforts of both of the instances would fail to leverage the potential of reuse and save precious resources.

An alternative approach is to develop IoTSE components independently, and to engineer IoTSE instances from the accumulated components. Architectural patterns, such as the deployment of indexing components on decentralized nodes at the network's edge, can also be reused in engineering new IoTSE instances. A common IoTSE architecture, which defines IoTSE components, their interfaces, and their exchanged data, provides a blueprint for engineering components and composing search engine instances. However, the architecture by itself is inadequate to enable architecture level reuse in IoTSE development. As an example, consider an IoTSE architecture that models IoTSE components as containerized RESTful Web services. To transform an IoTSE-related mechanism into an IoTSE component, *a component developer* must learn architecture, implement a Web service compliant to the architecture, generate and publish a container image. To compose IoTSE instances, *a search engine operator* must search for necessary container images, assess their trustworthiness and performance, and set up a software infrastructure to run container images on computing nodes and manage the flow of data and control between them. These activities cause significant overheads. The involved developers and operators also carry out redundant tasks.

Due to the identified problems above, additional software tools and libraries are required to leverage reuse in IoTSE engineering. For component developers, these tools and libraries simplify the development of components that are compliant to an IoTSE architecture. For search engine operators, these tools and libraries help find, deploy, run components, and compose them into IoTSE instances. Collectively, these tools and libraries form a software infrastructure – *a platform* – for enabling the reuse-centric engineering of IoTSE. The rest of this chapter will identify the requirements, describe an architecture, and demonstrate the feasibility of the reported software infrastructure.

5.3 Requirements of the Software Infrastructure

Given there is currently no software infrastructure to support the architecture level reuse for engineering IoTSE, we could not find the readily usable requirements from the literature to guide the development of such a software infrastructure. Therefore, our first research activity was to identify the functional requirements and the potential constraints for such a software infrastructure. For conciseness, we refer to the reported software infrastructure as the *IoTSE Platform (ISEP)*.

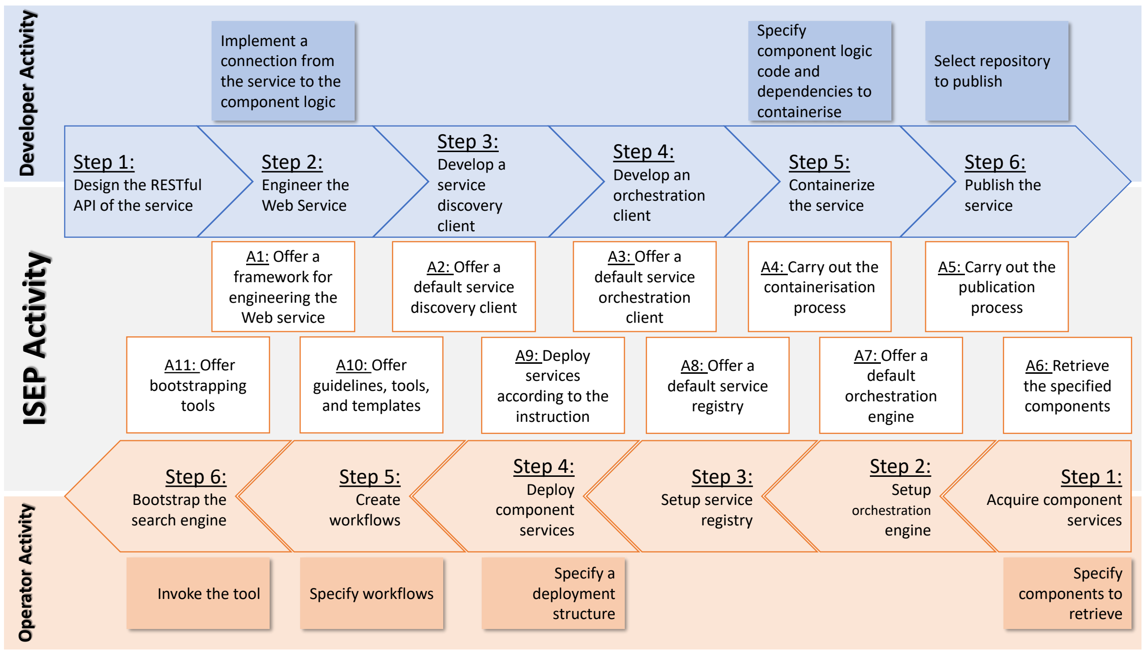

Developer Activity
Implement a connection from the service to the component logic
Specify component logic code and dependencies to containerise
Select repository to publish
Step 1: Design the RESTful API of the service
Step 2: Engineer the Web Service
Step 3: Develop a service discovery client
Step 4: Develop an orchestration client
Step 5: Containerize the service
Step 6: Publish the service
ISEP Activity
A1: Offer a framework for engineering the Web service
A2: Offer a default service discovery client
A3: Offer a default service orchestration client
A4: Carry out the containerisation process
A5: Carry out the publication process
A11: Offer bootstrapping tools
A10: Offer guidelines, tools, and templates
A9: Deploy services according to the instruction
A8: Offer a default service registry
A7: Offer a default orchestration engine
A6: Retrieve the specified components
Operator Activity
Step 6: Bootstrap the search engine
Step 5: Create workflows
Step 4: Deploy component services
Step 3: Setup service registry
Step 2: Setup orchestration engine
Step 1: Acquire component services
Invoke the tool
Specify workflows
Specify a deployment structure
Specify components to retrieve

Fig. 5.2 The process to develop IoTSE components and compose IoTSE instances, and activities carried out by the involved parties.

5.3.1 Determining the Functional Requirements of the ISEP Infrastructure

Method

We identified the functional requirements for ISEP by (1) analyzing the process carried out by component developers and operators in engineering an IoTSE instance, (2) identifying the activities that do not belong to the responsibility of developers or operators, and can be outsourced to ISEP, and (3) aggregating these activities into the functional requirements of ISEP.

The responsibility of component developers and operators and the process that they follow were derived from the above motivating scenario. Our experience in engineering component-based IoTSE instances also provided input for this analysis.

Results

We identified a six-step process that component developers utilize to transform an algorithm or mechanism into a reusable, composable IoTSE component (Figure 5.2).

Step 1: Designing the RESTful API of the component service. For instance, to transform a query assessment mechanism into an IoTSE component, a developer must decide on combinations of URL endpoints and HTTP verbs that the component uses to receive queries and return search results. The responsibility of this step belongs to neither a developer nor ISEP. *It belongs to an IoTSE architecture upon which developers, operators, and ISEP agree.*

Step 2: Engineering the Web service. Continuing from the previous example, in this step, the developer engineers a Web service based on the designed RESTful API, and connects the requests and responses of the service to the inputs and outputs of the query assessment mechanism. The engineering of the Web service is effort-intensive, duplicated among component developers. Moreover, the developer is responsible for developing the query assessment mechanism, not engineering a Web service. Therefore, *ISEP must offer a framework for engineering the Web service (A1)* according to the designed RESTful API and supporting developers in connecting their mechanisms to the Web service.

Step 3: Developing a service discovery client. To support the engineering of IoTSE instances by operators, the component service developed in the previous step must include a client so that it can take part in service discovery activities, such as registering and querying service registries. Given a component developer is not responsible for engineering a service discovery client, *ISEP must offer a default service discovery client (A2).*

Step 4: Developing a service orchestration client. IoTSE instances are engineered by orchestrating component services into a coherent system. To join an orchestration, the component service engineering in the previous steps must include a client to interact with the orchestration engine utilized by operators. Engineering an orchestration client is not a responsibility of component developers. Therefore, *ISEP must offer a default service orchestration client (A3).*

Step 5: Containerizing the service. Continuing from the previous example, in this step, the developer creates a container image that includes the query assessment mechanism, the Web service, the clients for discovery and orchestration, and other dependencies. The developer is responsible for specifying the content of the container image. *ISEP is responsible for building a container image (A4)* according to the instruction of the developer.

Step 6: Publishing the service. The developer specifies the repositories for sharing the generated IoTSE component. *ISEP is responsible for carrying out the publication process (A5).*

The process of engineering IoTSE instances from the accumulated components also consists of six steps. The inputs of this process comprise the type of query to be resolved by the engineered IoTSE instance and the computing infrastructure that will host the instance. The output of this process is an operational IoTSE instance.

Step 1: Acquiring component services. An operator is engineering an IoTSE instance for addressing queries similar to “finding meeting rooms reporting abnormal energy consumption”. This instance requires components for detecting IoT-enabled sensors, collecting meta-data, collecting sensor readings, detecting abnormality in sensing data streams, resolving queries on meta-data and sensor readings, and aggregating search results (Figure 5.1). The operator is responsible for specifying these requirements. *ISEP is responsible for retrieving the relevant IoTSE components from repositories (A6).*

Step 2: Setting up an orchestration engine. This software system controls the invocation and passing of data between services. *ISEP is responsible for offering a default orchestration engine (A7).* This engine is compatible with the default orchestration client that ISEP offers to component developers.

Step 3: Setting up a service registry. ISEP is responsible for offering a default registry (A8). This registry is compatible with the default discovery client that ISEP offers to component developers.

