A STATE-BASED APPROACH TO CONTEXT MODELING AND COMPUTING

by

SONGHUI YUE

RANDY SMITH, COMMITTEE CHAIR
TRAVIS LEVESTIS ATKISON
JEFFREY CARVER
ANDREW GRAETTINGER
JEFF GRAY

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2019

ABSTRACT

Context-aware computing is one of the most essential computing paradigms in pervasive computing. However, current context-aware computing is still in lack of good representation models, particularly in modeling proactive behaviors and historical context data. For context-aware computing, explicitly putting forward states of high-level context can be beneficial and intrigue new angles of understanding and modeling activities. In this dissertation, I propose a state-based context model, and based on the model, I introduce Context State Machines (CSM) for simulating state changes of context attribute, situation, and context, which imply important behaviors of related to context.

This research develops and demonstrates CSMs for known context-aware problems from the literature including a smart elevator control system. First of all, the smart elevator, as a context-aware application in the literature, is introduced. Secondly, I introduce the implementation of the CSM engine. Thirdly, I describe two context-aware scenarios and show the model can help automatically capture the contexts and reason the context without the inference from the developers. It is the first time in literature to apply state-based modeling approach and the CSM engine to a real-world context-aware system.

To evaluate the CSM engine as well as the CSM modeling approach, I generate high-level contextual testing data to feed the engine. I surveyed the data quality issues regarding context-aware software and rubrics of the data quality and dimensionality are developed to

address the challenges of applying context to context-aware systems. The rubrics are applied in the generation of synthetic data for feeding the CSM engine in this dissertation.

DEDICATION

This thesis is dedicated to everyone who helped me and guided me in the life of the PhD, and through the trials and tribulations of creating this manuscript. In particular, my family, advisor, and friends who stood by me throughout the time taken to complete this dissertation.

LIST OF ABBREVIATIONS AND SYMBOLS

ADS	Autonomous-Driving System
BN	Bayesian Network
CAS	Context-Aware Software
CASM	Context Attribute State Machine
CDC	Continuous Data Collection
CMSM	Context Mealy State Machine
CSM	Context State Machine
CSSM	Context Situation State Machine
CTU	Control Tuning Unit
ERS	Emergency Response Analysis
GA	Genetic Algorithms
GCU	Group Control Unit
HMM	Hidden Markov Model
IoT	Internet of Things

OO	Objected-Oriented
ORDBMS	Object Relational Database Management Systems
OWL	Web Ontology Language
POI	Places of Interest
RDF	Resource Description Framework
UC	Ubiquitous Computing

ACKNOWLEDGMENTS

I am pleased to have this opportunity to thank the many colleagues, friends, and faculty members who have helped me during the whole period of my graduate study. First and foremost, I would like to show my gratitude and appreciation to my advisor, Dr. Randy Smith, for providing me the opportunity to complete my Ph.D. study at the University of Alabama. Without Dr. Smith's continuous support and encouragement, this work is impossible.

I would also like to extend my gratitude to the other members of my committee. To Dr. Jeff Gray, Dr. Jeffrey Carver, Dr. Travis Levestis Atkison, and Dr. Andrew Graettinger, I feel grateful for your efforts to help me develop the necessary knowledge and skills throughout the stages of my graduate study. I really appreciate your precious time and effort in serving on my committee.

To my fellow coworkers at UA, Huseyin Ergin, Kuai Meng, Xin Zhao, Saheed Popoola, Miao Xie, Xue Wu, Jyothsna Kondari, and Songqing Yue. I really appreciate your friendship and all the wonderful and fun times we shared. To all my colleagues in the graduate school at UA, Jason Phillips, Hengxia Zhao, Michael Fox, Coston Perkins, Julianna Proctor, Lorne Kuffel, and Ashirul Mubin, I really enjoyed working with you as a team to provide professional service for the university management. I will always be grateful to the financial support from the graduate school.

This dissertation would not have been possible without the support of my friends and my family who never stopped encouraging me to persist. Finally, I want to express my gratitude to my dear wife, Yang Shao, and because of her accompany, I can accomplish this work with

happiness and joy. My creator and savior Jesus Christ always help and love through the people surrounding me and teach me the truth of life through the valleys and mountains.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND MOTIVATION

Modern electronic devices are very powerful in both computing and obtaining information from the environment. Many new devices employ multi-core processors along with technological advances in networked computing. Particularly, a modern smart phone can be equipped with as many as fourteen sensors [36], such as a proximity sensor, ambient light sensor, accelerometer, magnetometer, and gyroscopic sensor. The extra computing and information processing abilities of modern devices make Context-aware Software (CAS) and Context-Aware Computing (CAC) more and more important to Ubiquitous Computing (UC) and Internet of Things (IoT) paradigms [18] [31] [55]. As a result, a large variety of information can be used as *context* to enrich the functionality of software applications.

To represent the contextual data from heterogeneous sources, in another word, to model context, is a challenging task [6] [10] [35] [56]. Context reasoning, as a process to extract machine-understandable or higher-level contextual information, is one of the most important aspects of the context modeling phase. Context reasoning usually is necessary given the large amount of lower-level context information. A context modeling approach supporting context reasoning will greatly aid the context understanding of CAS. In the following subsections, the concepts of context modeling, context reasoning and the state-based modeling approach are introduced. The motivation of this research is illustrated. Finally, the research questions addressed in this research are given.

1.1 Context Modeling

Computer modeling is to *"construct and manipulate abstract (mathematical or graphical) representations of economic, engineering, manufacturing, social and other types of situations and natural phenomenon"* [14]. Since computer modeling usually is related with computer programs – writing computer programs version of a mathematical or graphical model, computer modeling is also referred as computer simulation. Context modeling, as a branch of computer modeling, is defined as the context representation that provides assistance in the understanding of properties, relationship, and details of context [55]. Context modeling includes corresponding computer program realization of the mathematical and/or graphical representations of context. Thus, context modeling in our research domain consists of two parts: 1) a mathematical or graphical representation of context; 2) a program realization of the representation. Theoretically, there could be different realizations for one type of representation.

1.2 Context Reasoning

Context reasoning is a process to extract or deduce new knowledge or context for context-aware applications to use or for better understanding from lower-level context or high-level context [55]. According to Nurmi et al. [58], there are three tasks related to context reasoning: 1) context preprocessing; 2) sensor fusion; and 3) context inference. In the context preprocessing, relevant context attributes are recognized to make later processing easier. Missing attributes can be handled, and the data can be cleaned in this step. The sensor fusion task aims at integrating data from multiple sources. Context inference recognizes new context by processing context and mapping low level context to higher-level context.

1.3 Calling for a State-based modeling approach

Context-aware computing enables context-aware applications to perceive the context information so as to dynamically adapt their behaviors for the benefit of users without users' explicit intervention. As a large number of advanced sensors are being deployed and large amounts of data with complex interrelationships being generated, context comes in multiple forms. There is a form of secondary context [18] or high-level context as a result of storing, synthesizing (reasoning) and sharing useful context data from lower-level sensor data [65]. A good context modeling formalism can be crucial to improving the accuracy of expressing the essence of context and facilitating context reasoning. A good context modeling formalism reduces the complexity of context-aware systems and improves their maintainability and evolvability.

However, current context-aware computing is still lacking good representation models. Khattak et al. [6] in their review work proposed that both *context representation* and *context fusion* are underestimated by most of the context-aware systems and should be explicitly addressed in design and development processes. One reason lies in that research on context-aware systems has focused on reactive behaviors rather than proactive ones that could facilitate proactive decision-making [2]. As a result, most of the current context modeling approaches, such as ontology and object-oriented methods, focus on the representation of the current context, like the current values of location (e.g. Home, Office, Gym), timestamp, activity and the relationship between them [6] [7]. Historical context modeling, such as the modeling of the learning results from the historical context data and the influences among the values of each context attribute—which can imply various meaningful behaviors (e.g. returning to home from campus)—which in our understanding are also important forms of context, is neglected.

State diagrams are usually used for modeling behaviors of a system, and one classic form is a Finite State Machine (FSM). A well-known study using state diagrams is Harel's *statecharts* [22], which present a visual formalism for modeling complex systems. In order to model the interrelationships among the values of current context and historical context (behaviors), leveraging Harel's work, this research presents a state-based modeling method to represent high-level context. I argue this approach serves as a complement to existing modeling methods to model the behaviors of context and empower context reasoning and prediction by putting forward the state of context. To begin with, states of context can be a representation of a basic element of context. Secondly, state transition can be used to represent potential underlying relationships, which themselves are forms of context. Thirdly, states and state transitions can be treated as a computing axiom, where interoperability, prediction and imprecision can be resolved on the state level.

1.4 Research Questions

As discussed in the context modeling subsection, it requires a modeling approach to consider both representation and computing/programming part simultaneously. The following researching questions are addressed in this research.

Research Question 1

RQ1) Can states be used for modeling context?

In order to address Research Question 1, this dissertation will provide a state-based context model and build a mealy state machine style model – called a Context State Machine (CSM) capable of modeling simple and complex context situations with contextual data involving multiple entities and constraints.

Research Question 2

RQ2) Can CSM model for context-aware software development be used to demonstrate effectiveness for known context-aware problems from the literature?

In order to address Research Question 2, this dissertation develops the CSM model for a basic situation of an elevator control system with limited contextual entities and constraints.

Research Question 3

RQ3) Can CSM model for context-aware software development scale to multiple entities and constraints?

In order to address Research Question 3, this dissertation extends the elementary situation from Research Question 2 to a complex situation of the elevator control system with multiple contextual entities and constraints.

Research Question 4

RQ4) Can synthesized data for context aware systems be developed addressing the inaccuracies and poor quality of data expected from real-world sensors?

In order to address Research Question 4, data quality and data dimensionality rubrics are developed for context data in general. The rubrics are applied in the generation of synthetic data for the elevator control contexts in this dissertation.

1.5 Outline of the Dissertation

The organization of this dissertation document consists of a general introduction, the concepts of context and context modeling, the main contribution of "a state-based approach to context modeling and computing", and the application of the modelling approach to a smart context-aware elevator.

Chapter 2 introduces related concepts, including a discussion of existing modeling methods and context reasoning techniques. Chapter 3 describes our work in a stated-based context modeling approach which is capable of modeling simple and complex context and situations. Chapter 4 describes a context-aware elevator system's two scenarios for using CSM method to model context with multiple entities and constraints. Chapter 5 describes the CSM engine implementation and evaluation methodology and results. Chapter 6 introduces the requirement of a synthesized data that is used for context-aware systems and describes the process of generation of the data. Chapter 7 is a conclusion and future work. Appendix A is the implementation details of CSM engine.

CHAPTER 2

CONTEXT AND CONTEXT MODELING

This section provides an overview of the terminology and techniques prominent in context modeling and computing. The basic concepts of context, context-aware, context modeling, context reasoning and the related modeling techniques are described and discussed.

2.1 Context and Context-Aware

The context definitions given by researchers are slightly different from each other because of their different understanding or application of the term. Schilit and Theimer [11] first introduced "context-aware" in their work and defined context as location, identities of nearby people and objects and changes to those objects (1994). Brown [13] defined context as a combination of elements of the user's environment that the computer knows about (1996). Dey et al. [4] defined context as the user information and user's changing location, the changing objects in the environment, and the familiarity with the environment (1998).

Based on all the prior attempts to define context, Dey & Abowd (2000) [30] provided a comprehensive definition of context which is often cited by most of the current related studies as "*any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects.*" They also provided

a widely known definition for the term context-awareness as *"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

Context-aware software can be broadly defined as any software that applies data analysis for better decision making, while the data analysis is for context recognizing (modeling) and reasoning.

2.2 Context Data

Context data is used by applications to add context-aware features, which can be generated by heterogeneous sources. In a context-aware application, context data can be retrieved with the assistance of hardware (real sensors) or software (virtual sensors). For location-based context-aware software, context information contains discrete data to mark the locations, which are usually derived from the hardware level [49] [79]. Sensors are widely utilized to capture changing contextual data and then pass them to the software [27] [71]. Context data may also come from the software level. For instance, contextual information can be collected from other applications running in the same or related devices [27] [48].

Context does not only include data of the current status, it also includes context data generated from the history data. There are different ways of categorizing context, such as Physical/Virtual and Direct/Indirect [44]. I divide context into three categories, as shown in Figure 1.

The three categories are as follows:

***World Context*:**

Current world context represents the current context outside the device. It could include context data being generated from sensors, devices and other sources from anywhere outside the device.

Historical world context represents the historical context outside the device. It could include context data that generated by learning algorithms from historic data of sensors, devices and other sources anywhere outside the device.

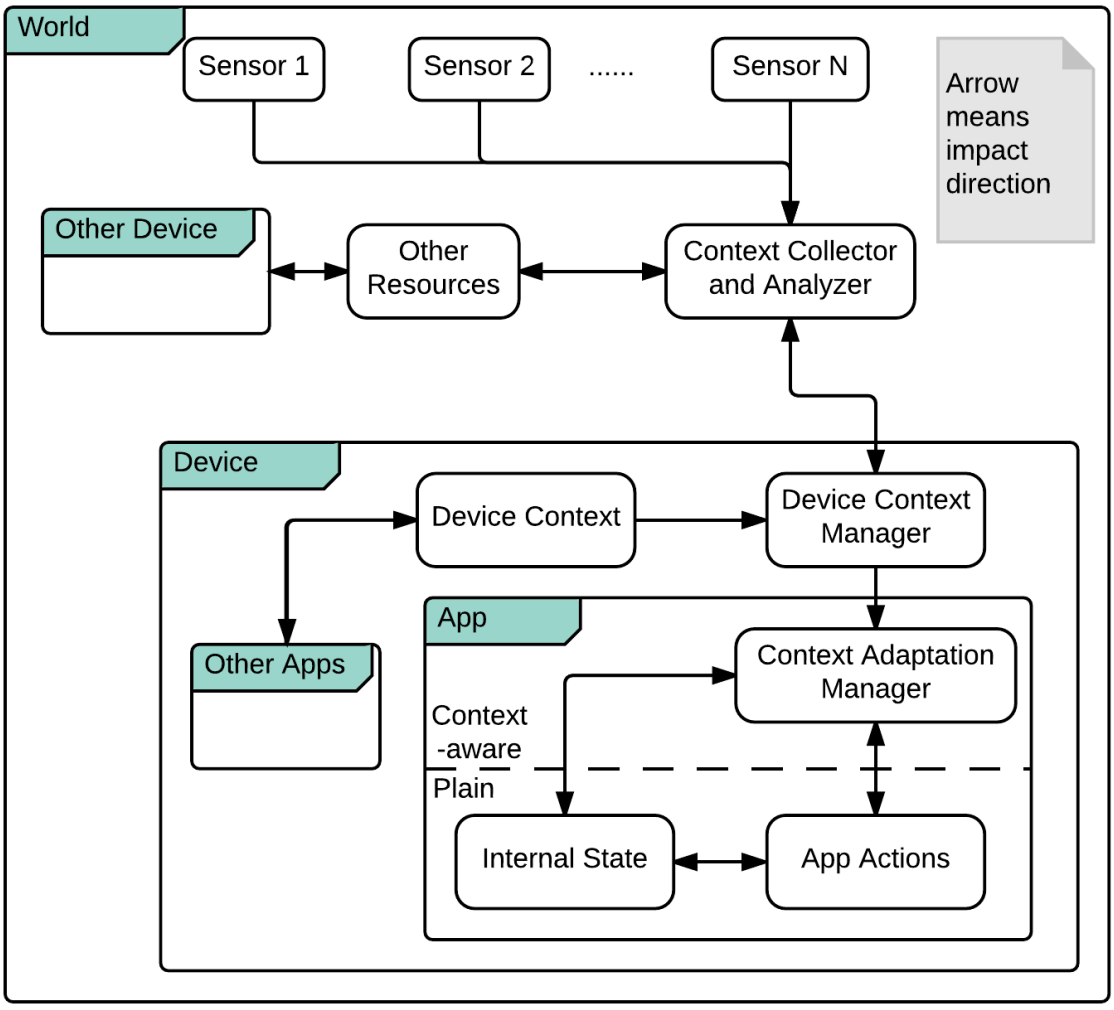

Figure 1. Context Division by Categories

***Device Context*:**

Current device context represents the current context inside the device but outside the application. It could include context data being generated from sensors, other applications and current status of the device.