Step 4: Deploying the component services. In this step, the acquired IoTSE components in the first step, the orchestration engine, and the service registry are placed on computing nodes. Besides providing the infrastructure for hosting the IoTSE instance, the operator must also specify how the components are placed on the nodes. *ISEP carries out the deployment according to the instruction of the operator (A9).*

Step 5: Creating workflows. This step involves specifying the flow of data and control between components in an IoTSE instance. Continuing from the example in step 1, the operator specifies two flows of data from the detector component, through collectors, processors, query assessors, to the aggregation component (Figure 5.1). The operator also specifies the invocation order of services (e.g., parallel, sequential). *ISEP offers guidelines, tools, and templates to help the operator define workflows (A10).*

Step 6: Bootstrapping the IoTSE instance. In this step, the deployed components start accepting requests, and the orchestration engine receives the definitions of workflows to start the orchestration. The operator issues a command to start the IoTSE instance. *ISEP is responsible for providing a bootstrapping tool to carry out these initializations (A11).*

In summary, ISEP supports developers and operators with 11 activities (Figure 5.2), which can be aggregated into *the 5 key functional requirements of ISEP*. The requirement R5 on offering a runtime environment for IoTSE components is not apparent in the process. It is derived from the requirement of ISEP to deploy components on computing nodes.

R1: Providing a framework for implementing the interface of IoTSE components (From A1 - A3).

R2: Providing tools for packing and publishing IoTSE components (From A4, A5).

R3: Providing tool for acquisition, deployment, and composition of IoTSE components into IoTSE instances (From A6, A9, A10, A11).

R4: Providing a default infrastructure for composing IoTSE instances from components (From A7, A8).

R5: Providing a runtime environment for IoTSE components.

Whilst the reported process corresponds to the encapsulation of IoTSE components as containerized Web service, this process is generalizable to different architectural styles. We will address this concern in a later discussion of threats to validity.

5.3.2 Influence from IoTSE Architecture

According to the requirement R5, ISEP must provide a runtime environment for IoTSE components. It follows that ISEP must span all computing nodes that host IoTSE components. As a result, the deployment structure of IoTSE, regarding the topology and physical location of computing nodes, presents a requirement to the architecture of ISEP.

Method

To identify the influence of the deployment structure of IoTSE on ISEP, we need to know the types and the distribution of these structures. In order to acquire this information, we performed a systematic extraction and synthesis of the topology and physical location of hardware nodes utilized by representative IoTSE prototypes. From this information, we derived notable features of the IoTSE architecture, and identified their influence on ISEP.

We conducted the stated analysis on 36 representative IoTSE prototypes that have been introduced in Chapter 2. From each of the IoTSE prototype, we extracted the topology and physical location of its computing nodes. Results were synthesized by tabulation.

Results

Three topologies that involve either cloud nodes, edge nodes, or a combination of them were identified from the prominent IoTSE prototypes (Table 5.1). Centralised, cloud-based deployment was identified as the most common deployment pattern. A majority of non-IoT search engines have also utilized this pattern. Other prominent deployment patterns that were identified from IoTSE prototypes involve multiple computing nodes spanning from edges to cloud. These nodes can be arranged as a hierarchy with a central node and multiple leaf nodes, or as a collection of peer nodes.

Two notable features of the IoTSE architecture that emerged from the extracted data are the distribution of an IoTSE instance across computing nodes and the utilization of

Table 5.1 Distribution of topology and physical location of computing nodes identified in IoTSE prototypes

	Cloud	**Cloud - Edge**	**Edge**
Multiple peer nodes	1		7
One central node - Multiple leaf nodes	3	7	
One node	16		1

edge nodes with limited computing capabilities. Based on these features, we identified the requirement R6 for ISEP.

R6: The platform must be able to manage and deploy a component of an IoTSE across multiple, potentially heterogeneous computing nodes.

5.4 A Platform for Engineering and Operating IoTSE

5.4.1 Components of the ISEP

Based on the identified requirements from Section III, we now propose a software infrastructure for enabling architecture level reuse in IoTSE development and operation. This infrastructure comprises 13 components that are clustered by their functionality into five groups (Table 5.2). To help readers understand these components, we discuss them in the context of engineering IoTSE components as containerized Web services.

Table 5.2 Functionality and potential deployment location of components in the platform

Group	**Component**	**Functionality**	**Deployment**
Component Framework	Interface Library	Encoding the interface of IoTSE components, which is predefined by an IoTSE reference architecture	Development machine; inside IoTSE components

Continuation of Table 5.2

Group	Component	Functionality	Deployment
	Interface Engine	Making interface of IoTSE components accessible	Development machine; inside IoTSE components
	Interoperation Mechanism	Making the interoperation and composition of IoTSE components into IoTSE instances possible	Development machine; inside IoTSE components
Tools for Developers	Packing Tools	Supporting the packing of IoTSE components into reusable, composable software components	Development machine, or cloud back-end
	Publishing Tools	Supporting the publishing of IoTSE components to repositories chosen by the developers	Development machine, or cloud back-end
Repository	Component Repository	Storing IoTSE components published by developers	Cloud back-end
	Pattern Repository	Recording composition and deployment patterns of previous IoTSE instances for future reference and reuse	Cloud back-end
	Automatic Evaluation	Performing automatic evaluation on performance, security, and trustworthiness of accumulated IoTSE components	Cloud back-end
Tools for Operators	Component Acquisition Tools	Querying and acquiring IoTSE components from repositories	Operator machine, or cloud back-end
	Component Deployment Tools	Deploying IoTSE components on computing nodes based on the specification of operators	Operator machine, or cloud back-end
	Composition Engine	Controlling the invocation of IoTSE components according to a predefined composition pattern	Operator machine, or cloud back-end

Continuation of Table 5.2

Group	Component	Functionality	Deployment
	Bootstrapping Tools	Bootstrapping IoTSE instances according to the specification by operators	Operator machine, or cloud back-end
Runtime Environment	Runtime Environment	Comprising of everything necessary for running IoTSE components.	Computing nodes hosting an IoTSE instance

End of Table

The *Component Framework* group offers a foundation to build reusable, composable IoTSE components from algorithms and mechanisms that are contributed by component developers. This group addresses the requirement R1. In the context of engineering IoTSE components as containerized Web services, the Interface Engine is a Web server. The Interface Library is a Web service implementation library that has been extended to include the specification of RESTful APIs of all IoTSE component types. With this library, component developers can ensure that their IoTSE components are compatible with others that utilize the same library, without having to learn the reference architecture of IoTSE. The Interoperation Mechanism comprises the service discovery and orchestration clients.

The *tools for developers* group offers software tools for packing and publishing IoTSE components to different repositories that form the *repository* group. These tools address the requirement R2, while the repositories address the requirement R3. The *tools for operators* group offers IoTSE operators with software tools for acquiring, composing, and deploying IoTSE components on computing infrastructures to generate IoTSE systems. These tools address the requirements R3 and R4. Finally, the *runtime environment* group offers everything necessary for running IoTSE components on computing nodes. This group addresses the requirement R5 and is influenced by the requirement R6. In the context of the motivating scenario, the packing tools correspond to containerization tools, the repositories correspond to container registries, and the runtime environment corresponds to daemons that execute container images.

5.5 Feasibility of the Platform-based Approach in the Reuse-centric IoTSE Engineering

5.5.1 Method

In order to demonstrate the feasibility of enabling architecture level reuse in IoTSE development with the proposed platform, three case studies were conducted. In each case study, a series of IoTSE prototypes were developed and assessed. In the first case study, simple IoTSE instances that discover and search for IoT-enabled sensors based on their metadata were investigated. Partial reuse of components was investigated in the second case study, which involves IoTSE instance querying sensors by their real-time state. Full reuse of components was demonstrated in the third case study on multi-modal IoTSE. To support the development and the assessment of IoTSE prototypes, a reference implementation of the proposed platform and a test bed were developed.

The performance impact – overhead – of developing and operating an IoTSE instance as a set of loosely coupled components also contributes to the feasibility of leveraging architecture level reuse in IoTSE development. The overhead caused by ISEP in an IoTSE instance was quantified as the difference between the response time of the instance – the time between the reception of a query and the return of query results – and the total execution time of the components. In order to acquire these metrics, each IoTSE instance was subjected to a query 30 times, and the measurements were extracted from the statistics reported by the service orchestration engine. The results from all IoTSE instances were tabulated and visualized together for comparison and analysis.

A reference implementation of ISEP

The component framework of ISEP codifies service types and service interfaces presented in Table 5.3. Each type of service is codified in the framework as an abstract class. It contains resource handlers that response to incoming requests for a combination of endpoint-verb. These handlers validate and parse incoming data, populate relevant data structures, and invoke abstract methods that would be implemented by component developers to call their component logic. Resource handlers are built upon Flask RESTful library[1], which is an extension to develop Web service on a Python-based Web application framework. Gunicorn - a production-ready, WSGI (Web Server Gateway Interface) compliant Web servers, is included as the default Web server for hosting the service interface. For service orchestration, a configurable client is provided for interacting with Netflix's Conductor – an open-source and scalable workflow-based service orchestration engine.

The majority of the development and operation tools in the reference implementation of ISEP revolve around Docker and Netflix's Conductor engine. The containerisation tool is based on the image building tool that is available in the Docker daemon. Service publishing is based on the ability of the Docker daemon to push images to remote repositories. Tools for retrieving container images of IoTSE components are also based on built-in utilities of the Docker deamon. For service deployment, a combination of the Docker stack deployment ability and the "Docker-compose" utility is utilised. These tools accept YAML based declarations of the deployment structure of IoTSE instances (i.e., which service on which node) and deploy containers onto nodes automatically. The composition is handled by Netflix's Conductor. This orchestration engine accepts JSON based declarations of control flows and data flows between services, and orchestrate services accordingly.