Historical device context represents the historical context inside the device but outside the application. It could include context data that generated by learning algorithms from historic data of sensors, other applications and status of device.

***Application Context*:**

Current application context represents the current context within the application. It could include the current context-aware adaptation status, plain program logic status, and system behaviors.

Historical application context represents the historical context within the application. It could include context data that generated by learning algorithms from historic data of context-aware adaptation status, plain program logic status, and system behaviors.

In Figure 1, *Context Collector and Analyzer* collects the lower-level context from different sensors or collects lower-level or higher-level context from other resources, then it generates higher-level contextual data, which can be used by a Device *Context Manager* deployed in a smart device.

The *Device Context Manager* module receives context from outside sources and collects context from internal applications and the device status. The *Device Context Manager* analyzes or reasons about context and disseminates context to context-aware applications.

The *Context Adaptation Manager* in a context-aware application is a program module to reason about context and to adapt the behaviors of the application depending on the current context.

2.3 Context Sharing and Interoperability

Context sharing is necessary when combining multiple context-aware services to a composite service and when one context-aware service needs to use the context from other services. The concept of context sharing is highly related to interoperability. The IEEE Standard Computer Dictionary defines interoperability as "*the ability of two or more systems or components to exchange information and to use the information that has been exchanged.*" [37] Two aspects in context interoperability are observed:

- Two or more context-aware services can interact with each other and exchange context information.
- Different context-aware services can make use of exchanged context information.

To better achieve context sharing and fulfill the requirement of interoperability, pre-designed context modeling approaches should be applied. In the next section, different approaches of context-modeling are introduced, some of which support context sharing and interoperability better than others.

2.4 Context-modeling Techniques

Context modeling is the core of context processing supporting different context reasoning, context sharing, and semantic interoperability of heterogeneous systems [6]. There are several proposed approaches and each of them has their advantages and limitations. In this section, I survey several classical context modeling techniques, including key-value, markup, logic-based, ontology-based, object-oriented, and graphical modeling. Additional context

modeling approaches not applicable to this work include multidisciplinary, domain-focused, user-centric, and chemistry inspired modeling [44], as they are not built from a system's perspective or less broadly applied.

2.4.1 Key-value Modeling and Markup Modeling

Key-value pairs approach uses keys and values to represent attributes and their values. Markup is referred to as tagged encoding, and context information is stored within tags and in a format of typical markup languages such as the eXtensible Markup Language (XML). These modeling methods are mainly used in early research and services.

2.4.2 Logic-based Modeling

With logic-based modeling, facts, expressions, and rules are used to represent information about the context [18]. Rules enhance logic-based modeling with the reasoning capability and expressive richness on policies, constraints, and preferences, so that it can work as a supplement for other modeling techniques. However, the complex of polices, constraints and preferences could result in lack of standardization and further reduce the re-usability and applicability.

2.4.3 Ontology-based Modeling

Ontology concerns the nature and relationships of beings. An ontology modeling approach is used to represent the concept of entities within the context domain and their inter-relationships. This modeling approach provides rich expressiveness when modeling context and it also supports reasoning and is propitious to interoperability. Therefore, ontology-base modeling is considered as one of the most promising methods [81]. Nguyen et al. [72] use ontology to model context information for a smart home.

2.4.4 Object-Oriented Modeling

Object-oriented is a classical paradigm for programming languages and system design. When being used for programming languages, it is defined by the concepts of inheritance, encapsulation, and polymorphism. In the area of context modeling, the object-oriented model employs class hierarchies and relationships to represent context data and incorporates encapsulation, inheritance and reusability into context expression [19]. Bhogal et al. [38] explore the use of Object Relational Database Management Systems (ORDBMS) to model context and conclude that the use of ORDBMS to implement context modeling offers benefits over ontology-based context modeling.

2.4.5 Graphical

Graph-based modeling approach is approved to be efficient for modeling of semantics [8], huge archives [80], and image-dependent contextual relationships [32]. Chihani et al. [8] use this approach to overcome the disadvantage of other modeling approaches of constraining application to share specific semantic for the modeled context data. Their graph is essentially related with an ontology model of the context domain and the reasoning relies on the link types between the entities. Wu et al. [80] define the context graph for recommender system by treating users, items, attributes, and contexts as vertex and the connection between them as the edges, so that the edges can represent the relationship between different vertexes. Myeong et al. [32] extract the contextual information from images by introducing a context link view of contextual knowledge and utilize the k-nearest neighbor similarity graph to model the links between the images. Their experiment on images' object class segmentation outperforms the current state-of-the art methods.

2.4.6 Discussion

A detailed comparison of existing context modeling techniques is presented in Table 1 [44] below. I can conclude that no individual modeling approach is perfect for solving all challenges. Each kind of modeling approach may belong to a category, and all approaches inside the category are specialized to solve one aspect or challenge of modeling context, such as graphical modeling approach is designed for rich expressiveness. A better approach would be for an implementation in a compositional way, for example, a composition that includes markup, OO, ontology and state-based techniques. Different domains or applications may apply one or several approaches, depending on their unique requirements.

Table 1. Comparison of existing context modelling techniques: Adapted From [44]

Context Modelling Technique	Advantages	Disadvantages
Key-value	Simple; Ease of use; Flexible	Lack of standards; Useless when big in size; Cannot represent relationships; Difficult to retrieve information; Lack of validation tools; Lack of scalability; Only exact matching.
Markup	Structured; Some validation tools are available; Flexible.	Lack of standards; Problems in capturing relationships; Timeless; Dependencies; Inconsistency checking; Reasoning and Uncertainty.
Graphical	Rich expressiveness; Relationships are allowed; Validation is possible through constraints; Different standards and implementations are available.	Interoperability is unsolved; Configuration must be required; A generic and well-developed standard is needed.
Object-oriented	Relationships are allowed; Some development tools are available; Can be fused by using programming languages.	Lack of standards; Lack of validation; Hard to retrieve information; Reasoning is not supported.
Logic-based	Rich expressiveness; Support reasoning; Consistency check; Simplicity; Processing tools are available.	Lack of standards; Lack of validation.
Multidisciplinary	Comprehensive understanding for context based on multiple disciplines; The division of context is concrete.	Too complex; Still at the first stage; Interoperability is unsolved.
Domain-focused	Expressive; Flexible; Structured.	Lack of standards; Lack of validation.
User-centric	Express context in an organized way; Scalability; Allow reasoning.	Lack of standards; Complex to use; Lack of validation; Lack of formality.
Ontology-based	Support reasoning; Rich expressiveness; Relationships are allowed; Strong validation; Processing tools available; Mature standards; Interoperability.	Representation can be complicated; It will be complex to retrieve context information; Unable to address uncertainty.
Chemistry inspired	Medium expressivity to represent many kinds of context; Support for triggering services autonomously; Cross-domain inspired.	Lack of standards; Lack of validation; Not dynamic and scalable; In a nascent stage.

2.5 Context Reasoning Techniques

Context reasoning is a process to extract or deduce new knowledge or context for context-aware applications [20]. Further, context reasoning is used for better understanding from lower-level context or high-level context. Table 2 shows the mapping between the context type and their related most-usually-used reasoning approaches. For context computing and modeling, context reasoning is one of the most important terms. On one hand, context modeling should support context reasoning; on the other hand, context reasoning is one of the most important requirements the context modeling phase should consider.

Table 2. Reasoning Techniques for Different Type of Context: Adapted From [12]

Context	Reasoning Approaches
Social Context (Who is nearby? What's their availability? How can I reach them?)	Logic, Rules
Availability (Busy, Free)	Rules, Decision Tree
Proximity (close, near, far)	Fuzzy Logic
Activity (Driving, Exercising, Sitting, Walling, Running, Sleeping, Eating, Talking)	HMM, Decision Tree, Bayesian Network, DSET, Discriminant Analysis
Mobility (in moving vehicle)	HMM, Dynamic Bayesian Network
High Level Identity (family member, coworker, neighbor, friend, unidentified)	App specific Social Ontology, Rules
High Level Location (work, school, home, store, library, post office)	Bayesian network, Decision Tree, Logic, Rules
Low Level Identity (Tom, John, MID)	App specific DB Lookup
Low Level Location (GPS coordinates)	No reasoning/Direct from device

Context reasoning techniques are usually based on more traditional logic methods or rules such as fuzzy logic and probabilistic logic. The ontology-based context reasoning is related with the ontology-based context modeling approach. Machine learning techniques can also be used as methods for context reasoning. Nyugen et al. [54] use a graph-based approach to conduct context reasoning.

2.5.1 Fuzzy Logic, Probabilistic Logic, Rules

Fuzzy logic considers that partial truth is acceptable. As a reasoning method, it is frequently used along with techniques such as ontological or probabilistic approaches. Probabilistic approaches calculate the probabilities of events and facts and then make decisions. Dempster-Shafer and Hidden Markov Models (HMMs) are most frequently used methods in context awareness of this category [55]. For rules reasoning, the output can be acquired with an if-else structure.

2.5.2 Ontology-based

Ontology-based context reasoning approaches typically use Resource Description Framework (RDF) and Web Ontology Language (OWL) to build ontology models, and then a set of reasoning rules defined to reason on them [78]. This requires the context being modeled into ontologies and ontology model at first. Besides combined with rules reasoning, other reasoning techniques such as fuzzy logic and probabilistic logic can be used.

2.5.3 Supervised Learning and Unsupervised Learning

Contextual data are firstly collected and then could be reasoned using supervised or unsupervised learning techniques. For supervised learning, context data is firstly divided into training data and testing data. The training data should be labeled first for the model generation. And the testing data will be used to test the model or find the best algorithms [64]. The most

widely used supervised learning algorithms includes decision trees, naïve Bayes, and K-nearest neighbor algorithm etc. For unsupervised learning, clustering is used to extract results with meaningful categorization. The most widely used unsupervised learning algorithms includes K-means, hierarchical clustering, and unsupervised neural networks.

2.5.4 Graph-based

Nguyen et al. [54] use contextual graph to conduct context reasoning. They use perceived context to build each node/action or path in a contextual graph. Their graph is not systematically defined and depends on the design for each individual application or domain.

2.5.5 Discussion

Context reasoning, inference models and inference techniques differ with context modeling techniques. Context modeling is for the representation of the data while context reasoning is for computing and context-aware software decision making. As shown in the literature in this subsection, the various approaches for context reasoning are ultimately based on logics and rules, even for the machine learning methods. Other reasoning approaches through modeling is to model the data first so as to facilitate the use of rules or probabilistic methods, such as the ontology-based and the graph-based. methods Our proposed state-based context modeling approach supports context reasoning by modeling the context using states and then combine the techniques of probabilistic logic, supervised learning, and unsupervised learning.

2.6 Context-aware Middleware

Context-aware middleware is widely used for facilitating development and execution of context-aware software [51] [68]. Middleware refers to software systems, which provide an abstraction and mechanisms between the network operating system layer and the applications layer [42] [45]. Researchers have developed various middleware systems for building and

rapidly prototyping context-aware services [59] [73]. As the work in [24] suggests, typical middleware architecture for developing context-aware software contains two key components: the context manager and the adaptation manager (presented in Figure 1). The context manager captures and manages context from surroundings and pushes the context changes to the adaptation manager. The Context Adaptation Manager is responsible for reasoning on the impact of context changes and then choosing proper reactions for applications behaviors.

CHAPTER 3

A STATE-BASED APPROACH – CONTEXT STATE MACHINE

In order to address Research Question 1, a state-based approach to context modeling is introduced in this chapter to show the feasibility of using states to model context and help the development of context-aware software by using a context-aware system for monitoring experiment tables in a smart campus project. This chapter includes four parts and a summary. Firstly, related work about context modeling and state modeling is introduced. Secondly, an extended context understanding is introduced to show the necessary of the state-based approach to model the dynamic context and their internal relationships. Context is modeled using state as the basic component and further, state is introduced in the definitions of context attribute and context situation. Based on the model, I have proposed Context Mealy State Machine (CMSM or CSM) to model context states and their interrelationships. I also describe a context-aware system where our approach is used to model the context behaviors and solve prediction problems.

3. 1 Related Work about Context Modeling and States Modeling

For good context representation, context is usually defined in a format that can be easily stored, accessed, and exchanged [7]. Xin et al. [44] summarize ten modeling techniques researchers have been using including “key-value,” “Markup,” “Graphical,” “Object-oriented,” “Logic-based,” “Multidisciplinary,” “Domain-focused,” “User-centric,” “Ontology-based” and

"Chemistry-inspired." Bettini et al. [10] recognize three prominent approaches to context modeling: object-role based, spatial and ontology-based.

Ontology-based modeling approaches are widely used to model context details. For example, Hong et al. [35] outline their idea of a user-centric context model based on six fundamental context parameters: who, when, where, what, how and why. In their model, Context-Element serves as a basic type, and context is a composition of instances of Context-Element. In our approach, I use the state of context attributes to serve as the basic type. Wang et al. [78] use ontology-based modeling techniques to model context and use Ontology Web Language (OWL) for context reasoning. Wang's proposed extensible context ontology "CONON" defines upper ontology, which is a generic model, and domain-specific ontologies can refer to it. Similarly, Ejigu et al. [23] introduce an ontology-based context model, and their work supports context reasoning by providing structures for context, rules and their semantics.

State machines, as a lightweight, human-readable and easy to parse approach, have been used for activity recognition [47] [74]. Teixeira et al. [74] present an activity-recognition system for assisted living applications and smart homes, using lightweight hierarchy of finite state machines (FSMs) to detect actions and activities (sequences of actions). Benitez et al. [47] use a mealy machine to realize action recognition in video sequences. The State Transition Data Fusion Model is proposed by Lambert [21], and it is "*A unification of Sensor and Higher-Level Fusion.*" The model uses states and transitions to represent the real-world status. Its purpose is to combine the sensor and higher-level fusion concerning the prediction, observation and explanation of state transitions.

Bousdekis et al. [28] utilize Bayesian Network (BN) for knowledge representation and reasoning under conditions of uncertainty. Mourchid et al. [40] present a recommender system

for Places Of Interest (POI) using Markov Chain State Models. The system leverages contextual information for providing more relevant POI recommendations. Our approach generalizes the usage of states for modeling context by introducing a state-based context conceptual model.

Some research uses historical context to enable context-aware proactive decision. Hong et al. [40] propose to predict user's preference using context history in order to provide proactive personalized services. They introduce a four-layered agent-based system framework, which uses an ontology-based approach to represent context. Their framework is designed to generate static decision trees of activities to guide proactive decisions. However, their work is not capable of describing dynamic context changes and relations among states of context, which are the core concerns of our paper. Vanrompay et al. [77] discuss the importance of quality of predicted information derived from historical context and introduce the quality of future context metrics for facilitating context prediction.

3. 2 Extended Context Understanding

In this section, I first introduce existing definitions and understanding of context. Then I present the rationale of using states to model context and also provide definitions for terms used in our research. Based on the understandings and definitions, I introduce our state-based context model.

3.2.1 Context Definition

One of the most known definitions for context is provided by Dey & Abowd [30] as

> "*Any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application*

themselves. Context is typically the location, identity and state of people, groups and computational and physical objects."

Based on this definition, context can be used to describe the whole situation of an entity, and it can also be used to refer to a specific attribute, e.g., "a body temperature is 98 °F" is a person's context. In this paper, I refer to context as all relevant information.

3.2.2 Context Understanding

An object in the domain of an application can have various attributes. Only those attributes that are relevant to representing the situation of the object constitute its context. One exemplary context attribute is the location of a person. A person's location information can be described as "Person Peter (P1) is in Room R1 of Building B1", which specifies a direct relationship between the location (attribute) and the person (object). It can also be described as "Person Peter comes from Room R2 to Room R1," which involves "from…" and "to…" factors describing a state change in the location (attribute).

$$(P1, R2, B1) \rightarrow (P1, R1, B1)$$

Most of the conventional modeling techniques are focused on describing only the static context state and have not paid enough attention to the dynamic state changes as well as internal complex relationships among states, which is highly related to historical context data. However, in our understanding both state changes and relationships among states are also important components of context.

To represent a dynamic context and their internal relationships among states, I introduce concepts of context attribute, context attribute state and context attribute states transition. These concepts are partially inspired by the work of Padovitz et al. [3]. I defined context attribute, context attribute state and context attribute state transition as follows:

Definition 1. *A Context Attribute is the ontology of entities that constitutes context of an object. It is another word for characteristics and can be represented by any type of data. For example, the temperature of a house is an attribute of the house (object) and can serve as a context attribute of the house.*

Definition 2. *A Context Attribute State refers to a particular condition that a context attribute is in at a specific time. The condition types include, but are not limited to, a value (meaningful or just numerical) or a set of values to that context attribute where each of the values in the set has high relevance and corresponds to a state that may or may not have been assigned a meaningful word. For example, the temperature is a context attribute of a house, and can have "Hot" or "Cold" as its states. The value may also be numbers as well as strings, like "70 °F."*

Definition 3. *A Context Attribute State Transition describes state changes of context attributes, and it usually includes the conditions and effects of a state change, which can be represented by a pair of assertions and triggering functions. When the assertions are true, the corresponding state transition happens. Effects may happen before a state change.*

Not all context attributes need to maintain many states; for example, users' profiles. Once created from receiving attributes of a user may not change frequently. However, context related to a user's daily life, like location of a person, health condition of a patient, or any features that are prone to change, can have many states.

3.2.3 A State-based Context Model

Context and situation [47] can have states just as a context attribute. Context state and situation space have already been defined by Padovitz et al. [3]. I define situation state as follows:

Definition 4. *A Situation State is an evaluation of situation space [3]. It is a combination of context attribute states at a specific time, and the combination forms a meaningful situation. For example, one evaluation of a combination of location and activity could be "eating at restaurant."*

The relationships of aforementioned contextual concepts can be illustrated using Figure 2. Context attribute state as the basic element of a context attribute and the state-based context model constitutes more complex elements including context state and situation state, and each of them forms context and situation, respectively. States are related to transitions. Transitions of context state and situation state share the same definition with context attribute state transition, and these transitions are also a part of context in our model.

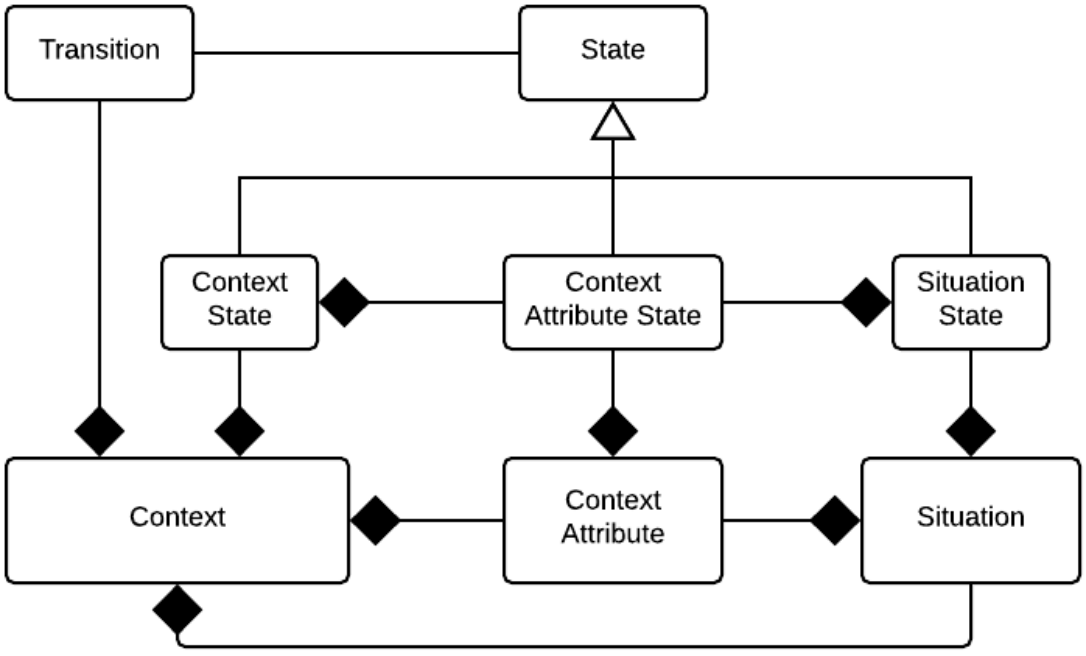

Figure 2. A State-based Context Model

3. 3 Context State Machine

Context attribute states as basic computing elements have inter-relationships among each other. Complex elements such as situation states and context states can also have inter-relationships. To better model the context-related state changes and the complex relationships among states, I propose a state-based modeling approach that is intrigued by mealy machine [29], and name it as Context Mealy State Machine (CMSM or CSM). A mealy machine [29] is a kind of FSM that can have multiple inputs and outputs during each state transition. A CMSM uses the values of context-related concepts such as context attributes to generate states and uses the transitions of states to represent the relationships of the values. The CMSM of context attribute, situation and context are called Context Attribute State Machine (CASM), Situation State Machine (CSSM), and Context State Machine (CSM), respectively. It is possible that a situation includes all entities in the context domain, then a CSSM in this scenario means a CSM (In other chapters, CSM refers to a general concept and includes CASM, CSSM).

3.3.1 Context Mealy State Machine

A CMSM is a 5-tuple $(S, \Sigma, \Lambda, T, G)$ consisting of the following:

1) A finite set of states S (context states, situation states, or context attribute states).
2) An optional start state (also called initial state) S_0, which is an element of S.
3) A finite set called the input alphabet Σ.
4) A finite set called the output alphabet Λ.
5) A transition function $T: S \times \Sigma \rightarrow S$ mapping pairs of a state and an input symbol to the corresponding next state.
6) An output function $G: S \times \Sigma \rightarrow \Lambda$ mapping pairs of a state and an input symbol to the corresponding output symbol.

I define a few steps in building a CMSM:

1) Identify the type of CMSM that needs to be generated and identify the related context attributes.
2) Generate states for context attributes. Because I am focusing on high-level context modeling, the states are usually the results of context fusion of lower-level context.
3) Build connections between states. Identify each transition's trigger conditions and effects. Specify interaction rules.
4) Update the CMSM when necessary. A CMSM can add new states or create new transitions between states.

3.3.2 CASM, CSSM, and CSM

I use location as a key context attribute to build CASM for a person and to help generate SSM and CSM. Figure 3 shows a CASM of a person's daily routine with respect to (w.r.t.) location. The condition of the transition in this example is "from Home to Starbucks." The output is an exemplary action trigger, and it sets the previous state of "Starbucks" to be "Home."

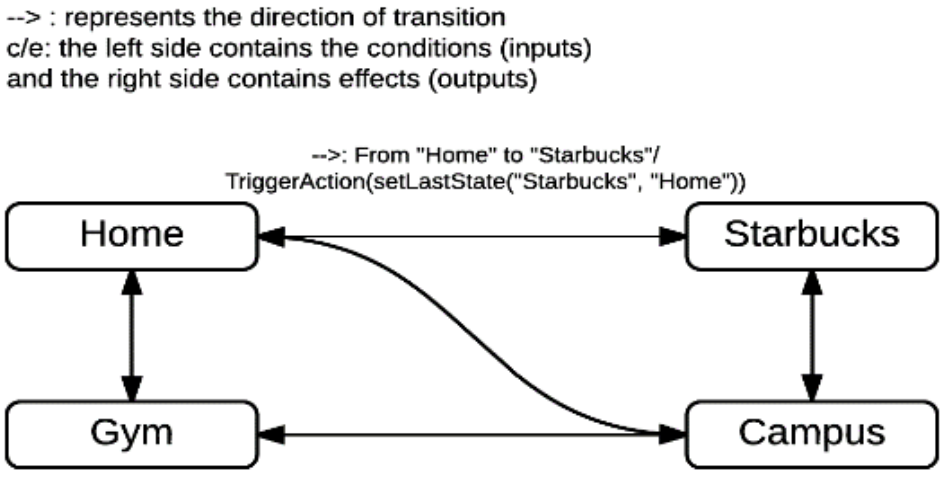

Figure 3. A CASM of a Person's Daily Routine w.r.t. Location

A situation includes those attributes of interest. The location information of a person can help to derive a situation of the person. If we know the states of the person's laptop, we may infer whether the person is relaxing, learning, or working. The states of the person's body activity can help us determine whether the person is exercising or not. By combing these attribute states, we can generate the situation states of the person, such as "Relaxing at Home," "Learning at Home," "Relaxing at Starbucks," "Working on Campus," "Exercising at Gym", and so on. The CSSM is depicted in Figure 4.

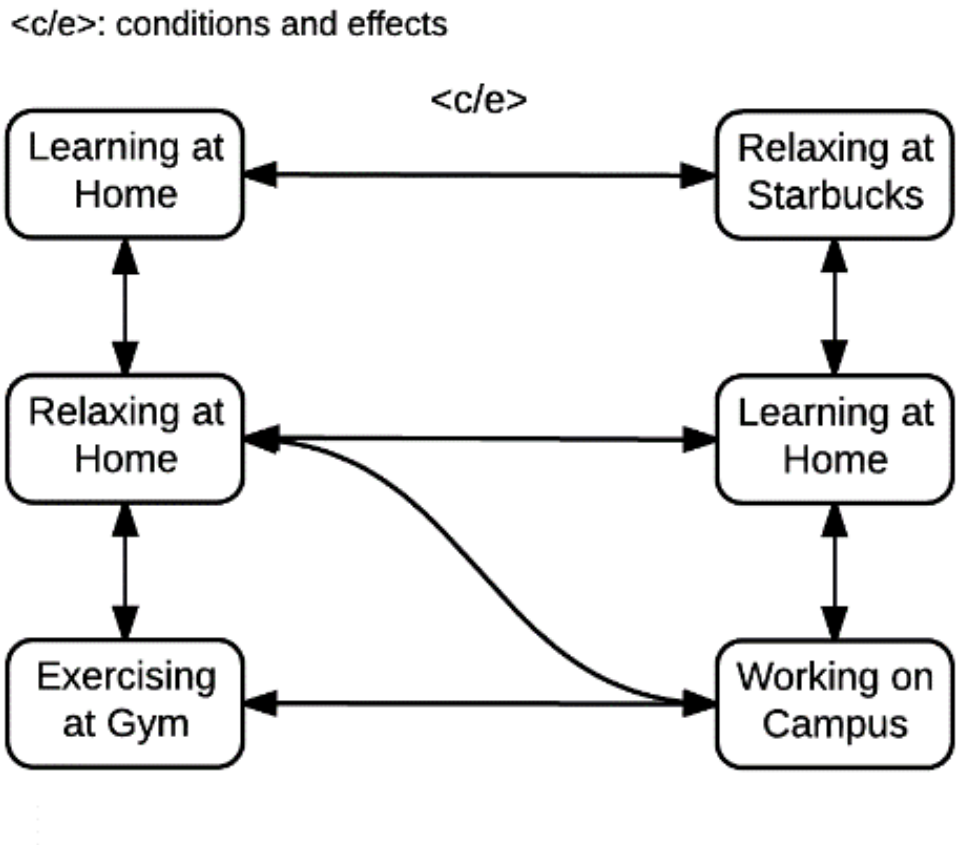

Figure 4. A CSSM of a Person's Daily Routine

A context state of a person includes all available information of the person within a period of time, such as location, health condition, device's status, temperature of the person's

environment, the person's family states, and so on. Although it is possible, it will be very hard to generate CSM if there is too much information because of the combinatorial explosion problem. I just list two states in Figure 5 to illustrate a CSM.

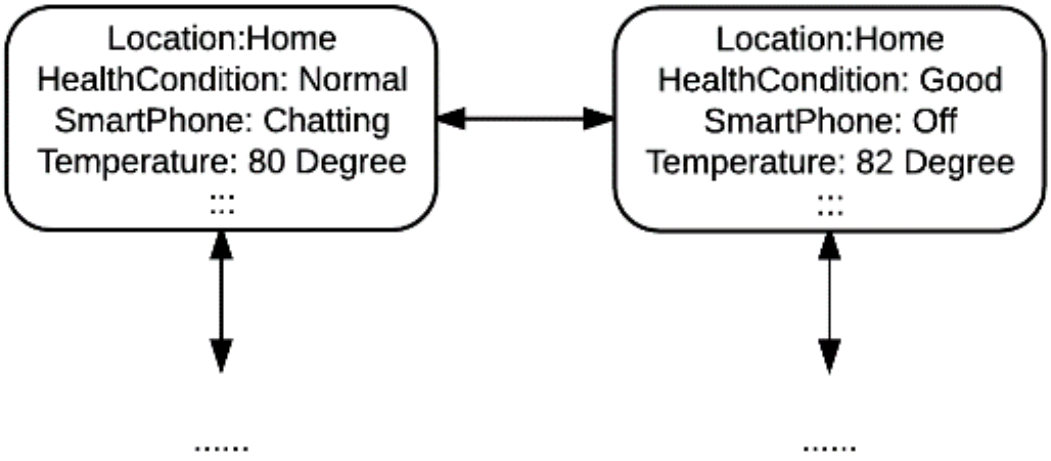

Figure 5. A Fragment of a CSM

3.3.3 Discussion

As discussed in subsection 3.3.2, we can use CASM, SSM and CSM to represent complex context. The benefits and concerns of using them lie in five facets, namely number, granularity, domain, constraints, and extension.

Number: The number of states is an important factor of CMSM because as the number increases, it may cause exponential growth. To solve this, more modular, hierarchical and well-structured state diagrams should be explored. If we focus on the most valuable states and their inter-relationships, the number of context states for some objects can

become steady. Like a person (object) can have locations (attributes) of "usually to go," it can only take values like office, lab, home, and gym.

Granularity: CMSMs can be built on different levels of granularity. For example, the location information can be geography latitude and longitude. It can also be building names, which are daily terms (e.g. "Starbucks") and can be understood by human beings. Usually the lower-level can be used to produce the higher-level information, but in consideration of usability, the best practice should be a context attribute state machine built upon a level where it is practical to balance the information representation and computing efficiency.

Domain: Domain problems may occur when building CMSMs for more than one object of different domains, but they share same context attributes. For example, a person and a car are from two different domains, and both the person and the car have a context attribute "temperature". The domain problem in this case can be caused by different standards of being "hot" or "cold". "Hot" for a car may imply a value more than 200 ^{o}F, while for a person, only 100 ^{o}F.

Constraints: Constraints in state transitions include conditions and effects. They are interactions between a state machine and external environment or between different state machines. Constraints should be carefully designed because the interactions can be complicated. For example, if two state machines wait for similar types of conditions from one source (e.g. two robots are standing in front of an elevator and waiting for the same signal), the sequence of execution may matter a lot if the two machines are interacting with each other.

Extension: A CMSM can be extended by adding a state, generating transitions, and adopting state machines from other systems. A state can be added through merging existing states. For example, one state is "Doctor A is in operation room R", another state is "Patient B is in operation room R", and then a potential new state can be fused into "Doctor A is treating patient B". These source states can come from different objects and the result state can go to another object.

3. 4 Implementation

In this section, I first describe a context-aware system, which is part of our smart campus project. Then I describe the prototyping of CASMs in the system and use prediction scenarios to illustrate the usage of our modeling approach. The description of the system is as follows.

> *A central context-aware system for monitoring experiment tables controls the "open," "pre-heating," "working," and "close" functions of facilities. The system can acquire information from cameras in the lab and smart devices like people's phones and watches, and use them to get location information of every person. Mike and Peter are colleagues of a laboratory. Both are working with experiment tables.*

Location and time are two of the most often used attributes of context information [30], so in our system I concentrate our examples on collecting information of these two attributes. The list below serves as the input of our context-aware system:

PERSONAL: Mike; Location: Home; Time: from::00:00am, to:: 12:20pm.

PERSONAL: Mike; Location: Gym; Time: from::12:40pm, to:: 1:45pm.

PERSONAL: Mike; Location: Lab; Time: from::2:10, to:: current.

…

PERSONAL: Peter; Location: Home; Time: from:00:00am, to:: 12:10pm.

PERSONAL: Peter; Location: G Library; Time: from::12:30pm, to:: 1:00pm.

PERSONAL: Peter; Location: F Dining Hall; Time: from::1:20pm, to:: current.