[1]https://flask-restful.readthedocs.io/en/latest/

Table 5.3 Types of Component Services and Service Interfaces in our reference implementation of ISEP

Component	URL Endpoint	HTTP Verb	Functionality
Detector	/api/new-res-ids	GET	Invoke the content detection process
Collector	/api/res-contents	POST	Invoke content collection on the given set of URL and return a req-id for retrieving collected data.
	/api/res-contents/req-id	GET	Get the set of collected IoT content identified by req-id
Storage	/api/iot-resources	POST	Store IoT content
	/api/iot-resources/res-id	GET	Retrieve the IoT content identified by res-id
Indexer	/api/index	POST	Invoke the indexing mechanism
Searcher	/api/queries	POST	Submit a query; invoke the query processing; creates a result_id
	/api/results/res-id	GET	Retrieve a set of search results identified by res-id
Aggregator	/api/agg-results	POST	Accept a list of search results for future aggregation
	/api/agg-results/res-id	GET	Aggregate result sets linked to the res-id
Facade	/queries	POST	Submit a query; initialise query processing workflows; creates a result_id for future lookup.
	/results/result_id	GET	Retrieve a set of search results identified by result_id

The runtime environment of the platform consists of Docker daemons that resides on computing nodes hosting an IoTSE instance. These daemons are connected to each other and to the host that offers operator's tools.

Test Bed

All IoTSE prototypes and experiments were hosted by a Linux-based workstation with a quad-core Xeon E3 CPU and 8GB of RAM. Up to three virtual machines would be hosted on the workstation, depending on the deployment pattern of an IoTSE prototype in an experiment.

IntelLab dataset[2] was utilised to demonstrate the functionality and performance of IoTSE prototypes. This dataset consists of data from 54 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004. These sensors report timestamped topology information, along with humidity, temperature, light and voltage values once every 31 seconds.

In order to expose these data streams on the Web as IoT-enabled sensors, a NodeJD-based sensor gateway was developed. This gateway transforms metadata and sensing data of sensors into JSON documents and serves them as Web resources according to the SensorThings API standard[3] developed by The Open Geospatial Consortium (OGC).

5.5.2 Case Study 1

Analysis and component development

This case study addresses the class of IoTSE that discovers and searches for IoT-enabled sensors based on their metadata. Functionalities of these systems were identified as locating IoT gateways that serve IoT-enabled sensors on the Web, detecting the endpoints of these

[2]http://db.csail.mit.edu/labdata/labdata.html
[3]http://www.opengeospatial.org/standards/sensorthings

sensors, retrieving sensor metadata, storing and indexing sensor metadata, and resolving queries for sensors with the collected data and generated index.

Component services to be implemented were derived by matching the identified functionalities with the types of component services codified in the reference implementation of the platform (Table 5.3). *Six component services* were identified and implemented:

Source Detector is responsible for detecting IoT gateways. It belongs to the Detector component type. For detecting sources of IoT data, the IoTSE literature reports various approaches such as utilising multicast DNS (mDNS), multicast CoAP, Simple Service Discovery Protocol (SSDP) and developing specialised crawlers for the Internet of Things (e.g., [62]). As our experiment involves only one IoT gateway and our focus is on the integration of IoTSE components, not developing state-of-the-arts ones, we opted to provide the address of IoT gateway to the source detector instead of implementing detection mechanisms.

Sensor Detector is responsible for detecting endpoints of IoT-enabled sensors hosted by the gateway. It belongs to the Detector component type. This task is complicated as the interface to access sensors varies significantly between IoT gateways. We developed sensor detection based on SensorThings API standard, which specifies templates for endpoints of IoT-enabled sensors. As our IoT gateway was also compliant to the SensorThings API, sensor detection was possible.

Metadata Collector is responsible for collecting metadata of detected sensors. It belongs to the Collector component type. Data extraction and parsing were implemented based on the data template defined in the SensorThings API standard.

Metadata Searcher is responsible for storing, indexing, and resolving queries on sensor metadata. This component service combines the capability and interface of Storage, Indexer, and Searcher component types. This combination was motivated by the large and frequent flow of data between them. Implementation of this component was based on the capabilities offered by MongoDB – a NoSQL database.

An *Aggregator* and a *Facade* component service were implemented to complete this IoTSE instance, according to the architectural vision codified in the platform. The aggregator was implemented to return the intersection of multiple lists of search results based on sensor ID. The facade is responsible for offering a single point of access to the IoTSE instance. It was instantiated from the platform without alteration.

Architectural patterns

Two composition and two deployment patterns were utilised to generate four IoTSE instances from six component services.

Two deployment patterns used were *centralised* and *distributed*, in which detector and collector components were deployed on one virtual machine while remaining components were deployed on a different virtual machine.

Two composition patterns used were *Parallel Discovery (PD)* and *Interlaced Discovery (ID)*. In PD, the detection and collection of IoT content are carried out independently from the query assessment activities. In ID, the detection and collection of IoT content is invoked only on reception of a query. This pattern emerges from IoTSE prototypes.

Results

Four IoTSE instances were subjected to a query for IoT-enabled sensors that measure "apparent temperature" in "degree Celsius". All instances achieved perfect precision and recall because a database was utilised for storing and resolving queries on sensor metadata. For conciseness, query and search result samples of each instance are omitted in favour of the excerpts from multi-modal IoTSE instances in Case Study 3 (Figure 5.3).

The response time IoTSE instances utilising the Interlaced Discovery pattern was around 6 seconds on average, while instances with the Parallel Discovery pattern responded after 3 seconds on average (*meta_ID_C* and *meta_PD_C* in Figure 5.4). The overhead on both patterns was around 1 second (Figure 5.5a). Due to having much shorter response time, the pattern PD was subjected to higher percentage of overhead, which reaches 35% on average (Figure 5.5b).

5.5.3 Case Study 2

Analysis and component development

This case study addresses the class of IoTSE that discovers and searches for IoT-enabled sensors based on their sensing value. This class is nearly identical, functionality-wise, to the one in the first case study. The only difference is the type of IoT content utilised for assessing queries: real-time sensing value. Consequently, the majority of analysis and components developed in the previous case study are applicable to this case study.

New components developed in this case study include *Reading Collector* and *Reading Searcher*. The reading collector was implemented to extract and parse real-time sensing value of IoT-enabled sensors from gateway. The reading searcher is functionally equivalent to the metadata searcher. For large sensor datasets, time series databases such as Influx DB[4] are suitable solutions. However, given the small scale of IntelLab dataset, we opted for Mongo DB.

Architectural patterns

Patterns specified in the first case study were reused to generate four IoTSE instances from six components.

Results

The generated IoTSE instances were subjected to a query for sensors whose latest readings are less than 25. Similar to the first case study, these instances also achieved perfect precision

[4]https://www.influxdata.com/

and recall, at the cost of high response time. The average response time and overhead of IoTSE instances in this case study (*rt_ID_C* and *rt_PD_C* in Figure 5.4, 5.5a, 5.5b) were nearly identical to the results from the first case study.

5.5.4 Case Study 3

Analysis and component development

This case study addresses the class of IoTSE that discovers and searches for IoT-enabled sensors based on both their metadata and sensor readings. Functionality-wise, this class is a hybrid between two prior classes. Consequently, the prototypes of this class can be composed entirely from component services developed in the previously reported two cases.

Architectural patterns

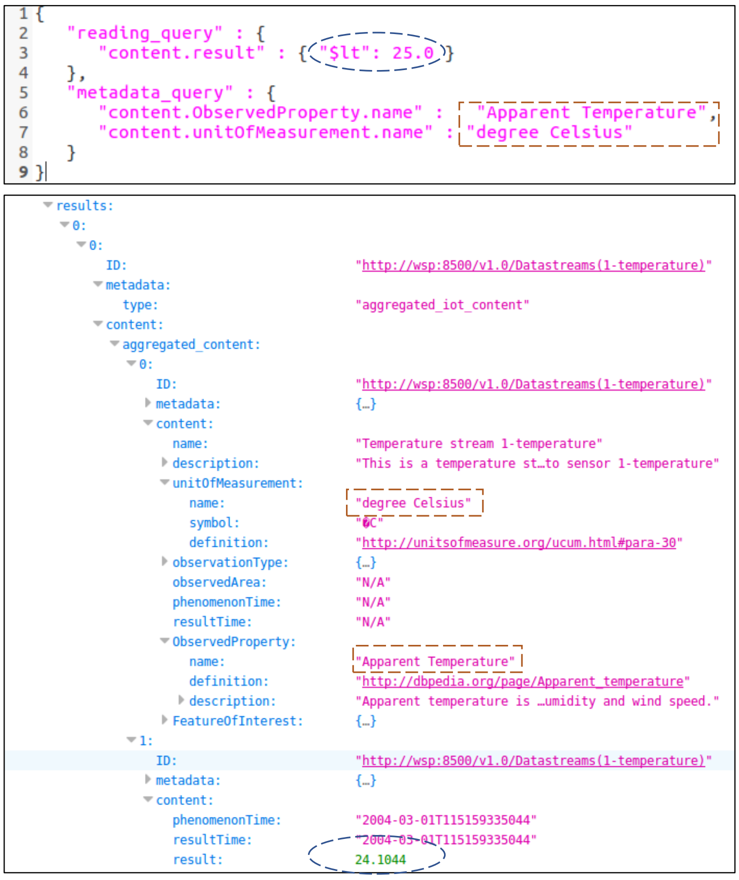
Due to the existence of two searcher components, two new composition patterns were introduced in this case study. In *Sequential Search (SS)* pattern, query assessment on different aspects of IoT content is carried out as a sequence, followed by the aggregation. In *Parallel Search (PS)*, query assessment on different aspects of IoT content is carried out in parallel, and then the aggregation is invoke when all sets of search results are provided.

In summary, four composition patterns (PD-SS, PD-PS, ID-SS, ID-PS) and two deployment patterns were utilised to generate eight IoTSE prototypes in this case study.

Results

The generated IoTSE instances were subjected to a query for sensors that measure "apparent temperature" in "degree Celsius", whose latest readings are less than 25. The search result comprised SensorThings-compliant JSON representations of matching sensors (Figure 5.3).

The response time was measured to be from 6 to 13 seconds on average, with the overhead contributing around 1 to 2 seconds on average (*multi_ID_PS_C*, *multi_ID_SS_C*,



```
{
   "reading_query" : {
      "content.result" : {"$lt": 25.0}
   },
   "metadata_query" : {
      "content.ObservedProperty.name" :  "Apparent Temperature",
      "content.unitOfMeasurement.name" : "degree Celsius"
   }
}
```

```
results:
  0:
    0:
      ID:                                  "http://wsp:8500/v1.0/Datastreams(1-temperature)"
      metadata:
        type:                              "aggregated_iot_content"
      content:
        aggregated_content:
          0:
            ID:                            "http://wsp:8500/v1.0/Datastreams(1-temperature)"
            metadata:                      {…}
            content:
              name:                        "Temperature stream 1-temperature"
              description:                 "This is a temperature st…to sensor 1-temperature"
              unitOfMeasurement:
                name:                      "degree Celsius"
                symbol:                    "�C"
                definition:                "http://unitsofmeasure.org/ucum.html#para-30"
              observationType:             {…}
              observedArea:                "N/A"
              phenomenonTime:              "N/A"
              resultTime:                  "N/A"
              ObservedProperty:
                name:                      "Apparent Temperature"
                definition:                "http://dbpedia.org/page/Apparent_temperature"
                description:               "Apparent temperature is …umidity and wind speed."
              FeatureOfInterest:           {…}
          1:
            ID:                            "http://wsp:8500/v1.0/Datastreams(1-temperature)"
            metadata:                      {…}
            content:
              phenomenonTime:              "2004-03-01T115159335044"
              resultTime:                  "2004-03-01T115159335044"
              result:                      24.1044
```

Fig. 5.3 Query and an excerpt of the search result. Matching criteria are highlighted.

multi_PD_PS_C, and *multi_PD_SS_C* in Figure 5.4, 5.5a, 5.5b). Around 8% of average overhead, which was the lowest among studied instances, was achieved by the ID-PS pattern. Due to the limitation of our computing infrastructure, the IoTSE instances that utilizes multiple virtual machines consumed all the processing capability and incurred large response time, which was not representative due to the overloaded state of our computer. Therefore, we omitted the measurements from the IoTSE prototypes with distributed deployment patterns to avoid skewing the results.

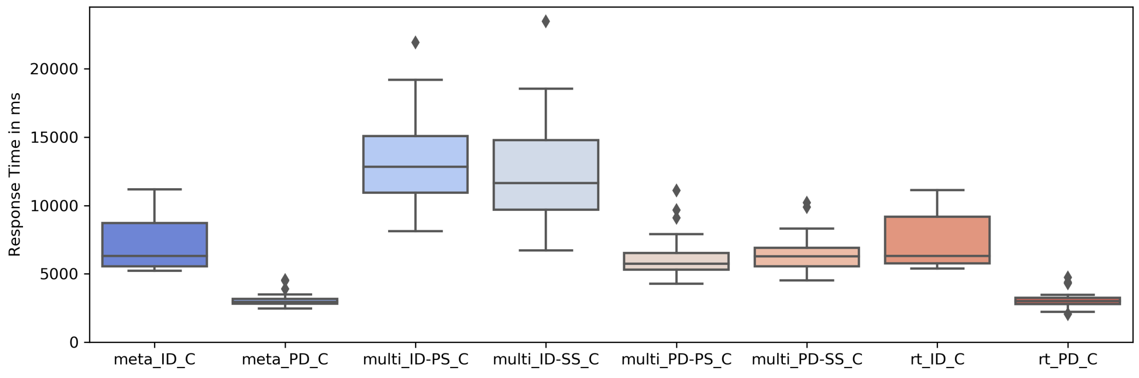


Fig. 5.4 The query response time of IoTSE instances, measured in milliseconds.

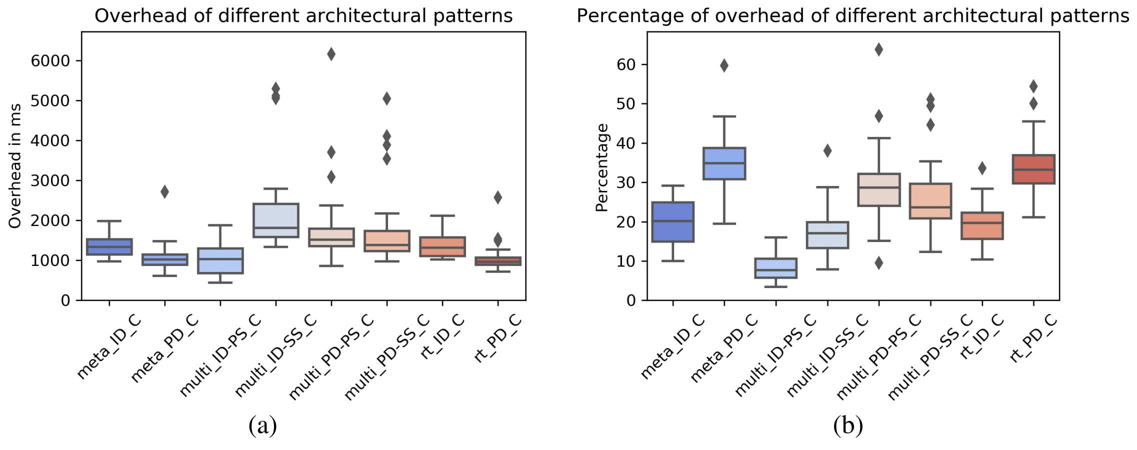


Fig. 5.5 The overhead caused by the platform on IoTSE instances, measured in milliseconds (a) and percentages of the response time (b).

5.6 Discussion

With the three case studies, we have demonstrated the feasibility of leveraging architecture level reuse in engineering IoTSE with the support of a shared software infrastructure. The first two case studies demonstrate that it is possible to build IoTSE from components independently developed according to a shared architectural vision. The third case study presented the prospect of generating more complex IoTSE instances by combining the components from simpler ones. As more components had been accumulated, each following case study exhibited a further reduction in the development effort. In the third case study, IoTSE instances of a new class were generated without new development effort.

The reported case studies also demonstrate the simplicity of altering composition and deployment patterns of an IoTSE. With the support of the proposed platform, an operator can modify the workflows and deployment of an IoTSE instance without changing the implementation of its components. This ability allowed us to isolate and study the impact of architectural patterns on performance of an IoTSE instance, given a fixed set of components. The significant influence of architectural patterns on the performance of IoTSE instances, such as the drop of average response time by half when the ID-PS pattern is replaced by PD-PS, suggests a potential for creating substantial advances in IoTSE by combining the Software Engineering research and the IoTSE research. With the ability to alter architectural patterns at ease, and to accumulate these patterns along with the components for building future IoTSE instances, the proposed platform is expected to serve as the catalyst to facilitate such incorporation.

A limitation of our case studies was the simplicity of both reference implementation of the platform and the IoTSE components, which resulted in high response time and overhead in the prototypes. This limitation is attributed to both the lack of access to the implementation of the state-of-the-art solutions, and the question that our case studies address. Specifically, an optimal implementation might improve the performance of prototypes; however, such an improvement does not offer any new insights into the feasibility of leveraging reuse in IoTSE engineering using our proposed platform.

Another limitation is that we cannot draw a definite conclusion on the reduction in development effort and the overhead as the result of the platform-based approach to IoTSE development. The results from the three cases, albeit informative, are specific to the utilized IoTSE reference architecture, to the reference implementation of ISEP, and to the limited computing resource that was used to host the IoTSE prototypes. To generalize this finding, additional implementations of ISEP based on different IoTSE reference architectures are required. Such study is only possible when the platform is introduced to the broad research community and adopted.

5.6.1 Threats to validity

Threats on identification of functional requirements

The identified requirements are based on the motivating scenario, which utilizes a specific architectural vision of IoTSE. This architecture defines a specific functional decomposition of IoTSE and dictates the use of containerized Web services for encapsulating IoTSE components. It influences the processes utilized by developers and operators, and therefore presents a threat to the external validity of the identified requirements. This threat was minimized in two ways. First, the processes were confined to the transformation of IoTSE components into reusable building blocks and the their utilization. How IoTSE components were scoped did not contribute to these processes. Second, we identified a generalized version of the activities specific to building IoTSE components as containerized Web services, and derived the requirements from these activities.