…

We can extract a CASM from the information as shown in Figure 6, where the context-aware object is a person, and the attribute is the location. Peter and Mike are in the same lab, so they can share the same CASM. A person can go from G Library to F Dining Hall, then go to Lab, and after that go to Gym. The G Library, F Dining Hal, and Lab are all in the campus area, so when the person is in these buildings, the person's location can also use the "Campus" in (b). To represent the locations of the smart campus in a higher-level granularity, the (a) diagram is zoomed out to (b):

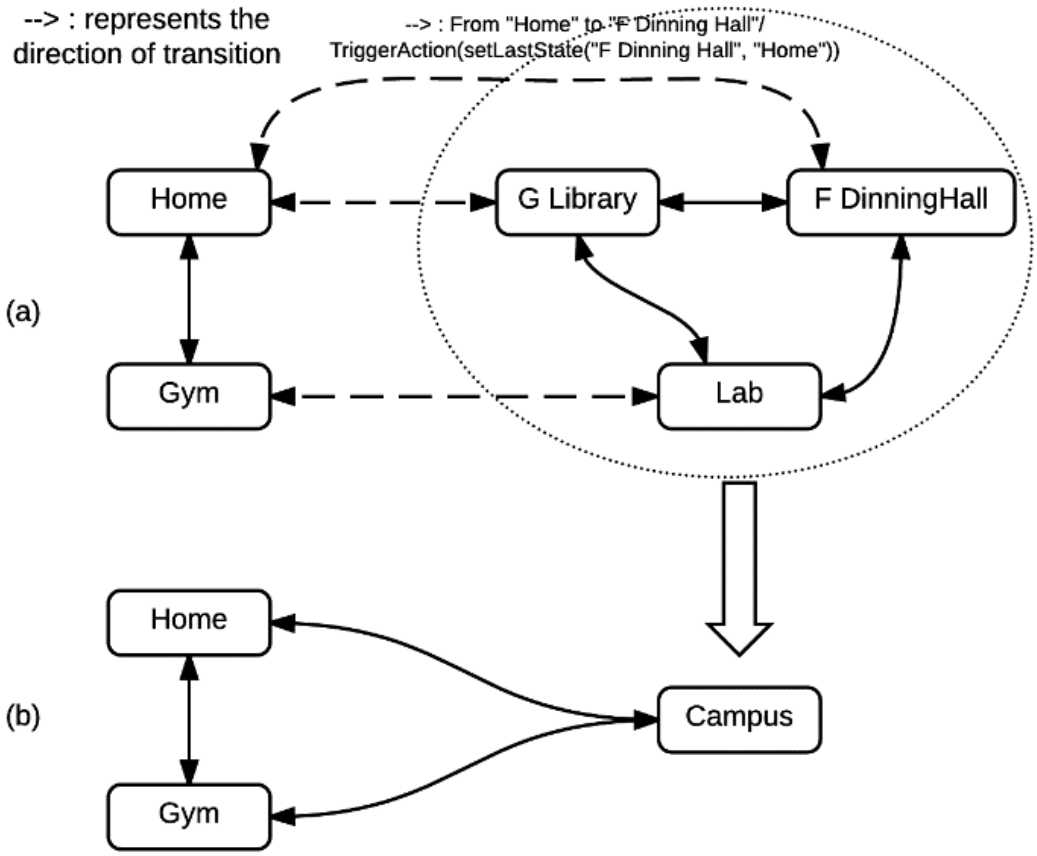

Figure 6. A CASM for Smart Campus

The implementation of the system is represented in the class diagram as seen in Figure 7. One *context object* can have multiple *context attributes*, and many *context objects* can share one *context attribute*. Persons are instances of *context object.* A person can have context attributes such as location, time, and activity. In our system, all colleagues in a lab have a location attribute and they may share the same CASM of location.

The system uses the *ContextAttributeStateMachine* class to implement a CASM. One *context attribute* can have many CASMs like (a) and (b) in Figure 6. A CASM can have at least one *transition* and two *context attribute states*. Two *context attribute states* are connected via a

transition. A CASM maintains a *current state*, a *context attribute states list* and a *transition list*. A *transition* maintains the *input list* and *output list*, and the *former state* and *next state*. A *context attribute state* maintains a *name* of the attribute, its *last state list*, and *next state list*.

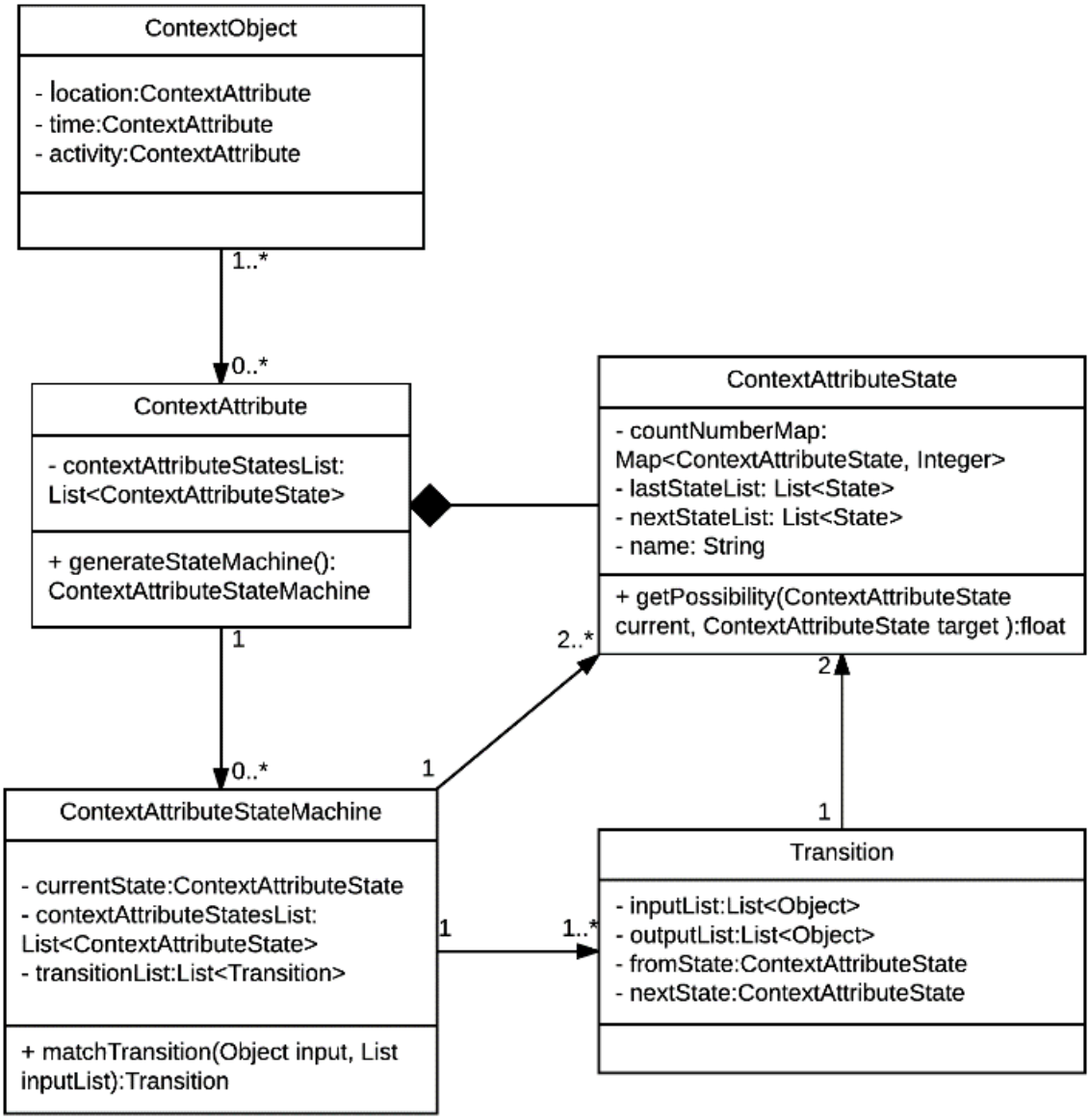

Figure 7. CASM Class diagram

This CASM can help to model and generate complex behaviors and make predictions. Suppose through analyzing the CASMs of Mike and Peter's locations, we find that if they go to Lab from F Dining Hall, they will stay for a long while working on the experiment table, while if they go to lab from Gym, they will stay only for a short time, and then leave for F Dining Hall. These patterns (contextual information) can be obtained from the combination of the CASM in Figure 6 and a CASM of their activities in lab. Then, the system can choose not to preheat the experiment tables if they go to lab from Gym.

Another prediction scenario can be more complex. Suppose Mike is in F Dining Hall, and from historical data we can know the probability that Mike will go to Lab is 0.98 and the probability that he will go to other places is 0.02. The experiment table can preheat when the context-aware system detects that Mike is leaving F Dining Hall and without being sure of where Mike is heading. Further, through comparing the both CASMs for Mike and Peter, the system can know that if Mike goes from F Dining Hall to Lab, and Peter can be anywhere else, there is a probability of 0.90 that Peter also goes from F Dining Hall to Lab. Even the individual probability that Peter goes from the place he is now in to the lab is low, the system still can preheat and if the system predicts wrongly, the confidence of the effects factor from Mike can decrease. To fulfill these mentioned functionalities of prediction, the system can adapt naïve algorithms to count the possibilities using the "*countNumberMap*" within each context attribute state. It can also use other learning algorithms like Bayesian Network or Markov Chain [2] [28], which can be generated with the help of our modeling approach.

3. 5 Summary

Modeling context is essential and challenging. In this chapter, I describe a state-based approach to context modeling and the definitions of CSMs. A central context-aware system for

monitoring experiment tables is described to illustrate how to use CSMs to model context in the real-world and compute the related possibilities by utilizing the state-based context models.

The state-based modeling approach is built on top of the ontology concept, a modeling and computing mechanism, and gives special attention to inter-state relationships of each context attribute. This approach can be further implemented using an object-oriented programming languages and relational databases.

CHAPTER 4

CONCEPTUAL DESIGN FOR APPLYING CSM TO A CONTEXT-AWARE ELEVATOR SYSTEM

While context as a concept is being explored for context-aware systems, the information which can be concluded as context evolves [6], with the advancement of information science, from simple location information to complex facts (the states of all related entities for an application). These facts can usually be calculated out from multi-source information (data of sensors deployed for the entities) and historic contextual data. This research is to model complex context for facilitating context-aware software development and for implementing the CSM model. A better implementation can improve the efficiency in the context data area with respect to complex systems and make data live in context state machines, analogous to what happens when programmers first use classes to encapsulate data and makes data live in classes in a program. To demonstrate how to apply the CSMs in a real-world context-aware system, I implement a CSM engine and show how the engine can be used to model the context of a context-aware smart elevator system. The CSM engine realizes the core concepts of the state-based modeling approach and provides interfaces for reasoning contextual information.

In this chapter I use a basic smart elevator scenario to show that CSM models can be used to demonstrate effectiveness for known context-aware problems from the literature. First of all, the smart elevator, as a context-aware application in literature, is introduced. Secondly, I introduce the implementation of the CSM engine. Thirdly, I describe a scenario which uses

limited context information to show the model can help automatically capture the contexts and reason the context. To our knowledge, it is the first time in literature of applying state-based modeling approach and the CSM engine to a real-world context-aware system.

4.1 A Revisit of the State-based Modeling Mechanism

As shown in Figure 8, adapted from a multilayer framework of context processing [10], the state-based representation and reasoning is in Layer 3, the level for context modeling and reasoning, in between the processed contextual data and the context query languages. Layer 1 is for contextual data acquisition to extract raw contextual data; Layer 2 is the initial processing of the raw data to fit for application domain and context modeling; Layer 3 can use different context modeling and reasoning techniques and support the context query language to search context information. Layer 4 is the context query languages which can be used within context-aware applications to query meaningful context data for context adaptation.

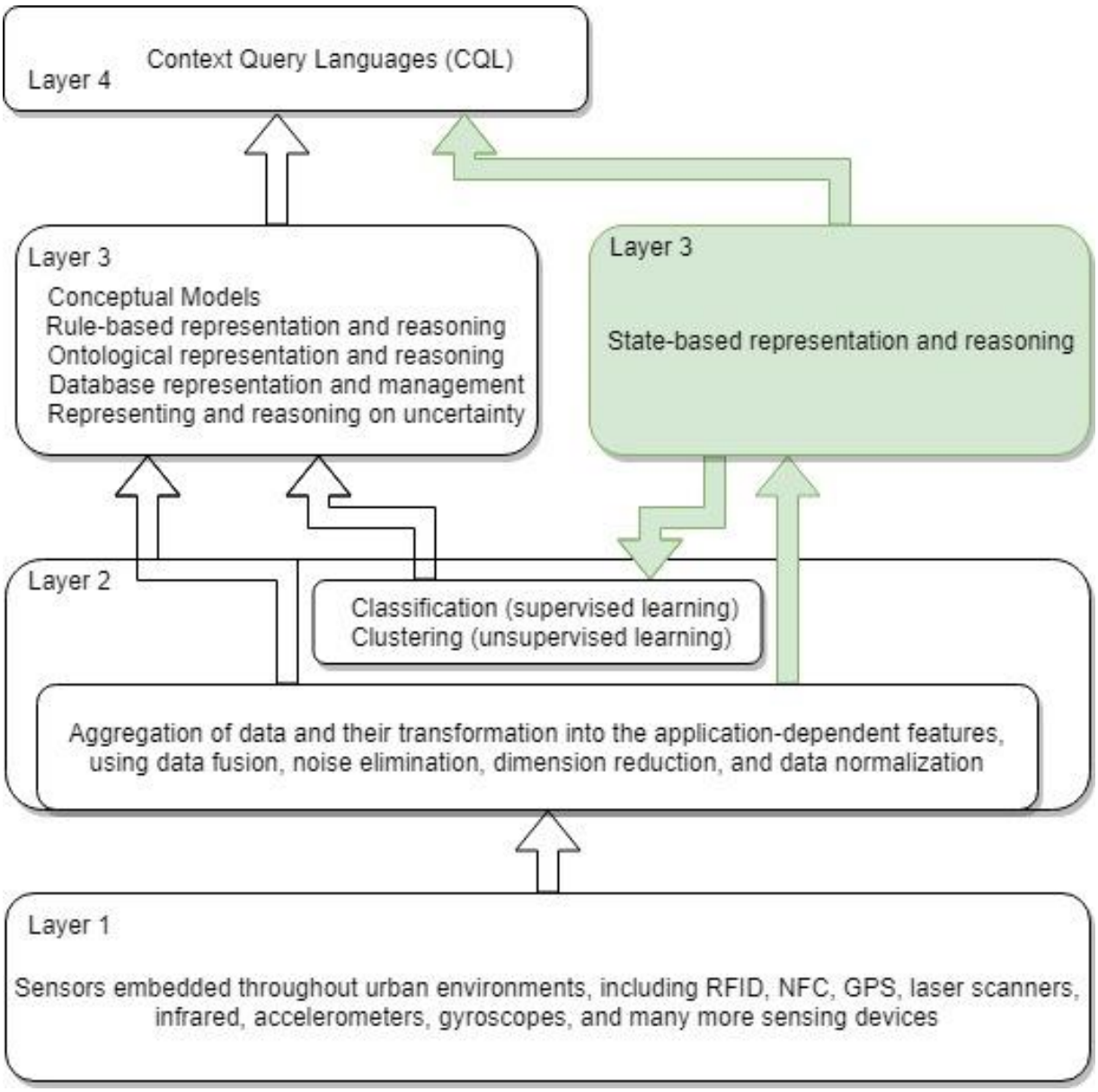

Figure 8. State-based Representation and Reasoning in CAS (Adapted from [10])

Layer 4 is for the context-aware applications to query context data, so it can also use other techniques beyond context query languages, such as calling public interfaces supported by layer 3. Thus, the implementation of the state-based modeling task involves representation part

and the coding phase. In the representation part, it is based on the ontology modeling of the context. It does not only use ontologies to represent the entities but also to represent the states of the entities. It could also use ontologies to represent the transitions of the states. In the coding level, it is implemented in an object-oriented style; however, it is different with the existed object-oriented modeling of context [38], I treat the context object, context attribute and context state as individual classes [66]. The implementation supports context-aware software to use public interfaces to search meaningful context.

4.2 Context-aware Elevator Controlling System

A "Smart" Elevator in this research is different than a traditional elevator by adding contextual information to improve its own decision-making during service. I selected an elevator as an example because it can be related with rich context. Usually elevators are located in buildings and people can use them to go to different floors. The persons' information and the building room information, as well as the weather can broadly count as the context of the elevator. It has its own context such as temperature and speed; however, to demonstrate our modeling approach to general context, I mostly consider the persons' information and their decisions.