Threats on the identification of architectural requirements

One can argue that the analyzed IoTSE prototypes do not represent the IoTSE as a whole. Therefore, the representativeness of the analyzed prototypes presents a threat to the external validity of our findings. This threat was addressed by a systematic process of identifying and analyzing the prototypes, which was conducted and reported in Chapter 2.

Threats to the validity of the case studies

The external validity of this study faces two threats. The lack of involvement from external parties in case studies is the first threat. One can argue that the case studies were only possible due to our familiarity with both the platform and the IoTSE architecture. To address this threat, we avoided utilizing the knowledge that has not been presented in this paper nor in the reference implementation when conducting the case studies.

The reliance on a specific architectural vision of IoTSE is the second threat. As all the analysis and functional decomposition of IoTSE instances in three case studies utilized the specific eight service types (Table 5.3), one can argue that the proposed solution is applicable only to this architecture. To address this threat, we emphasize that the platform, as described in Section 5.4, is independent of the chosen architectural vision of IoTSE. It provides a software infrastructure to enable architecture level reuse in IoTSE, based on an architectural vision that is shared between involving component developers and search engine operators. We assert that if a chosen architectural vision can decompose IoTSE into non-overlapping components, the demonstrated architecture level reuse can be achieved.

5.7 Related Works

Our work approaches IoTSE from a unique perspective which has been to date relatively unexplored by the existing IoTSE literature. We investigate the architectural support and the tools required to address the diversity of the solution space of IoTSE via independent development of components and separation of IoTSE operation from development. We observe that the existing work focuses exclusively on a class of IoTSE. For example, real-time sensor search (Dyser [21], CSS [14, 13, 15]), context-based sensor search (CASSARAM [10, 12], ThingSeek [62]), functionality search based on semantics [22, 23], object localization (Snoogle [38], MAX [35], OCH [37]), and discovery services based on EPCglobal specifications [29]. These efforts have concentrated on particular technical perspectives to create new mechanisms and algorithms for efficient indexing, query processing, or simliar functionalities.

As a result, there are only a few pieces of work in the IoTSE literature that are related and directly comparable to our ideas reported in this paper. These related work model IoTSE components as shared software libraries, and offer templates to simplify their development. For example, ThingSeek [62] focuses on detecting URL of sources and sensors, while Kernel-based IoTSE [93], which was proposed in Chapter 4, covers the entire workflow of an IoTSE instance. These efforts share two limitations. First, developing an IoTSE instance requires deep integration of components – distributed as shared software libraries – into the code base of the system. This integration impedes the modification and independent scaling of components. Second, the shared libraries make assumptions on how an IoTSE component should be implemented (i.e., variables to use, functions to implements). Our solution reported in this chapter addresses both of these limitations. By modelling IoTSE instances as workflow-based composition of containerized Web services, our solution simplifies the modification of an IoTSE instance and allows independent scaling of its components. Moreover, our solution only makes assumptions on inputs and outputs of components to ensure their interoperability. We give component developers total control over data models, algorithms, and technologies utilized for implementing components.

Software Engineering and Service-oriented Computing literature addresses various aspects of engineering component-based service-oriented software systems, concentrating on recommending services for composition [94, 79, 95, 96], reconfiguring service compositions [97–99], enabling service interoperability [100], and documentation for further composition [101]. Our proposed architectural solution lays a foundation for applying these techniques in the context of CBSE of search engine systems for IoT.

5.8 Conclusion

In this chapter, we have proposed a platform-based approach for IoTSE engineering. In this approach, the engineering and operation of IoTSE instance revolve around a software infrastructure called the IoTSE Platform (ISEP). This platform can be divided into three parts.

The part given to component developers has a similar role to the innermost level of the kernel proposed in the previous chapter. The developer side of ISEP comprises a component framework and software utilities to help to transform algorithms and mechanisms into reusable, composable IoTSE components. The operator side of ISEP comprises engines and runtime environments for running individual IoTSE components on computing nodes and controlling their interactions according to the workflows specified by the search engine operators. The operator side also includes tools to help operators with defining the workflows and the deployment structure of the engineered IoTSE instance. The backend of ISEP includes repositories of components and patterns. It also includes mechanisms for the automatic evaluation.

The most notable advance of the platform-based approach over the kernel-based approach is the total separation of the architecture from the implementation of an IoTSE instance. Given the same set of components, operators can specify various workflows and deployment structures, and generate multiple IoTSE instances, without having to alter the implementation of components. Three case studies, which engineered 16 IoTSE instances, have demonstrated this ability. Notably, in the third case study, complex IoTSE instances were engineered entirely from the components of simpler ones. This ability is unprecedented in the IoTSE literature.

Future works can incorporate security, privacy, and trust support to the software infrastructure to provide built-in protection to IoTSE instances. Future works can also consider automatic IoTSE engineering, based on the components and patterns that will be accumulated. Implementation wise, future works can also improve the overall efficiency of the reference implementation of ISEP based on containerized Web services.

Chapter 6

Conclusion

At the beginning of the effort on developing novel discovery and search systems for the Internet of Things, we recognized the potential and the necessity of leveraging reuse in engineering these systems, and the lack of architectural support for such reuse-centric IoTSE engineering.

Due to the diversity of IoT content, which in turn generate a diverse solution space for IoTSE, we believed that the successful realization of IoTSE does not hinge on the discovery of an unknown, magical algorithm that can "find anything, at any time and any place". Instead, this success depends on the ability to combine the state-of-the-art mechanisms for discovering and querying different IoT content types to address any combination of IoT content types that is demanded by incoming queries.

Moreover, due to the sensitivity and location-dependent nature of IoT content, and due to the emerging trend of decentralizing IoTSE in the literature, it was apparent that IoTSE will emerge, not as a large and centralized entity, but as a collection of smaller instances that monitor different IoTSE infrastructure and specialize in different types of IoT query.

Both observations pointed toward enabling reuse-centric engineering as a potential solution for realizing the Internet of Things Search Engines. The aims of this thesis were proposing the reference architecture and the software infrastructure necessary for engineering

IoTSE instances, which support leveraging the reuse of components and architectural patterns. In this chapter, we summarize our achievements in realizing these aims and the limitations of our proposed solution. We also discuss further applications areas that might benefit from the proposed solution and finally conclude this thesis with a discussion on future works.

6.1 Thesis Summary

To summarize the thesis, we revisit its three objectives that were specified in Chapter 1 and present details on how they have been achieved with the research conducted in this thesis.

6.1.1 Providing a holistic insight in the current state of IoTSE

The first objective of the thesis was to provide a holistic insight into the current state of IoTSE research and development. To achieve this objective, we conducted a systematic and comprehensive study that cover over 200 research works on IoTSE, 6 industrial works, and 4 related international standards (Chapter 2). By the time this study was completed, its coverage in terms of the number and the diversity of IoTSE works was unprecedented.

By analyzing the number of publication, the number of in-field citations, and the number of works receiving in-field citations, we identified a cluster of research works published between 2010 and 2012 that influences majority of IoTSE literature. Most of these works consider IoTSE as a centralized system that retrieves sensors based on either their static semantic description or their latest sensing value. Together, they drove the perception of what IoTSE should be.

To analyze the functionality and internal operations of IoTSE, we proposed a model called Meta-path and a basic modular decomposition of IoTSE. The meta-path model provides a succinct and precise description an IoTSE instance regarding the types of IoT content that it utilizes for assessing queries and deriving search results. We assessed 36 representative

IoTSE research works, which were chosen from over 200 related works, against the meta-path model and the shared modules.

Results of the analysis showed that searching for objects based on their ID or metadata ($R \Rightarrow R$) is the most common form of IoTSE, followed closely by searching for objects using their real-time state (e.g., sensor readings, location) and searching for sensor streams ($D \Rightarrow D$). Surprisingly, searching for real-world functionality is not commonly supported even though it is crucial in the interaction with smart environments in the IoT. Regarding the internal operations, most IoTSE prototypes do not consider the mobility of IoT-enabled things. The ranking of IoT content by their natural order (i.e., Query Independent ranking) also lacks the support, despite playing a crucial role in the success of Web Search Engines. Notably, security, privacy, and trust are also barely addressed by prototypes.

Based on the study, we identified open issues in IoTSE research, including IoT crawling, enabling location-based search, supporting dynamic IoT content, supporting scalability to both extremes of the spectrum, addressing security, privacy, and trust, and finally, addressing the diversity of the IoTSE solution space by leveraging the overlaps among IoTSE instances.

6.1.2 Proposing an IoTSE Reference Architecture

The second objective of this thesis was to propose a reference architecture that captures the commonalities of IoTSE. The research presented in Chapter 3 addresses this objective. Based on the representative IoTSE prototypes and the common modules identified in the prior survey (Chapter 2), we developed the reference architecture by extracting and synthesizing the workflows and deployment structure of IoTSE prototype iteratively.

By following the stated method, we identified 18 functional components of IoTSE from the overlapping activities of workflows. The extracted workflows were generalized and organized into a taxonomy of 13 composition patterns, which can be combined to describe a wide range of workflows in an IoTSE instance. Extracted deployment structures were generalized into a taxonomy of 6 deployment patterns. Finally, we introduced a framework to guide the instantiation of IoTSE instances from the reference architecture.

The proposed IoTSE reference architecture was evaluated by mapping into two IoTSE prototypes – Dyser [21] and IoT-SVK [17]. These works are representative and present the extremes in terms of the architectural complexity in the IoTSE literature. In both cases, both the workflow and the deployment structure of the prototype were entirely captured by the components and patterns proposed in the reference architecture. The case studies conducted in the following research on the software infrastructure, which involve engineering IoTSE instances based on the reference architecture, further validated its suitability and utility.

6.1.3 Proposing a Software Infrastructure to support Reuse-centric IoTSE Engineering

The last objective of this thesis was to propose a software infrastructure to support the reuse-centric engineering of IoTSE. The purpose of this infrastructure was to simplify the development of reusable, composable IoTSE components, and enable the engineering of IoTSE instances from those components.

Our research on the software infrastructure started with a kernel-based approach to IoTSE engineering. This approach was inspired by modern operating systems, which have most of their functionality and operations developed as plug-able components around a microkernel. Similarly, in the kernel-based approach to IoTSE engineering, each IoTSE instance consists of modules that are plugged into an IoTSE kernel, which defines abstract templates of modules and utilities to bootstrap and operate the IoTSE instance. Each module is a code package that was developed upon the abstract templates in the kernel. These templates codify the building blocks and patterns defined in the reference architecture. An IoTSE operator has limited control over the workflow of an IoTSE instance via the choice of modules to include. In a case study, which involves engineering an IoTSE prototype with a reference implementation of the kernel, the effort for engineering the instance was reduced to just 30%.

The kernel-based approach supported a wide range of IoTSE instances. However, we also identified some enhancement potentials that would extend the support of the kernel and prepare it for features that might emerge in the future IoTSE engineering. These enhancement potentials motivated us to propose a platform-based approach to IoTSE engineering. We would emphasize that this approach did not render the kernel-based approach obsolete, as the minimal infrastructure offered by the kernel-based approach would be preferable in projects where the additional features of the platform are not required.

The platform-based approach revolves around an Internet of Things Search Engine Platform – ISEP – that provides a framework for developing reusable, composable IoTSE components, provides repositories for accumulating components, provides the runtime environment for executing IoTSE components across various computing nodes, and offers tools for engineering IoTSE instances from accumulated components. We developed a prototype of the platform and conducted three case studies, in which we engineered 16 IoTSE instances of increasing complexity. Notably, in the third case study, we engineered 8 multi-modal IoTSE instances entirely from the components of simpler instances. Such degree of reuse in IoTSE engineering is unprecedented in the IoTSE literature and suggests the feasibility of the reuse-centric IoTSE engineering based on our proposed reference architecture and software infrastructure.

6.2 Limitations

This thesis proposed a reference architecture and software infrastructure for engineering IoTSE instances, which support leveraging prior components and architectural patterns. As the reference architecture focuses on the commonalities of IoTSE in terms of their operation, it offers limited support to aspects of IoTSE that are not common and do not contribute directly to the content discovery and query assessment. The following sections discuss these limitations.

6.2.1 Limited Support for Decentralized IoTSE

A decentralized IoTSE instance consists of functionally equivalent peer nodes, each of which maintains a subset of the IoT content discovered by the IoTSE instance. Nodes in a decentralized IoTSE instance can generally operate independently. They tend to resolve incoming queries locally before propagating them to other nodes in the instance. This class of IoTSE has become more prevalent in recent IoTSE literature.

The proposed IoTSE reference architecture can model individual nodes in a decentralized IoTSE instance. It also identified the edge-based decentralized pattern as a part of its deployment pattern taxonomy. However, it offers limited support beyond that. For instance, it does not model the topology of the overlay network, the routing algorithm, nor the consensus mechanism that a decentralized IoTSE instance utilizes. This limitation extends to both the kernel-based and the platform-based approach.

Our focus on capturing the commonalities of IoTSE in the reference architecture led to this limitation, as decentralized IoTSE has yet to be the norm in the IoTSE literature. We addressed this problem partially by including a vertical layer named “Management” which can be extended in the future to include components that support P2P IoTSE instances.

6.2.2 Limited Support for Security, Privacy, and Trust

IoTSE instances work with highly sensitive content, such as the sensing data from a smart-home or a health monitoring infrastructure. Therefore security, privacy, and trust are without a doubt even more critical to the success of an IoTSE instance than its ability to resolve queries. The survey on the IoTSE literature in Chapter 2 reported that these features are the least supported ones by the representative works, and highlighted them as the crucial open research issues.

The proposed IoTSE reference architecture, however, offered only limited support for modeling security, privacy, and trust measures of an IoTSE instance. This limitation was caused by our focus on capturing the commonalities of IoTSE regarding their operation. As stated previously, supporting security, privacy, and trust has not been common among representative IoTSE works and does not contribute directly to discovering IoT content and resolving queries. Therefore, the reference architecture has not capture components and patterns for securing IoTSE.

We addressed this limitation partially by including a vertical layer named "Security" as a placeholder for security, privacy, and trust measures to be identified in the future.

6.3 Extending Results

The studies and techniques presented in this thesis target the reuse-centric engineering of IoTSE. However, these solutions might be applied to problems in different domains, such as IoT Gateway engineering and Software Architecture.

6.3.1 IoT Gateway Engineering

Smart Gateways are protocol translation gateways at the edge of the network [26]. They allow devices, which are unable to connect to the Internet, to be available on the Internet and accessible from the Web. For instance, a smart gateway can interact with local sensor nodes via ZigBee, generate their virtual representations, and expose them on the Web as REST resources.

Similarly to the IoTSE, a large number of IoT gateways exist, and they have significant overlap regarding their internal operations. Specifically, all gateways must detect and communicate with things using ZigBee, X10, or other radio frequency (RF) protocols. All gateways must generate virtual representations of detected things based on a certain standard, such as OGC SensorThing API, and serve them as Web resources. These activities are the basis for identifying the shared components of IoT gateways, which can be built by independent component developers. We can then apply the kernel-based approach to the gateway engineering similarly to the IoTSE engineering.

We utilized the stated approach to engineer the IoT gateway that appeared in our case studies. The kernel-based approach was particularly suitable, as IoT gateways do not require complex workflows nor distributed deployment like IoTSE instances.

6.3.2 Investigating Impacts of IoTSE Architecture

IoTSE research has been focusing primarily on novel algorithms and mechanisms that can make each IoTSE component more effective and efficient. However, because IoTSE is a complex system that is made of various components, how these components interact with each other and how they are deployed on computing nodes also play significant roles in the system's performance. In other words, investigating the IoTSE architecture is a potential direction to improve IoTSE instances, which has been not addressed in the existing IoTSE literature. The case studies in Chapter 5 demonstrated this point: given the same set of components and computing nodes, changing the composition pattern of an IoTSE instance can result in nearly 50% drop in its response time.

The solutions proposed in this thesis can facilitate studies on IoTSE architecture. The reported taxonomies of composition and deployment patterns provide the constructs to model the logical and physical architecture of an IoTSE instance. The platform allows modifying different parts of an IoTSE instance independently with minimal effort. For example, a researcher can modify only the indexing component of an IoTSE instance and observe the impact on its response time. In another study, a researcher can focus on the impact of the composition and deployment pattern, as has been demonstrated in the case studies conducted in this thesis.

6.4 Future Work

In this thesis, we laid a foundation for the reuse-centric IoTSE engineering. The most notable result, we believe, was the ability to compose and deploy an operational IoTSE instance simply by providing two YAML-based documents that specify its architecture. Currently, human engineers produce these specifications by analyzing the type of query that the IoTSE instance must support and the available computing infrastructure for hosting it. Future research should focus on automating this process and can be conducted in the following directions.

6.4.1 Developing Formal and Semantic Descriptions

To enable automatic IoTSE engineering, allowing computers to understand query types, components, patterns, and the computing infrastructure involved in this process is a prerequisite. Facilitating such machine-understanding requires formal and semantic descriptions of all involving aspects. The meta-path model proposed in Chapter 2, as well as the components and patterns specified in the reference architecture can be the starting points.

6.4.2 Accumulating Architectural Knowledge

Over the course of engineering an IoTSE instance, architects make a range of decisions, such as on the type of components and architectural patterns to use to address a given type of query. The IoTSE platform can accumulate these decisions to form a knowledge base, which can be used as a “recipe book” for engineering future IoTSE instances.

This knowledge base is also a source for mining new insights on how IoTSE solutions are composed and deployed in practice. It also provides input for machines to learn and mimic the design decisions of IoTSE architects.

6.4.3 Automating IoTSE Composition and Deployment

In order to enable automatic composition and deployment of IoTSE instances, machines must generate the suitable workflows and deployment structure, and select the most suitable components to realize those workflows. Given a requested type of query and a set of computing nodes, machine-learning-based or heuristic-based approaches can be proposed to mimic the design decision of the architects and generate the architecture specifications. Techniques inspired by the Quality-of-service (QoS) service selection can be used to select the best IoTSE components among accumulated and overlapping ones.

The degree of automation can increase gradually. In the beginning, machines might only provide recommendations to human engineers. Then, machines can compose an entire IoTSE instance and wait for the approval of the engineers before deploying. Finally, when the automation is proven to be trustworthy and reliable, machines might compose and deploy IoTSE instances in real-time to adapt to the incoming queries and the state of the computing infrastructure.

By achieving this degree of automation, we will have realized a search engine for the "Library of everything" – the Internet of Things.