A context-aware elevator controlling system controls the "open door," "go-up," "go-down," "close door," and "stay" actions of all elevators of a building. Context-awareness enables elevators to change its actions to fulfill the passengers' needs. For example, if a student approaches an elevator, and there is no one else in-front of or using the elevator on any other floor; however, the elevator compartment is on another floor, then the system can dispatch the compartment to the floor where the student is and wait for the student. More complex context-aware functions of the system are that it can give predictions about whether the student will take the elevator as well as the urgency. To acquire contextual information, the system can be

connected with the cameras in the building and smart devices like people's phones or watches. Historic contextual data can also be used to make predictions. This type of system can help reduce the passengers' average waiting time, the maximum waiting time and the energy consumption of elevators [43] [69].

4.2.1 Literature Review of Context-aware Elevator

Strang et al. [69] research on context-aware software in a system-centric way, especially to elevator system. They show with three different scenarios that the system performance can be significantly improved if the elevator control can access related contextual information. The first one is about choosing an appropriate elevator group scheduling algorithm when the current or near-future passenger traffic situation changes. The second example is to use RFID sensors to predict the passengers' destination. The last example shows the adaptation to a specific emergent situation can increase the transportation capacity of an elevator system so that as many lives as possible can be saved in an emergency situation. The three scenarios cover three types of usage of context-aware mechanism during the context-aware elevator system design.

Kwon et al. [43] specially explore using the sensors deployed at each floor of an apartment building to detect the passengers' behavior before they come to the elevator door and push the elevator call button, and the detected information is furtherly used as contextual information and apply to the elevator scheduling system. Through extensive simulations with different passengers' traffic situations, the proposed context-aware solution performs better than the conventional elevator scheduling systems in terms of the most commonly used measures for scheduling algorithms, namely, the passengers' average waiting time, the maximum waiting time, and the energy consumption of the elevator.

The smart elevator system today can use context information because smart buildings can supply a lot of context information and the Internet of Thing (IoT) can give information of human beings. Thus, more types of information can be used as context, and the complexity of context increases. The design purposes usually are to decrease the waiting time and improve the efficiency of the system. In our approach, this research tries to model and utilize the complex context to further decrease the waiting time, or even to decide whether a person will take the elevator or not. The next subsection will introduce the scenario and the overall concept and design of the system, which greatly needs better context modeling techniques to handle the contextual data and make automatic reasoning. The key part is to make automatic reasoning with the rich potential of the IoT to provide complex context information beyond an ability to manually process the context information [5].

4.2.2 Context-aware Elevator System Structure

I adapt a system structure from [41], and replace the camera module with a more general context-aware module: a context-aware service module (Figure 9). The system is composed of two units: The Group Control Unit (GCU) and the Control Tuning Unit (CTU). The GCU selects a car using a dispatch function, which is kept being updated by CTU. The CTU runs as a semi-online procedure and produce s a dispatch function considering the context information. If the context of the elevator, including the traffic flow, changes remarkably, the context-aware service module will notify the GCU, and the real-time GCU requests the CTU to generate a new dispatch function by means of Genetic Algorithms (GAs). The GCU should contain a context-aware adapter inside the unit.

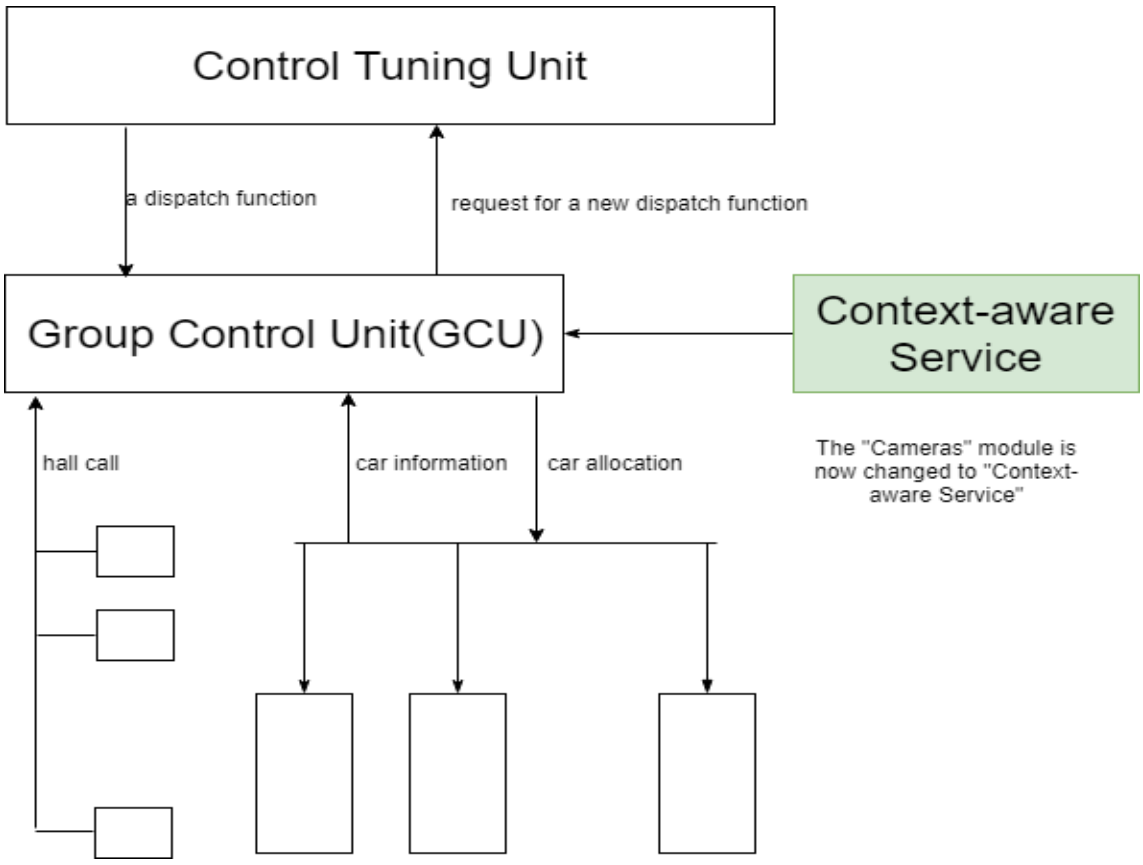

Figure 9. Structure of the Context-aware Elevator System (Adapted from [41])

The work of [41] shows that by using context, including traffic flow, the status inside elevator, and the directions of passenger movement and so on, the system can notably reduce the passengers' average waiting time when the crowding of elevators reaches between 45% - 75%. I use context-aware service to supply more contextual information, which can be used as parameters for tuning by GAs. As stated by their research work, the CTU can have a budget of a few minutes. It would be efficient to calculate out the context information by the context-aware service module within 1 or 2 minutes. In the implementation and evaluation chapter, I show that

the context modeling and reasoning process can be conducted within 2 minutes, given a great amount of data items.

In this structure, the *context-aware service* will use the CSM engine to model the context and reason the probability whether a person will take the elevator or not, which is a context reasoning process applying techniques such as probabilistic logic or rules.

Since the context information can help to improve the performance of the elevator systems has been proved [41] [43] [69], and the details of the proof is beyond the scope of our research, to evaluate the state-based modeling approach I focus on the measurement of the performance of the CSM engine, which is the core of the *context-aware service* module.

4.2.3 An Ontology Model for Smart Elevator

State-based context modeling is based on the ontology modeling and works in a higher-level (compared with meaningless raw sensor data level) to model complex context and support context reasoning. The first step of using states to model the context of smart elevator is to identify the problem domain's ontology model. Figure 10 shows a part of context ontology diagram of a smart elevator system.

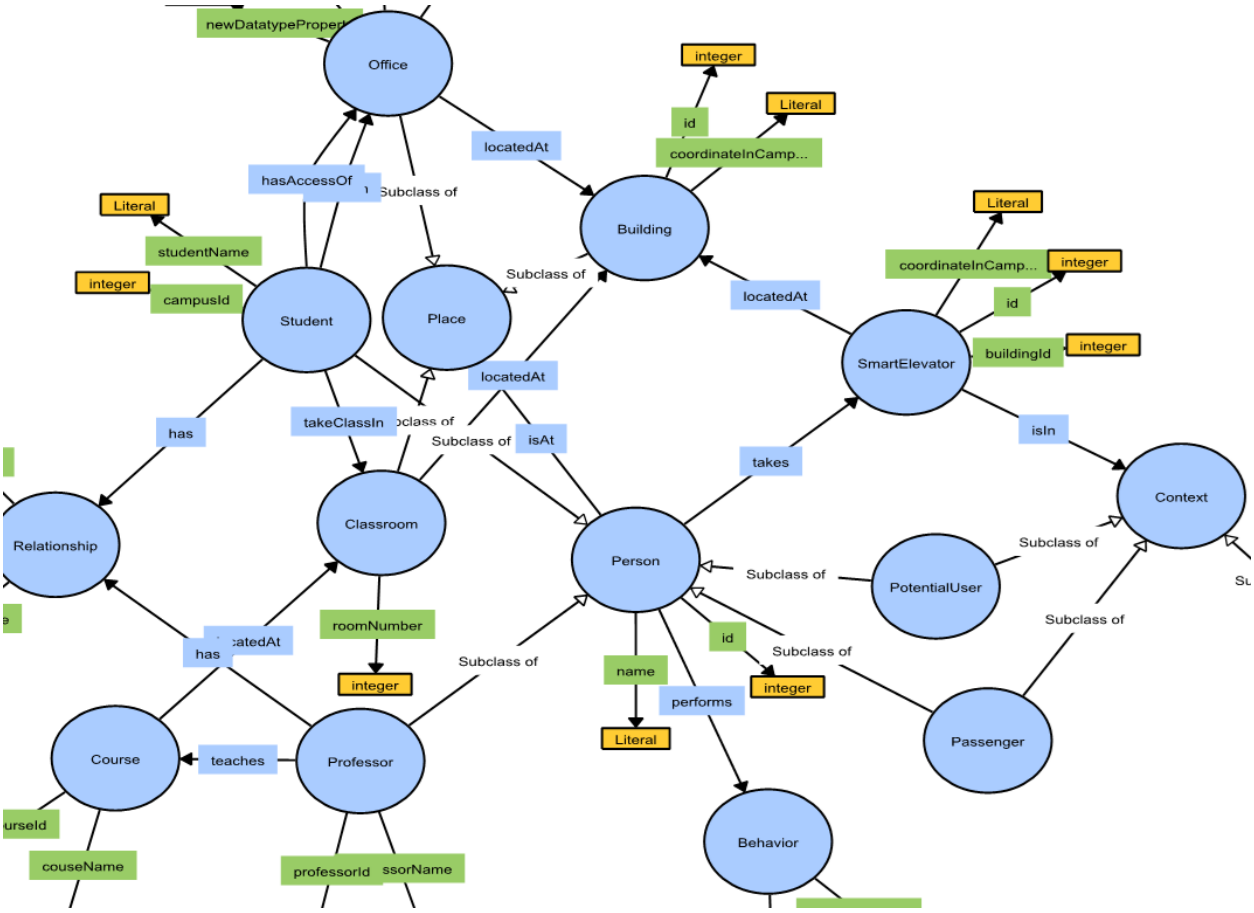

Figure 10. An Ontology Diagram for Smart Elevators

Smart elevator will use related contextual information to decide whether a person will take the elevator or not. Thus, the basic ontologies are the smart elevator itself, the building it is in, and the person that is approaching the elevator. For example, the persons can include students and professors. The relationships between them can be advisor, friend or colleague. The person may teach a course or take a course. The course is related to a classroom in a building. A student or a professor can have an office in a building. The context of the smart elevator is also a related ontology in the domain, which can include potential user, passenger, security etc.

I mentioned in the prior section that the state-based approach is based on the concept of an ontology model. In the experiment demonstrated in the next chapter, I only use person, place and behavior to build the state machine. Based on the ontology model of the smart elevator, I build context attribute state machines for the object: place, person and behavior.

4.3 Two Context-aware Scenarios and Research Questions

The smart elevator system contains rich context for testing the modeling and reasoning ability of CSMs. I examine two context-aware scenarios of the system, use CSMs as a method to model and reason their contextual data, as a result, I answer the two research questions of our dissertation.

4.3.1 A Basic Context-aware Scenario

Our research question 2 is that "Can CSM model for context-aware software development be used to demonstrate effectiveness for known context-aware problems from the literature?" In order to address Research Question 2, this dissertation develops the CSM model for the basic situation of an elevator control system with limited contextual entities and constraints. The research on context-aware elevator is from the literature and I surveyed the work in subsection 4.2.1. The implementation and evaluation of the modeling and reasoning part is discussed in Chapter 5.

With the simple context-aware mechanism, there can be a requirement for the elevator to park on a floor when someone is approaching. A basic context-aware mechanism requires reasoning processes on limited entities and constraints, for example to predict whether a passenger will take the elevator or not by only considering some characteristics of a person or the history of an attribute (e.g. location) of a person. If the person is less likely to take the elevator, the elevator compartment can stay on its original floor to avoid wasting power. To solve this

problem and model the problem domain, the ontology-based modeling approach can be used to represent the relationships between the person and an office or classroom in the building. If the person has an office on another floor, or some historic data exists showing the person has records of taking the elevator, the compartment would better locate to the floor of the passenger. Systems can use an ontology to represent all related entities (elevator, building, person etc.) and their relationships to support context reasoning. However, for this level of context-awareness the context reasoning processes can be designed before or during the system implementation.

4.3.2 A Complex Context-aware Scenario

Our Research Question 3 is that "Can CSM model for context-aware software development scale to multiple entities and constraints?" In order to address Research Question 3, this dissertation extends the elementary situation from Research Question 2 to an elevator control system with multiple contextual entities and constraints. I describe a more complex context-aware situation in the next paragraph, and I will show that the CSM models are able to model the context and make a more complex context reasoning. The implementation and evaluation of the modeling and reasoning part is discussed in Chapter 5.

The state-based modeling approach is specially fit for handling situations of a more complex context-awareness. Situations which make a person use the elevator or not can become complex and beyond an application's capabilities to know in advance. For example, if students got accompanied by professors, they may not use the elevator; or if a student is approaching the elevator and a professor whom the student is working with is in another building, the student will take the elevator. These habits may not be predicted by the application beforehand and can only be learned from real-time contextual data. A modeling approach which contains all meaningful states is necessary to facilitate learning processes. By using the Context State Machine (CSM)

modeling approach, not only will the entity's ontologies and the relationships of them be represented, but also all meaningful states of entities' attribute will be represented. The meaningful states can then be used to computerize the situations that a person will use the elevator. The compositions of these meaningful states of different entities can form different situations, and the situations can be connected with whether the person will or will not use the elevator. By accumulating the situations' data, a data source for learning processes can be built. Besides, the situations may need to refer to some state transitions, which is contained in the CSM approach. The state transitions assure that the contextual information does not only include static information such as what the student is doing, but it also contains what the student was doing.

4. 4 State-based Representation and Reasoning Design

According to the advocator of object-oriented (OO) approach to context modeling, the overall context can be viewed as an object of OO paradigm [38]. In the state-based approach, the universal concept atoms include object, attribute and state, and specially, attributes are taken out of objects and states of attributes are taken out of attributes, in which way it preserves the best extensibility for modeling in both concept level and programming level.

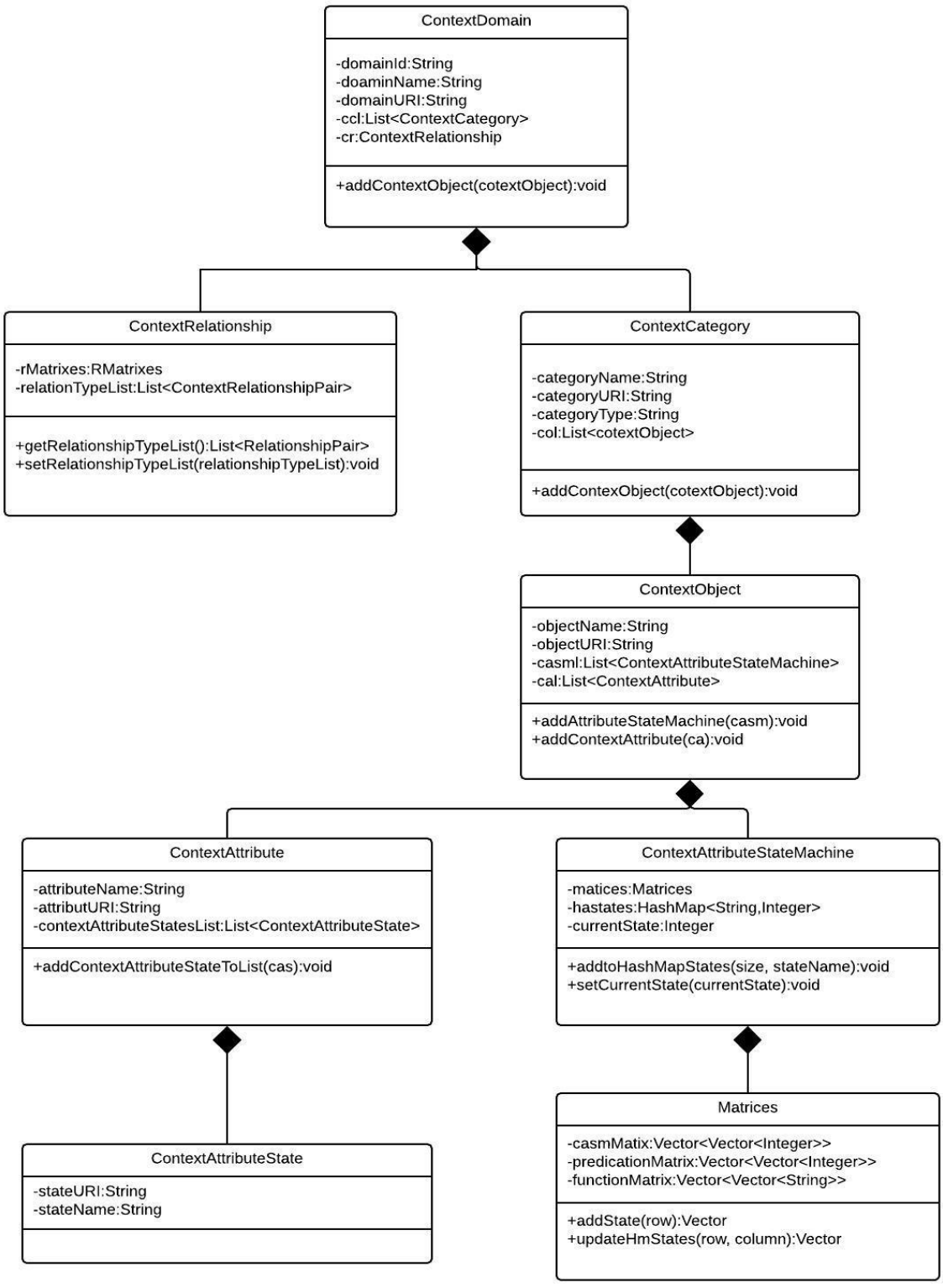

Figure 11. Context Core Model

In Figure 11 the class diagram of context core model is demonstrated. The most upper level of the model is “ContextDomain”. In each context domain, there are different context categories and a context relationship object. In each context category, there are different context objects, such as a person, a device or an application. For each of the objects, there will be a list of context attributes, which is related to a context attribute state machine. Thus, there are two list in a “ContextObject”. For each context attribute, there are several stable context states, which are the basic components in a context attribute state machine. In a context relationship object, matrix is used to represent the relationship status and the relation closeness.

Matrix computing can be applied when context-aware software use CASMs to compute and make context reasoning. One CASM is put in to a matrix. Trigger functions can be put into a different matrix. For CASM matrices, they can be divided into different types according to the time range of states. One type is for all states in history, one type is for one week, one is for one day, and one for the most recent. In this way, the related data are changed into integers and it can greatly save the storage space. Since we have a list to map the states to numbers, we can keep this list as a private key and let them transfer only between servers, and this increases the security of private data. If necessary, after a period of time, the state mapping can be changed as an extra mechanism to protect the data.

After transferring the contextual data into the objects of an object-oriented programming language, the representation of the data in the coding level is finished. Then, we can use the data stored in the CASMs and CSSMs, and design more matrices, for example for decisions, to support context reasoning. Since we transfer the contextual data into numerals in objects and CSMs, it is possible to design algorithms to reason on the numeral data based on fuzzy logics, probabilistic logics or rules.

To utilize the context core and let the contextual data live in context state machines, a CSM engine based on this model is designed and developed. The design and the implementation of the engine is introduced in chapter 5.

4. 4 Conclusion

This chapter firstly describes a context-aware elevator system which can reason with limited inference of developers. The implementation of the context modeling and reasoning part of the system, which is described in the next chapter, is the first case to apply the state-based modeling approach and generate a CSM engine to be used by the system. Thirdly, an ontology model of smart elevator domain is given. Fourthly, an algorithm, matrix computing, is used to make reasoning.

Except expressing the essence of context, and facilitating data mining processes, context state modeling and the application of this research will demonstrate contributions in three specific areas. Firstly, developer effort will be reduced to implement similar context-aware functionalities. The modeling approach has been realized into a Java library. Developers can call the APIs of the library to run related context-aware functionalities. Additionally, the developer can let the system collaborate with middleware servers to assemble the library. Secondly, the library is designed to accept context data and system decisions (go up, go down) as input for building the relationships between situations and decisions. Standard data formats are designed to make it easy to use and the developers only need to prepare data which conforms to these formats. The library is developed in an object-oriented programming language and the context is also treated as a composition of objects.

One shortcoming of the CSM approach can be excessive use of storage and computing resources if there are too many entities and states, especially when most of them are not needed.

If the context-aware system is not time-sensitive, the CSMs can be built when the server is in idle status to save computing resources, and the data can be deleted after CSMs are built. Distribution of related tasks to different servers can alleviate computing bottlenecks and improve responsiveness.

CHAPTER 5

CSM ENGINE IMPLEMENTATION AND EVALUATION

In this chapter, I describe the design and the implementation of the CSM engine and examine its efficiency on two situations of context-aware elevator system using contextual data. The basis situation involves limited entities and constraints. The complex situation, which involves multiple entities and constraints, needs the CSM engine to automatically deduce the implicit contextual relationships and further supports context reasoning. The efficiency of the CSM approach is supported by two facts. Firstly, the CSM engine can be used to model contextual information, and further support context reasoning for system decisions. Secondly, the execution time of the CSM and the reasoning process is within a time budget comparable to context-aware elevator controlling system in the literature.

5.1 Design and Implementation of the CSM Engine

The CSM engine is a Java Implementation of the state-based context modeling approach. It contains several modules, including a core module to represent the context data in Java entity classes, a matrix module to transform the contextual data into various matrices in support of context reasoning, a utility module to generate high-level contextual data for testing CSM engine, and a CSM generation module to transform the data into a CASM or a CSSM form. In this subsection, I introduce the key modules, the data storage and data flow to use the engine, and the matrix representation and computing.

5.1.1 Key Modules Introduction

Core:

The CSM engine contains a "core" module, which is highly related to the state-based modeling of contexts. It defines the basic classes for the concepts in context state machine including context domain, context category, context object, context attribute, and context attribute state etc.

To facilitate the data processing, I extend the Triple concept from Resource Description Framework (RDF), and implement the Triple in our framework as shown in Figure 12. A Triple is composed of a subject, a predicate and an object. An extended triple includes decisions and conditions. Condition can contain several triples to represent the condition details. Triples can be used to build and configure CASMs, which will be shown in the details of the implementation.

An example of a triple can be:

Subject: person;

Predicate: leave/arrive;

Object: gym/coffee shop;

Other guards: conditions, decisions etc.

For conditions, it could be the time; for decisions, it could involve the usage of related devices, for example, taking an elevator or not taking an elevator.

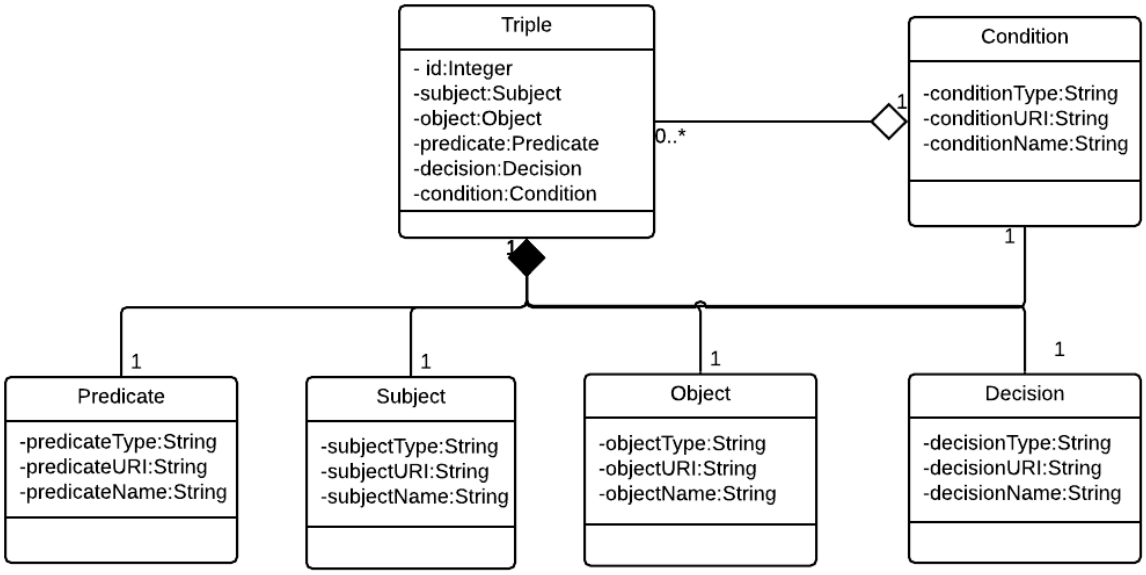

Figure 12. Concept of the Extended Triple

Matrix:

Matrix representation and matrix computation is the key computing paradigm in this implementation. I use matrices to represent the CASM and CSSM. Matrices are also used to represent the relationships. In the matrix representation and computing subsections, I describe our designs details for different data processing purposes such as relationship representation and registration.

***Others*:**

The "*statesGenerationEngine*" module is designed to load data from various sources. Then the data can be used for object extraction, attribute extraction and states extraction. The "*csmGenerationEngine*" module is to use objects, attributes and states information to build CSMs for context objects. The "*actuator*" module can receive CSMs from or distribute to other servers. It also contains functions for "file" to machine and machine to "file" transformation. The "file" can be implemented using xml style formats or database tables.

5.1.2 Data Storage Method

Relational database is a traditional approach to store data and can be explored to store the data of CSMs, objects, attributes, and states etc. One object is mapped to one Object table. Different rows of the Object table are different attributes. One attribute is mapped to one attribute table. Rows are used for storing context attribute states, which are stable meaningful values for context attributes. However, using this way, extra design efforts should be taken to store CSMs and map between the stored data and Java objects in memory.

In our implementation, I use JASON files to store the CSM data, including the related objects, attributes, and states etc. Java language has special packages to process JASON data, such as the packages of "com.fasterxml.jackson.databind.ObjectMapper" and "com.fasterxml.jackson.databind.ObjectWriter". By using them, it is easy to map the data from a file to a Java object in memory and map back from the memory to a file.

5.1.3 Data Flow

Information from sensors and applications can be transformed to triple objects, which can be used to build CSMs. Matrices are used to represent the states of context attributes or context situations and their relations and can be stored in JSON files. The overall of the data flow for a CASM is demonstrated in Figure 13.

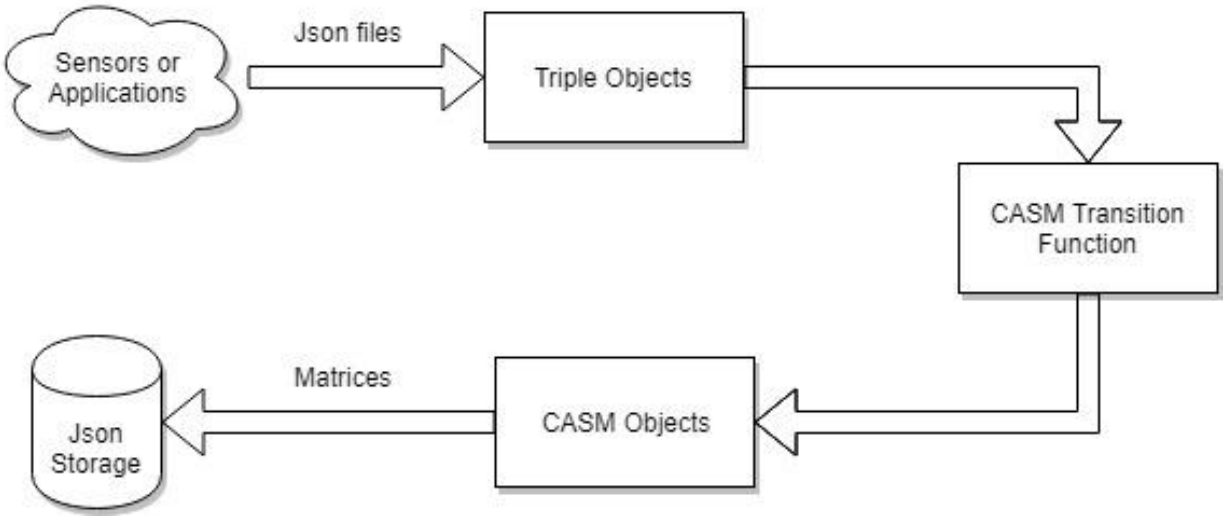

Figure 13. CASM Data Flow

Generation of the Testing Data

To simulate the usage of the CSM engine, contextual testing data from sensors or applications is needed. Since our research does not focus on the information retrieving and pre-processing from original resources but on proof of the prototype of CSM engine, I synthesize the data and then feed it into the engine. I developed a program module specialized for generating reasonable contextual testing data to feed the CSM engine, which can use the data to generate the CASMs or CSSMs and then persist them. The sample of the synthesized data is given in Table 3. The method and rules of generating the data will be introduced in chapter 6.

Id	pId	pName	time	action	actionId	locId	locName	decision
0,	p002,	Frank,	17-Jan-18 03:04:19,	arrive,	action02,	location03,	HOME,	TAKE
1,	p005,	Alice,	17-Jan-18 03:38:31,	arrive,	action02,	location06,	COFFEESHOP,	TAKE
2,	p003,	Joshua,17-Jan-18 04:38:23,		arrive,	action02,	location05,	DINNINGHALL,	TAKE
3,	p001,	Mike,	17-Jan-18 05:28:14,	arrive,	action02,	location04,	GYM,	TAKE
4,	p002,	Frank,	17-Jan-18 06:13:26,	leave,	action01,	location03,	HOME,	PASS
5,	p002,	Frank,	17-Jan-18 06:56:15,	arrive,	action02,	location06,	COFFEESHOP,	TAKE
6,	p002,	Frank,	17-Jan-18 07:37:11,	leave,	action01,	location06,	COFFEESHOP,	TAKE
7,	p002,	Frank,	17-Jan-18 08:25:31,	arrive,	action02,	location04,	GYM,	PASS
8,	p002,	Frank,	17-Jan-18 09:17:29,	leave,	action01,	location04,	GYM,	TAKE
9,	p005,	Alice,	17-Jan-18 10:10:17,	leave,	action01,	location06,	COFFEESHOP,	PASS
...								

Table 3. Mocked Contextual Information.

Convert from Generated Data to JSON Format

In different implementations, the messages can be designed in different formats. A unique or similar format of message can help information exchanging or sharing. URI technique can be used to uniquely identify some object or entity on the Internet, so as to enhance the capability of information sharing and interoperability. The message in our implementation is designed to have device information, domain information, and "infoList", which contains subject, object, predicate, condition, decision and an ID of the record. The information of device and domain, subject, object etc. contains URI for them to be identified in IoT. One message example is given in Figure 14.

```
{
 "infoType" : "any",
 "deviceType" : "smartphone",
 "deviceName" : "any",
 "deviceId" : "androidxx",
 "domainName" : "Smart Campus",
 "domainId" : "1",
 "domainURI" : "test",
 "infoList" : [ {
  "subject" : {
   "subjectType" : "person",
   "subjectURI" : "p002",
   "subjectName" : "Frank"
  },
  "object" : {
   "objectType" : "location",
   "objectName" : "HOME",
   "objectURI" : "location03",
   "fromTo" : null
  },
  "predicate" : {
   "predicateType" : "action",
   "predicateURI" : "action02",
   "predicateName" : "arrive"
  },
  "condition" : {
   "time" : "17-Jan-18 03:04:19"
  },
  "decision" : {
   "decisionName" : "TAKE"
  },
  "rid" : "0"
 },...
```


Figure 14. An example of contextual information in the implementation.