References

[1] Kevin Ashton. That ‘internet of things’ thing. *RFiD Journal*, 22(7):97–114, 2009.

[2] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[3] J Bleecker. A manifesto for networked objects–cohabiting with pigeons, arphids and aibos in the internet of things–why things matter what'sa blogject? what about spimes. In *Proceedings of the 13th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI.*

[4] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[5] Gerd Kortuem, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy. Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14(1):44–51, 2010.

[6] Friedemann Mattern and Christian Floerkemeier. *From the Internet of Computers to the Internet of Things*, pages 242–259. Springer, 2010.

[7] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[8] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *Internet of Things Journal, IEEE*, 1(1):22–32, 2014.

[9] Alessandro Bassi and Geir Horn. Internet of things in 2020: A roadmap for the future. *European Commission: Information Society and Media*, 2008.

[10] Charith Perera, Arkady Zaslavsky, Peter Christen, Michael Compton, and Dimitrios Georgakopoulos. Context-Aware Sensor Search, Selection and Ranking Model for Internet of Things Middleware. In *Proceedings of the 14th IEEE International Conference on Mobile Data Management (MDM)*, volume 1, pages 314–322. IEEE, 2013.

[11] Athanasios V Perera, CharithVasilakos. A knowledge-based resource discovery for internet of things. *Knowledge-Based Systems*, 109:122–136, 2016.

[12] Charith Perera, Arkady Zaslavsky, Chi Harold Liu, Michael Compton, Peter Christen, and Dimitrios Georgakopoulos. Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things. *IEEE Sensors Journal*, 14(2):406–420, 2014.

[13] Cuong Truong, Kay Romer, and Kai Chen. Fuzzy-based Sensor Search in the Web of Things. In *Proceedings of the 3rd International Conference on the Internet of Things (IOT)*, pages 127–134. IEEE, 2012.

[14] Cuong Truong, Kay Römer, and Kai Chen. Sensor Similarity Search in the Web of Things. In *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2012.

[15] Cuong Truong and Kay Römer. Content-based sensor search for the Web of Things. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, pages 2654–2660. IEEE.

[16] Zhiming Ding, Xu Gao, Limin Guo, and Qi Yang. A hybrid search engine framework for the internet of things based on spatial-temporal, value-based, and keyword-based conditions. In *In Proceedings of the IEEE International Conference on Green Computing and Communications (GreenCom), 2012*, pages 17–25. IEEE.

[17] Zhiming Ding, Zhikui Chen, and Qi Yang. IoT-SVKSearch: a real-time multimodal search engine mechanism for the internet of things. *International Journal of Communication Systems*, 27(6):871–897, 2014.

[18] Puning Zhang, Yuan-an Liu, Fan Wu, and Bihua Tang. Matching state estimation scheme for content-based sensor search in the Web of things. *International Journal of Distributed Sensor Networks*, 2015:221, 2015.

[19] Puning Zhang, Yuan-an Liu, Fan Wu, Suyan Liu, and Bihua Tang. Low-overhead and high-precision prediction model for content-based sensor search in the internet of things. *IEEE Communications Letters*, 20(4):720–723, 2016.

[20] B Maryam Elahi, Kay Romer, Benedikt Ostermaier, Michael Fahrmair, and Wolfgang Kellerer. Sensor ranking: A primitive for efficient content-based sensor search. In *Proceedings of the 2009 international conference on information processing in sensor networks*, pages 217–228. IEEE Computer Society.

[21] Benedikt Ostermaier, Kay Romer, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. A Real-Time Search Engine for the Web of Things. In *Proceedings of the 1st International Conference on the Internet of Things (IOT)*, pages 1–8. IEEE, 2010.

[22] Benoit Christophe, Vincent Verdot, and Vincent Toubiana. Searching the'Web of Things'. In *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC)*, pages 308–315. IEEE, 2011.

[23] Michael Mrissa, Lionel Médini, and Jean-Paul Jamont. Semantic Discovery and Invocation of Functionalities for the Web of Things. In *Proceedings of the 23rd IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 281–286. IEEE, 2011.

[24] Andreas Kamilaris, Koula Papakonstantinou, and Andreas Pitsillides. Exploring the Use of DNS as a Search Engine for the Web of Things. In *IEEE World Forum on Internet of Things (WF-IoT)*, pages 100–105. IEEE.

[25] Vlad Stirbu. Towards a restful plug and play experience in the web of things. In *Semantic computing, 2008 IEEE international conference on*, pages 512–517. IEEE.

[26] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*, volume 1, pages 97–129. Springer Berlin Heidelberg, 1 edition, 2011.

[27] Kay Romer, Benedikt Ostermaier, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. Real-time search for real-world entities: A survey. *Proceedings of the IEEE*, 98(11):1887–1902, 2010.

[28] Daqiang Zhang, Laurence T Yang, and Hongyu Huang. Searching in internet of things: Vision and challenges. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pages 201–206. IEEE.

[29] Sergei Evdokimov, Benjamin Fabian, Steffen Kunz, and Nina Schoenemann. Comparison of discovery service architectures for the internet of things. In *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*, pages 237–244. IEEE.

[30] Yuchao Zhou, Suparna De, Wei Wang, and Klaus Moessner. Search techniques for the web of things: A taxonomy and survey. *Sensors*, 16(5):600, 2016.

[31] Benoit Christophe, Mathieu Boussard, Monique Lu, Alain Pastor, and Vincent Toubiana. The web of things vision: Things as a service and interaction patterns. *Bell Labs Technical Journal*, 16(1):55–61, 2011.

[32] Vlad Trifa, Dominique Guinard, and David Carrera. Web thing model. *W3C Member Submission*, 2015.

[33] Simon Duquennoy, Gilles Grimaud, and Jean-Jacques Vandewalle. The web of things: interconnecting devices with high usability and performance. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages 323–330. IEEE.

[34] Anne H. H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. Iot middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal (IoT-J)*, 4(1):1–20, 2017.

[35] Kok-Kiong Yap, Vikram Srinivasan, and Mehul Motani. MAX: Human-centric Search of the Physical World. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 166–179. ACM, 2005.

[36] Mizuho Komatsuzaki, Koji Tsukada, Itiro Siio, Pertti Verronen, Mika Luimula, and Sakari Pieskä. IteMinder: finding items in a room using passive RFID tags and an autonomous robot. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 599–600. ACM.

[37] Christian Frank, Philipp Bolliger, Friedemann Mattern, and Wolfgang Kellerer. The sensor internet at work: Locating everyday items using mobile phones. *Pervasive and Mobile Computing*, 4(3):421–447, 2008.

[38] Haodong Wang, Chiu C Tan, and Qun Li. Snoogle: A Search Engine for Pervasive Environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1188 – 1202, 2010.

[39] Simon Mayer, Dominique Guinard, and Vlad Trifa. Searching in a web-based infrastructure for smart things. In *3rd International Conference on the Internet of Things (IOT), 2012*, pages 119–126. IEEE.

[40] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Proceedings of the 2007 International Conference on Mobile Data Management (MDM)*, pages 198–205. IEEE.

[41] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, and Howard Morris. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[42] Lina Yao and Quan Z Sheng. Correlation discovery in web of things. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 215–216. International World Wide Web Conferences Steering Committee.

[43] Yongrui Qin, Quan Z Sheng, Nickolas JG Falkner, Schahram Dustdar, Hua Wang, and Athanasios V Vasilakos. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications*, 64:137–153, 2016.

[44] N. Koshizuka and K. Sakamura. Ubiquitous id: Standards for ubiquitous computing and the internet of things. *Pervasive Computing, IEEE*, 9(4):98–101, 2010.

[45] Tingxin Yan, Deepak Ganesan, and R Manmatha. Distributed image search in camera sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 155–168. ACM.

[46] Chiu C Tan, Bo Sheng, Haodong Wang, and Qun Li. Microsearch: When search engines meet small devices. In *Proceedings of the 6th International Conference on Pervasive Computing*, pages 93–110. Springer-Verlag Berlin, 2008.

[47] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *IEEE multimedia*, (4):8–13, 2007.

[48] Michele Ruta, Eugenio Di Sciascio, Giacomo Piscitelli, and Floriano Scioscia. A ubiquitous knowledge-based system to enable RFID object discovery in smart environments. *Pacific Asia Journal of the Association for Information Systems*, 2(3), 2010.

[49] Simon Mayer and Dominique Guinard. An Extensible Discovery Service for Smart Things. In *Proceedings of the Second International Workshop on Web of Things*, pages 1–6. ACM, 2011.

[50] Mathieu Boussard, Benoit Christophe, Olivier Le Berre, and Vincent Toubiana. Providing user support in web-of-things enabled smart spaces. In *Proceedings of the Second International Workshop on Web of Things*, page 11. ACM.

[51] Sujith Samuel Mathew, Yacine Atif, Quan Z Sheng, and Zakaria Maamar. Web of things: Description, discovery and integration. In *International Conference on Internet of Things (iThings/CPSCom), 2011 and 4th International Conference on Cyber, Physical and Social Computing*, pages 9–15. IEEE.

[52] Qiang Wei and Zhi Jin. Service discovery for Internet of Things: A context-awareness perspective. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, page 25. ACM.

[53] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32, 2012.

[54] Darren Carlson and Andreas Schrader. Ambient Ocean: A Web Search Engine for Context-Aware Smart Resource Discovery. In *IEEE International Conference on Internet of Things (iThings), 2014*, pages 177–184. IEEE.

[55] Jonas Michel, Christine Julien, and Jamie Payton. Gander: Mobile, pervasive search of the here and now in the here and now. *IEEE Internet of Things Journal*, 1(5):483–496, 2014.

[56] Mohammad Ebrahimi, Elaheh Shafieibavani, Raymond K Wong, and Chi-Hung Chi. A New Meta-heuristic Approach for Efficient Search in the Internet of Things. In *Proceedings of the 2015 IEEE International Conference on Services Computing (SCC)*, pages 264–270. IEEE.

[57] Porfirio Gomes, Everton Cavalcante, Taniro Rodrigues, Thais Batista, Flavia C Delicato, and Paulo F Pires. A Federated Discovery Service for the Internet of Things. In *The 2nd Workshop on Middleware for Context-Aware Applications in the IoT*, pages 25–30. ACM.

[58] Ali Shemshadi, Lina Yao, Yongrui Qin, Quan Z Sheng, and Yihong Zhang. ECS: A Framework for Diversified and Relevant Search in the Internet of Things. In *Proceedings of the 16th International Conference on Web Information Systems Engineering (WISE)*, pages 448–462. Springer.

[59] Francesco Renna, Joseph Doyle, Vasileios Giotsas, and Yiannis Andreopoulos. Query processing for the internet-of-things: Coupling of device energy consumption and cloud infrastructure billing. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 83–94. IEEE, 2016.

[60] Yuanyi Chen, Jingyu Zhou, and Minyi Guo. A context-aware search system for Internet of Things based on hierarchical context model. *Telecommunication Systems*, 62(1):77–91, 2016.

[61] Luiz Nunes, Julio Estrella, Rafael Nakamura, Luisde Libardi, Carlos Ferreira, Liuri Jorge, Charith Perera, and Stephan Reiff-Marganiec. A distributed sensor data search platform for internet of things environments. *International Journal of Services Computing*, 4(1), 2016.

[62] Ali Shemshadi, Quan Z. Sheng, and Yongrui Qin. ThingSeek: A crawler and search engine for the internet of things. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1149–1152. ACM, 2016.

[63] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 542–553. ACM.

[64] W3C. Web of things - technology landscape. *W3C Member Submission*, 2016.

[65] BRIDGE. WP02 - High Level Design for Discovery Services. *BRIDGE Project*, 2007.

[66] Afilias. Afilias - Finding your Way in the Internet of Things. *An Afilias Whitepaper*, 2008.

[67] Ali Rezafard. Extensible Supply-chain Discovery Service Problem Statement. *IETF Internet-Draft*, 2008.

[68] Ali Shemshadi, Quan Z Sheng, Wei Emma Zhang, Aixin Sun, Yongrui Qin, and Lina Yao. Searching for the Internet of Things on the Web: Where It Is and What It Looks Like. *arXiv preprint arXiv:1607.06884*, 2016.

[69] Wu Qihui, Ding Guoru, Xu Yuhua, Feng Shuo, Du Zhiyong, Wang Jinlong, and Long Keping. Cognitive internet of things: A new paradigm beyond connection. *Internet of Things Journal, IEEE*, 1(2):129–143, 2014.

[70] A. M. Ortiz, D. Hussein, Park Soochang, S. N. Han, and N. Crespi. The cluster between internet of things and social networks: Review and research challenges. *Internet of Things Journal, IEEE*, 1(3):206–215, 2014.

[71] Chen Shanzhi, Xu Hui, Liu Dake, Hu Bo, and Wang Hucheng. A vision of iot: Applications, challenges, and opportunities with china perspective. *Internet of Things Journal, IEEE*, 1(4):349–359, 2014.

[72] ISO/IEC. ISO/IEC WD 18384-2: Distributed Application Platforms and Services (DAPS) - Reference Architecture for Service Oriented Architecture (SOA RA) Part 2. 2016.

[73] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. S3: A service-oriented reference architecture. *IT professional*, 9(3), 2007.

[74] Gerrit Muller. A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper*, 2008.

[75] Barbara Ann Kitchenham and Stuart M. Charters. *Guidelines for performing systematic literature reviews in software engineering*. 2007.

[76] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. Mobile cloud business process management system for the internet of things: A survey. *ACM Computing Surveys (CSUR)*, 49(4):1–42, 2016.

[77] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber. A vision of swarmlets. *IEEE Internet Computing*, 19(2):20–28, 2015.

[78] ISO/IEC. ISO/IEC WD 18384-1: Distributed Application Platforms and Services (DAPS) - Reference Architecture for Service Oriented Architecture (SOA RA) Part 1. 2016.

[79] Qiang He, Rui Zhou, Xuyun Zhang, Yanchun Wang, Dayong Ye, Feifei Chen, John C Grundy, and Yun Yang. Keyword search for building service-based systems. *IEEE Transactions on Software Engineering*, 43(7):658–674, 2017.

[80] Elisa Yumi Nakagawa, Flavio Oquendo, and Martin Becker. Ramodel: A reference model for reference architectures. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pages 297–301. IEEE.

[81] Jeffrey R Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh C Jain, and Chiao-Fe Shu. Virage image search engine: An open framework for image management. In *Proceedings of Storage and Retrieval for Image and Video Databases (SPIE)*, volume 1996, pages 76–87.

[82] Neal G. Shaw, Ahmad Mian, and Surya B. Yadav. A comprehensive agent-based architecture for intelligent information retrieval in a distributed heterogeneous environment. *Decision Support Systems*, 32(4):401–415, 2002.

[83] Karl Aberer, Fabius Klemm, Martin Rajman, and Jie Wu. An architecture for peer-to-peer information retrieval. In *Proceedings of the Workshop on Peer-to-Peer Information Retrieval in the 27th Annual International ACM SIGIR Conference (SIGIR 2004)*.

[84] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, 1031289. ACM.

[85] Huajing Li, Isaac Councill, Wang-Chien Lee, and C. Lee Giles. Citeseerx: an architecture and web service design for an academic document search engine. In *Proceedings of the 15th international conference on World Wide Web*, pages 883–884, 1135926. ACM.

[86] Michael D. Lieberman, Hanan Samet, Jagan Sankaranarayanan, and Jon Sperling. Steward: architecture of a spatio-textual search engine. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pages 1–8, 1341045. ACM.

[87] Damon Horowitz and Sepandar D. Kamvar. The anatomy of a large-scale social search engine. In *Proceedings of the 19th international conference on World wide web*, pages 431–440, 1772735. ACM.

[88] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012.

[89] Tingting Liang, Liang Chen, Jian Wu, Hai Dong, and Athman Bouguettaya. *Meta-Path Based Service Recommendation in Heterogeneous Information Networks*, pages 371–386. Springer International Publishing, Cham, 2016.

[90] Quan Z. Sheng, Yongrui Qin, Lina Yao, and Boualem Benatallah. *Managing the Web of Things: Linking the Real World to the Web*, volume 1. Morgan Kaufmann, 2017.

[91] EPCglobal. The epcglobal architecture framework. *EPCglobal Ratified specification*, 2005.

[92] Nguyen Khoi Tran, Quan Z Sheng, Muhammad Ali Babar, and Lina Yao. Searching the Web of Things: State of the art, challenges and solutions. *ACM Computing Surveys (CSUR)*, 50(4):1–34, 2017.

[93] Nguyen Khoi Tran, Quan Z Sheng, Muhammad Ali Babar, and Lina Yao. A kernel-based approach to developing adaptable and reusable sensor retrieval systems for the web of things. In *Proceedings of the 18th International Conference on Web Information Systems Engineering*, 2017.

[94] Wei Gao and Jian Wu. A novel framework for service set recommendation in mashup creation. In *Web Services (ICWS), 2017 IEEE International Conference on*, pages 65–72. IEEE.

[95] Tian Huat Tan, Manman Chen, Jun Sun, Yang Liu, Étienne André, Yinxing Xue, and Jin Song Dong. Optimizing selection of competing services with probabilistic hierarchical refinement. In *Proceedings of the 38th International Conference on Software Engineering*, pages 85–95. ACM.

[96] Ferdian Thung, Shaowei Wang, David Lo, and Julia Lawall. Automatic recommendation of api methods from feature requests. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 290–300. IEEE Press.

[97] Fabienne Boyer, Olivier Gruber, and Damien Pous. Robust reconfigurations of component assemblies. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 13–22. IEEE Press.

[98] Antonio Bucchiarone, Martina De Sanctis, Annapaola Marconi, Marco Pistore, and Paolo Traverso. Incremental composition for adaptive by-design service based systems. In *Web Services (ICWS), 2016 IEEE International Conference on*, pages 236–243. IEEE.

[99] Pradeeban Kathiravelu, Tihana Galinac Grbac, and Luís Veiga. Building blocks of mayan: Componentizing the escience workflows through software-defined service composition. In *Web Services (ICWS), 2016 IEEE International Conference on*, pages 372–379. IEEE.

[100] Paola Inverardi and Massimo Tivoli. Automatic synthesis of modular connectors via composition of protocol mediation patterns. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 3–12. IEEE Press.

[101] Sheikh Mohammed Sohan, Craig Anslow, and Frank Maurer. Spyrest: Automated restful api documentation using an http proxy server (n). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, pages 271–276. IEEE.