After the running of the CSM engine on the created contextual data, the CSM engine can generate a CASM list and a CSSM list. Figure 15 presents the JSON representation of a context object in the context object list.

```
{
  "col" : [ {
    "objectName" : "Frank",
    "objectURI" : "p002",
    "casml" : [ {
      "matrixes" : {
        "casmMatrix" : [ [ 2, 4, 4, 6, 5 ], [ 4, 2, 4, 4, 3 ], [ 5, 5, 10, 7, 2 ], [ 5, 5, 7, 4, 4 ], [ 4, 1, 5, 4, 4 ] ],
        "predicationMatrix" : [ ],
        "functionMatrix" : [ ],
        "decisionMatrix" : [ [ [ 1, 1 ], [ 2, 2 ], [ 2, 2 ], [ 3, 3 ], [ 4, 1 ] ], [ [ 2, 2 ], [ 2, 0 ], [ 0, 4 ], [ 1, 3 ], [ 2, 1 ] ], [ [ 3, 2 ], [ 2, 3 ], [ 5, 5 ], [ 3, 4 ], [ 1, 1 ] ], [ [ 3, 2 ], [ 0, 5 ], [ 4, 3 ], [ 1, 3 ], [ 3, 1 ] ], [ [ 4, 0 ], [ 1, 0 ], [ 4, 1 ], [ 1, 3 ], [ 2, 2 ] ] ]
      },
      "hmstates" : {
        "DINNINGHALL" : 4,
        "GYM" : 2,
        "COFFEESHOP" : 1,
        "HOME" : 0,
        "ElevatorBuilding" : 3
      },
      "hmdecisions" : { },
      "currentState" : 2,
      "lastTransition" : null,
      "transitionList" : [ ]
    } ],
    "cal" : [ {
      "contextAttributeStatesList" : [ {
        "stateURI" : "location03",
        "stateName" : "HOME"
      }, {
        "stateURI" : "location06",
        "stateName" : "COFFEESHOP"
      }, {
        "stateURI" : "location04",
        "stateName" : "GYM"
      }, {
        "stateURI" : "location07",
        "stateName" : "ElevatorBuilding"
      }, {
        "stateURI" : "location05",
        "stateName" : "DINNINGHALL"
      } ],
      "attributeName" : "location"
  } ]
   }
```

Figure 15. An example of context object in the implementation.

5.1.4 Matrix Representation

The states are mapped with the matrix rows and columns (Table 4 shows a matrix example of CASM; the row number and the column number are start with 0, the state number is also start with number 0; the number 2 is mapped with the state 2) and the states transitions are mapped with the coordinates of the table: for example, the transition from state 1 to 2 is the (1,2)

which is the row 1 and the column 2; the reverse transition from state 2 to 1 is the (2, 1), which is the row 2 and the column 1. The value of each spot of a coordinate pair refers to the times of travelling from the start state to the ending state.

	0	**1**	**2**	**3**
0	2	4	4	6
1	4	2	4	4
2	5	5	10	7
3	5	5	7	4

Table 4. A CASM Matrix Example

The matrix types include matrices for CASMs, prediction, registered functions, decisions, relation and relation closeness (see in Table 5). For one context domain and one functionality, these matrices are parallel with respect to the matrix size and the state numbers; however, the meaning of the value for the spots with the same coordinates are different. From the perspective of dimensions, the matrix includes 2-dimensional, 3-dimensional and could be more depending on the requirement and the computing efficiency.

Matrix Name	Characteristics
casmMatrix	2-dimensional; the data is the times of travelling.
predictionMatrix	Reserved and to be designed; can be used to store the predictions' statistics.
functionMatrix	Reserved and to be designed; can be used to store the registered function information for each transition or states
deicisonMatrix for CASM	3-dimensional; the first two dimensions are the same with casmMatrix; the third dimension is for the decisions; the corresponding number will increase if one decision is taken.
rmatrix	2-dimensional; the rows and the columns are the entities' sequences; if there is no relationship between two entities, the number is 0, otherwise -1.
rClosenessMatrix	2-dimensional; the rows and the columns are the entities' sequences; the number of one spot is used to measure the closeness of two entities.
cssmMatrix	2-dimensional; the data is the times of travelling among situations.

Table 5. Matrix Names and their Characteristics

5.1.5 Matrix Computing

Matrix storage and retrieving is convenient using the JSON related packages of the Java language. However, the processing of the matrix data for special purposes depends on the algorithms that need to be designed and explored.

CASM Matrix Reasoning

States are used to model context, meanwhile, this approach (CSM) of putting forward the states as the first citizen of modeling, support the calculation of probabilities through logics in the granularity of states and state transitions. To calculate the probability of transiting from a state S_x to a specific state S_y in a CASM (suppose the total number of states is n and the CASM matrix name is M; $M_{x\,y}$ refers to the value of the spot of row x and column y), the formula of the probability is:

Formula 1: $$P_{x \to y} = M_{x\,y} / \sum_{i=0}^{n} M_{x\,i}$$

A *"decisionMatrix"* is used for context reasoning about which decision a person will most probably take. The third dimension of the matrix is the decision types. To calculate the probability of a decision type is chosen when the entity transit from state x to state y (suppose the total number of decision types is d and the decision matrix name is D; $D_{x\,y\,z}$ refers to the value of the spot of (x, y, z)), the formula is:

Formula 2: $$P_{z|x \to y} = D_{x\,y\,z} / \sum_{i=0}^{d} D_{x\,y\,i}$$

The obtained probabilities can then be decided to be "Obvious" or not by compared with a threshold like "0.8". If a probability is obvious, then the related decision would most be probably taken by the examined person or other entities.

Relationship Representation and Registration

The situation of a person could include the friends', colleagues' or other related entities' current states. So, to extract a person's situation, it is necessary to have a data structure to store the relationship information. In our design, a relationship matrix is used to store this type of information. Firstly, we have an array of persons and each person has an index which is to be used as the coordinate of the matrix, and an array of relationship types. I use a 3-dimensional

matrix to store the data. The first and second dimension are the indexes of persons, and the third dimension is the relationship types.

If there is a new person encountered in the context, the CSM engine will register the person in the person array. Similarly, if there is a new relationship type encountered in the context, the CSM engine will register the relation type in the relation type array. Each of the above update types modify the relationship matrix of by adding a new element to their related dimension.

Using Relationship Matrix to Extract Situations and Build CSSM

To build the CSSM for a person, the system needs a mechanism to find the current situation and update the situation. A situation in our implementation is composed by the person's state and last state (transition), the person's related entities' state and last state (transition). To obtain the information of the latter, the system will process all triple inputs and for each triple, traverse the relations to see if the triple's subject has a relationship with the person, and if so, add the subject, its state and the latest transition to the situation. For example, we have a person P1, and get the first two features: P1.location.state & P1.location.transition. Using the two features, we can get the situation of the person: From the P1.relatinships matrix, we get a list of persons, and iterate each of them to get their states and transitions. Then the combination of their states and transitions composed a situation state.

If there are too many related entities for one entity, to refine the computing process in the future, we could only consider the closest entities' situations and the closeness could be defined using the time the entities spend together or other features.

The CSM engine maintains an array of situations, and for a new state of a situation we give the situation array a new number and let the new number map to the new situation. By this

way, we can build a CSSM using a mechanism similar to building a CASM, and the probability calculation method for a transition from one situation to another is similar to be within a CASM.

CSSM Decision Reasoning

Since the decision of whether taking or passing an elevator is related with a person's activities, and probably that the persons' activities are implicitly related with their related entities' status. Thus, we can explore the relationship between the persons' situations and their decisions, which is the basis for making decision reasoning using CSSM. To calculate the probability of a decision type is chosen when the entity transit from a state S_x to a specific state S_y in a CSSM, the method similar to Formula 2 (Suppose the decision matrix name is S*D*):

Formula 3: $$P_{z|x\to y} = SD_{x\,y\,z} / \sum_{i=0}^{d} SD_{x\,y\,i}$$

Supervised Learning Reasoning and More

After representation of the states and transitions using matrix, it is possible to use a supervised learning method to conduct context reasoning under the condition of using matrices to represent CSMs: the CSM mechanism supply more information to represent the states transitions, supervised learning methods can use them as one type of features to enhance the learning process. More research of using CSM method in supervised learning, unsupervised learning and other reasoning techniques is needed.

This computing and data storage method use numerals to represent the textual data and transform the relationships between these numerals into numerals of different matrices. In this way, plain JSON files can be used directly to store the contextual data requiring less storage space than a relational database.

5. 2 Efficiency of Supporting Context Modeling and Reasoning

The efficiency of using the state-based context modeling approach is to prove two facts. Firstly, the CSM engine can be used to model contextual information, and further support context reasoning for system decisions. CASM and CSSM are implemented to fulfil the requirement.

5.2.1 Applying the CSM Engine on the Base Case

The base case is an elevator control system with limited entities and constraints, which is described in subsection 4.3.1. The description for the base case is:

A Person visits several places one day and we can build a CASM from the location information. For each time associate with a location change, the person has possibilities to take the elevator or not in the destination building. For example, if a person arrives at a building from the recreational center, the person will not take the elevator; and if the person is from home, the person will more likely take the elevator. This information can be obtained from a decision matrix defined within the CASM object in the CSM engine.

To run the CSM engine on this base case, I firstly design the historic data so that the engine can use the data to build the CASM and the decision matrix for that person. In our implementation, the CSM engine can successfully build the CASM with the decision matrix. Secondly, the engine supports the access to the decision matrix to calculate the probabilities, according to Formula 2 of subsection 5.1.5.

5.2.2 Run CSM Engine on a Complex Case

A more complex case could be that we have two persons: a student and a professor, which case we have multiple entities and constraints (relationships) and make context computing for the student. We find their relationships array, and then using that array, we can identify the correlation between them. Using the states and transition information of both of them, we can

build a CSSM for the student and the decision matrix for the CSSM. The description for the complex case is:

Student S and professor P visit several places one day and we can build a CASM from the location information for each of them. The relationship (constraint) between S and P has already been defined in the context domain. There could be two situations: a) the student S arrives at a building while the professor P is at the building; b) the student S arrives at a building while the professor P is not at the building. For each situation, the person has possibilities to take the elevator or not in the current building. For example, if it is in the situation a, S will more likely take the elevator; and if it is b, the person will more likely not take the elevator. This information can be obtained from a decision matrix defined within the CSSM object in the CSM engine. It is not necessary for the designers of the system to consider these situations and the relationship between these situations and a person's decision beforehand, the implicit relation could be identified by using the state-based modeling and computing approach.

This case is fit for matrix computing for CSSM matrices, refer to the "CSSM Decision Reasoning" in subsection 5.1.5. In our implementation of the CSM engine, the situation state machine can be dynamically built and can consider the situations on behalf of the software designers. Most importantly, a decision matrix is built within the CSSM to support the context reasoning about deciding whether a person will make a decision or not.

Designers may not be able to capture the implicit context relationships beforehand but the CSSM has a mechanism to build the context relationship automatically and support context reasoning. The CSM engine is designed to have a module containing probabilities calculation interface for developers to use.

5. 3 Efficiency of Execution Timing

The second fact to prove is that the execution time of the modeling and the reasoning process is within a time budget that a context-aware elevator controlling system in literature can tolerate. Besides, if this part of the research will be used more in runtime module, and in runtime multiple scenarios can happen very soon. It needs a way to measure the timing efficiency. For example, if a person stops for a coffee, the CSM engine may calculate again. The context domain may involve a lot of people and it may need to calculate thousands of time at the runtime. The response time is limited, and it is required to respond simultaneously.

A Trace-driven simulation is conducted on CSM engine to show that the execution time is short enough for the CPT module to response to the ICU module. The simulation generates the time-consuming charts. Since the genetic algorithm is proved to be effective to optimize the algorithm selection, I do not need to consider this part and it is out of the research boundary. I only access the time-consuming part to show that it is effectiveness and it can detect that relationship (the context reasoning part) automatically, which is the most important part. The simulation is to feed mocked data into the CSM engine and use different amount of contextual data to generate the time-consuming table.

The execution results give researchers a hint about the time consumed to run CSM engine modules, including generation of contextual data using the utility module (Data), conversion from a text file with designed testing data to an object in Java and then to a JSON file (Conversion) and assembling context objects for different purposes (CASM/CSSM). The execution purpose includes testing for multiple entities, the effect of the amount of data for building CASMs, multiple constraints and the effect of the amount of data for building CSSMs given settled conditions. The execution environment is “Intel (R) Core (TM) i7-4790 CPU @3.60 GHz” and

the installed RAM is 8.00 GB. I run the CSM engine using Eclipse with heap size limit of 8.00 GB. To assurance the accuracy, I run each process triple times and utilize the average value.

In the Figure, the x axis is the total number of the testing data and the y axis represents the milliseconds spent by running a module. As we can see from the four figures (Figure 16, 17, 18 and 19): 1) the conversion of data spend more time than the other two processes; 2) the time spent for generation of CASM is steady when the data amount is in or less than 100k level, but it has a higher slope when the data is among 100k and 1000k level; 3) the most important finding is that when the data items is within 100k level, the time spent is within 1 minute in total for the 3 processes.

Multiple Entities

To represent multiple entities, if we have 10K (10,000) testing data, and the number of person entities are 5, 20, and 50, the execution result is shown in Figure 16. As the number of entities increases, the time consumed for data generation increases. There is a little decrease for the time consumed between 5 entities and 10 entities for the data conversion from raw to JSON triples, however, there is no obvious changes afterwards. The time consumed for constructing CASM from the triples is not affected by the number of entities, given a settled amount of testing data.

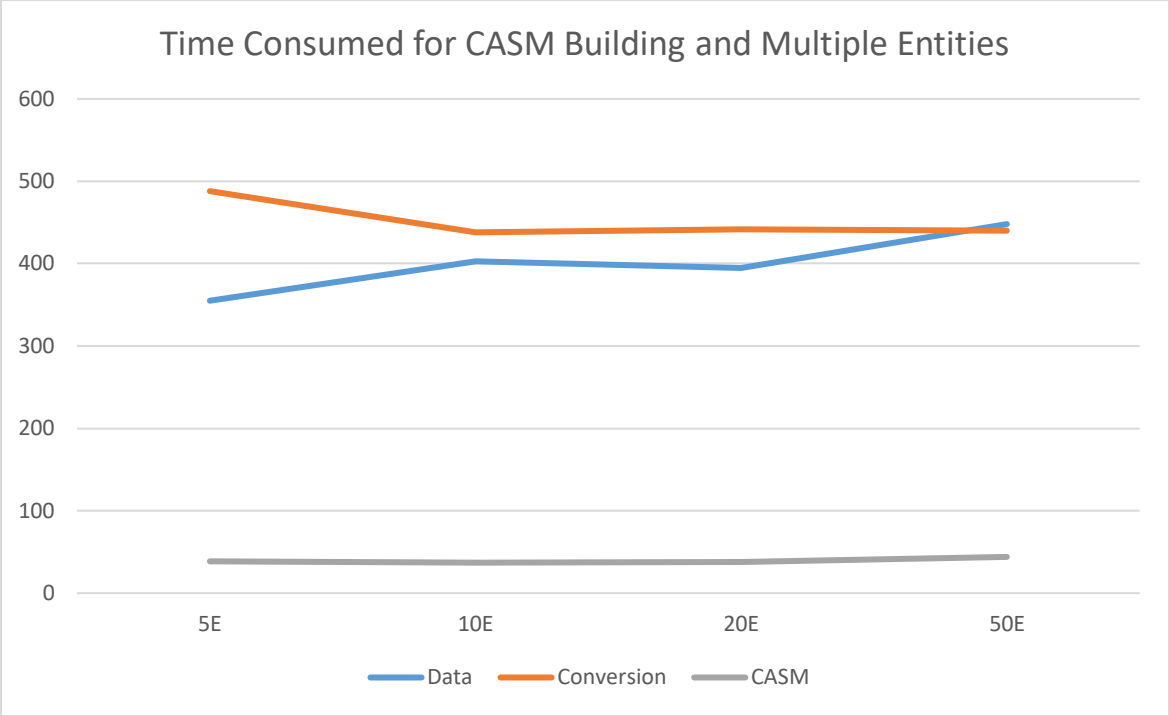

Figure 16. Time Consumed for CASM Building and Multiple Entities

Amount of Data for Building CASM

To see the effect from the number of data for building CASMs, if we have 50 entities, and the total number of data to process is 1k, 10k, 100k, and 1000k, the execution result is shown in Figure 17. The time spent for conversion is greater than the data generation and the CASM assembling for all the numbers of data. The difference becomes obvious when it comes to 1000k level.

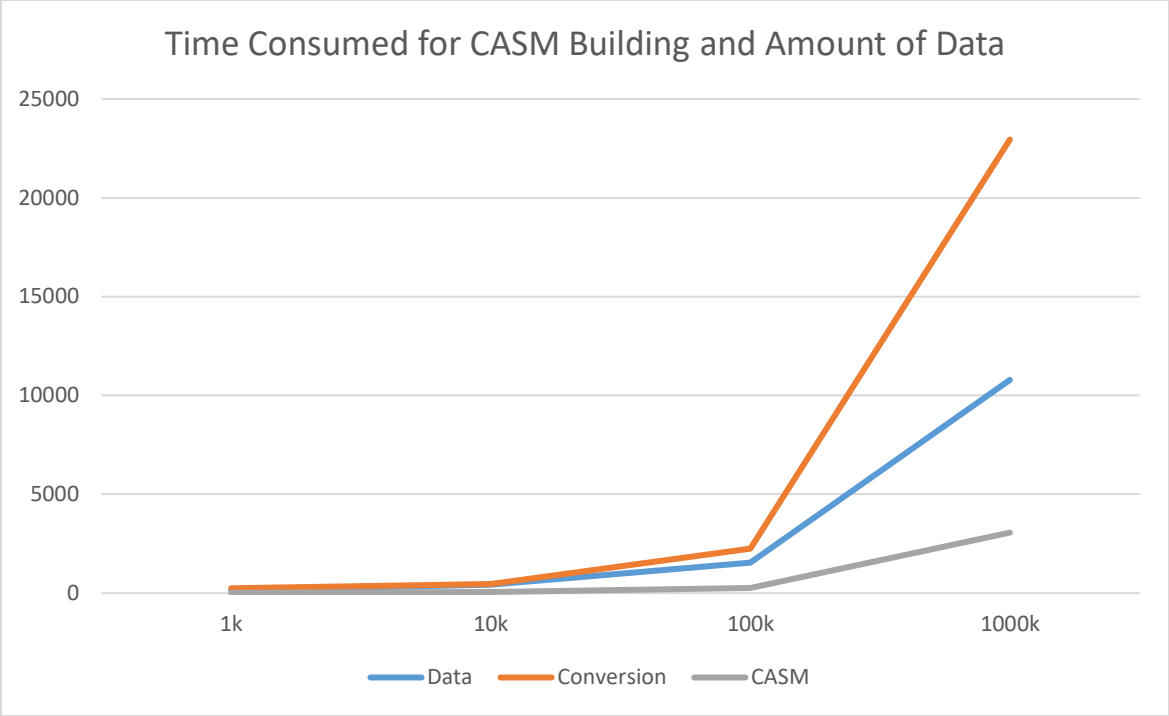

Figure 17. Time Consumed for CASM Building and Amount of Data

Multiple Constraints

Multiple constraints can mean there exist multiple relationships among the entities. The mapping relationships can be “one to many” or “many to many”. To represent multiple constraints, we suppose a professor advise one or more students so that the professor context object will contain situations involving the states of all related students. If there are 50 entities, 10k testing data, and the relationship constraints is: a professor advises 1 student, 5 students, 10 students or 20 students, the execution result is shown in Figure 18. It shows that as the number of constraints increases, the time spent for building CSSM increases. The slope of the line is relevantly small might be related with we use the mapping relationships “one to many”. The line will be much steeper if we use the “many to many” mapping.

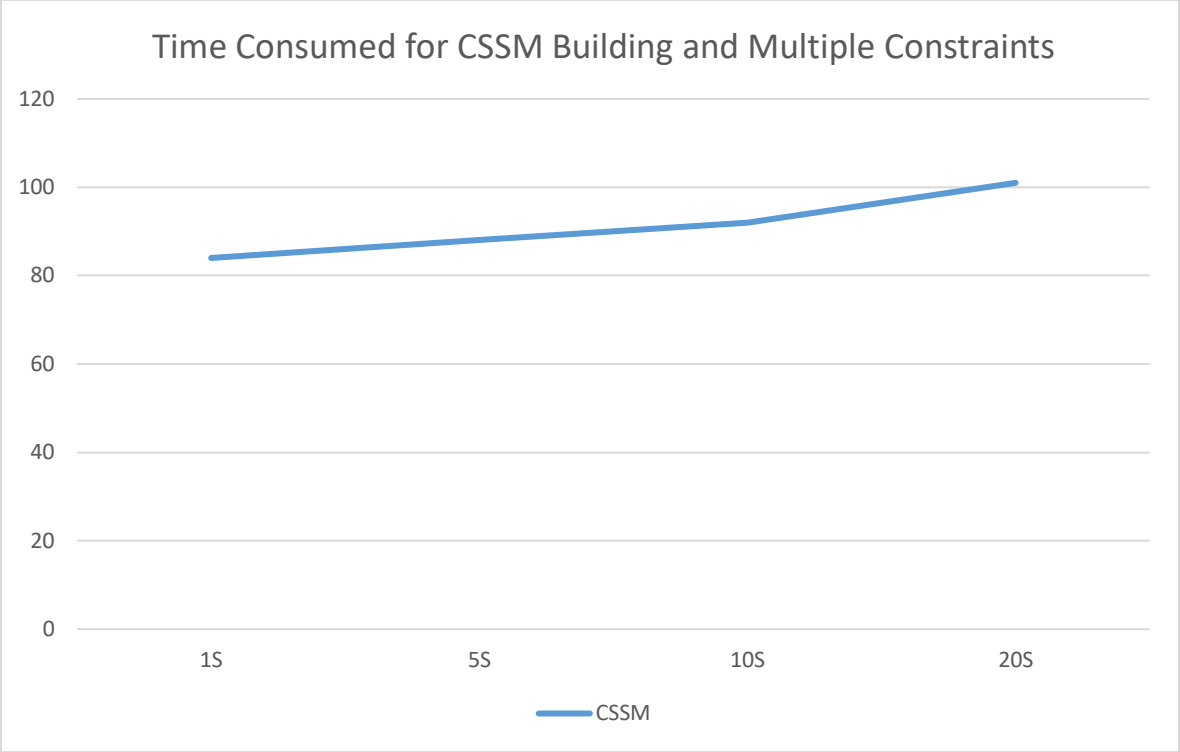

Figure 18. Time Consumed for CSSM Building and Multiple Constraints.

Amount of Data for Building CSSM

To see the effect from the number of data for building CSSMs, if we have 50 entities, there exist a professor who advises 5 students, and the total number of data to process is 1k, 10k, 100k, and 1000k, the execution result is shown in Figure 19. Compared with the time consumed for building CASMs, the time for building CSSMs is greater for each level of the amount of the data. When the data is within 100k level, the time for building CSSMs is within 1 minute and acceptable, however, when the data is in 100k level, it is more than 10 minutes, so we just put a fake time 100,000ms in the diagram, which requires a better machine to run the engine or optimizations of the system.

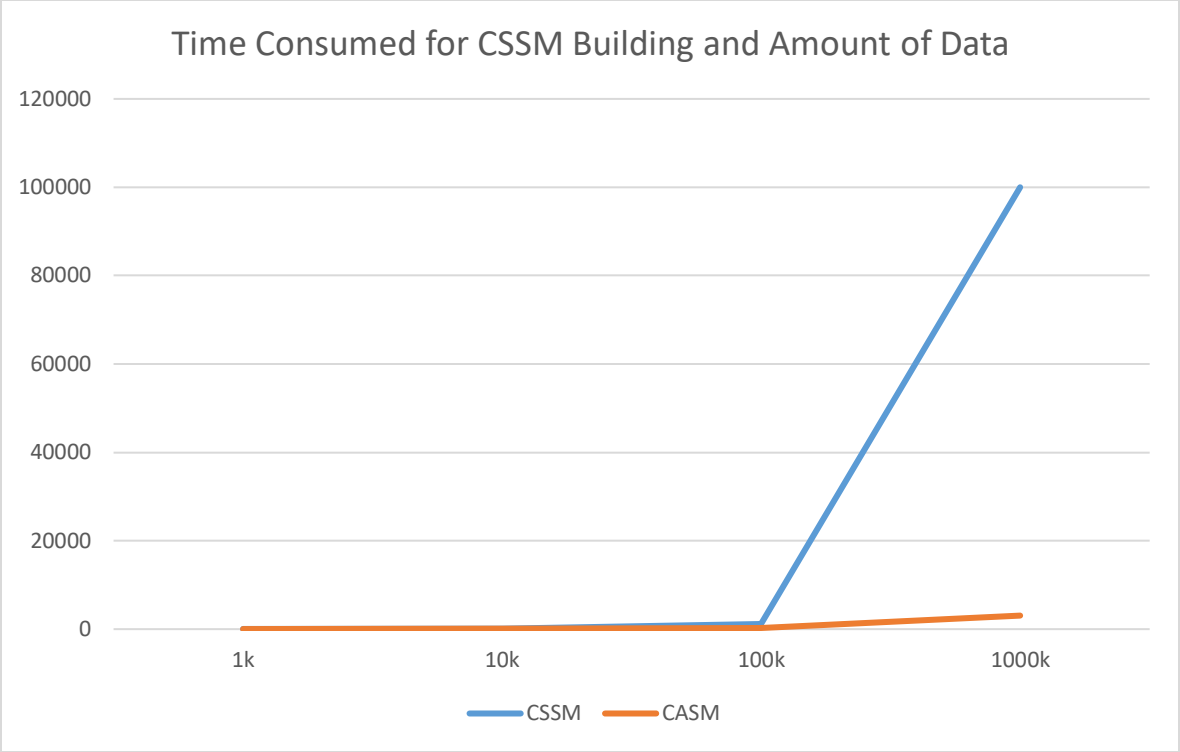

Figure 19. Time Consumed for CSSM Building and Amount of Data

5.4 Conclusion

This chapter introduces the detailed design and implementation of CSM engine, especially the matrix computing and reasoning. By applying CASM reasoning and CSSM reasoning, the CSM engine can successfully model and reason on a basic context-aware case and a complex context-aware case of an elevator system; and to answer the research questions 2 and 3, the chapter presents the time consumption of CSM engine execution to show the timing efficiency of this implementation and the approach.

Through converting the CASMs to matrices, the reasoning of context has been transformed to numeral computing, and the reasoning engine does not need to know the meaning of the states. This can improve in some extent the interoperability of context during the context

reasoning. To model the different domain with various entities, this method or implementation can be extended, and the data logic can be represented and conveniently accessed through the framework, then the results can be used directly to guide the decision of applications.

CHAPTER 6

CONTEXTUAL DATA FOR TESTING CSM: FROM RAW TO SYNTHESIZED

In our research of evaluating the CSM engine, I use synthesized (high-level) contextual data to feed and test the CSM engine; however, in the wild, most of the initial contextual data are raw data from the sensors or applications. The raw data has some characteristics such as inaccurate or inconsistent. To let those data be used by various context-aware systems and middleware, synthesized data should be developed based on the raw data.

The Research Question 4 is "Can synthesized data for context aware systems be developed addressing the inaccuracies and poor quality of data expected from real-world sensors?" In order to address Research Question 4, this chapter introduces the methods and rules of generating synthesized contextual data of elevator domain for testing CSM engine, surveys and mediates the challenges of using contextual data, and develop rubrics for generating synthesized context regarding the data quality and dimensionality. I briefly described the data flow of CSM engine in Chapter 5. In this Chapter, I describe the important implementation details in the data flow.

6.1 Contextual Data for CSM

Context data, as demonstrated in the Figure 1 of Chapter 2, includes data from sensors, data from devices, and internal states of an application. Before dispatching to a context-aware software, the context data may be processed by a *Context Collector and Analyzer* to a level that

can be passed to a *Device Context Manager*. When testing the CSM engine, I consider what kind of data should be accepted from the outside and how to design the data so that it could facilitate the building of CSMs. I divide the data into three stages of "Raw", "Triple", and "Context Object/CASM/CSSM", and developed a program module specialized for generating reasonable contextual testing data to feed the CSM engine, which can use the data to generate the CASMs or CSSMs and then persist them. In this subsection, I describe data examples and the methods and rules to process them.

6.1.1 Raw Context

The original context data is from the sensor; however, it is not applicable because most of them are raw data and needs preprocessing. Raw data refers to original observation or records such as the camera signals, Wi-Fi access point, and RFID records. To identify that a person is in a place, different sensors could be used [3][58]. If different sources pass different location information for a person, the information with higher degree of belief should be selected. These procedures are out of our research domain. I suppose the information such as the *location* or *action* can be obtained from a *Context Collector and Analyzer* or *a Device Context Manager*.

According to Dey and Abowd [30], there are certain types of context that are, in practice, more important than others, namely, *location*, *identity*, *activity* and *time*. These four items are also the key context attributes of the context-aware elevator system. As illustrated in Table 6, I mocked the passengers' identification, location, time, and instead of activity, I mocked the action and decision.

Id	pId	pName	time	action	actionId	locId	locName	decision
0	p002	Frank	17-Jan-18 03:04:19	arrive	action02	location03	HOME	TAKE
1	p005	Alice	17-Jan-18 03:38:31	arrive	action02	location06	COFFEESHOP	TAKE
2	p003	Joshua	17-Jan-18 04:38:23	arrive	action02	location05	DINNINGHALL	TAKE
3	p001	Mike	17-Jan-18 05:28:14	arrive	action02	location04	GYM	TAKE
4	p002	Frank	17-Jan-18 06:13:26	leave	action01	location03	HOME	PASS
5	p002	Frank	17-Jan-18 06:56:15	arrive	action02	location06	COFFEESHOP	TAKE
6	p002	Frank	17-Jan-18 07:37:11	leave	action01	location06	COFFEESHOP	TAKE
7	p002	Frank	17-Jan-18 08:25:31	arrive	action02	location04	GYM	PASS
8	p002	Frank	17-Jan-18 09:17:29	leave	action01	location04	GYM	TAKE
9	p005	Alice	17-Jan-18 10:10:17	leave	action01	location06	COFFEESHOP	PASS
...								

Table 6. Mocked Contextual Information (Also used in Table 3).

Each person in the table will be treated as a context object in the CSM engine. The synthesized data tuples are transferred to triples, based on which context objects can be created and updated. In the table above, the “pName” column refers to person’s name, the “locName” column refers to location name, the “action” column refers to the action a person with respect to a location, and the “decision” column represents the action taken by a person when the person reaches a smart elevator.

In the mocked data, I assign IDs to persons, locations and even actions to uniquely identify them in a context domain. So that if two persons have a same name, they still will be treated as two entities instead of one by referring to the IDs. Similarly, the IDs can help to uniquely identify the actions and locations. The items in the mocked data are supposed to be received by the CSM engine sequentially. Several rules are followed for designing the data which can simulate the context in the real scenarios. These rules assure the data consistency and accuracy.

1) The initial action for a person is to arrive a place. When a person is detected in a building or a place, the first message will contain an action of "arrive". If the person is detected to leave a place, then a new message containing "leave" action will be generated.

2) A person needs to leave a place before arriving a place.

3) There is a random time slot between the two messages for a person.

4) The decision whether the person will "take" or "pass" an elevator is generated randomly in this testing data.

6.1.2 From Raw to Triple

A Triple is composed of a subject, a predicate and an object. I extend the Triple in our conceptual design of CSM engine to make it contains decisions and conditions. Conditions and Decisions can recursively use triples to represent the condition details. A normal condition could be a timestamp, and a decision can only contain a name. Figure 20 shows an example of a triple information in the implementation.

The triple is built from a tuple of Table 9. The subject is a person. The object is the places the person leaves or arrives. The predicate includes two types: arrive or leave. The time condition shows that there is a period between each action and I use the decision as an important contextual information complement to make reasoning afterwards.

```
"infoList" : [ {
  "subject" : {
    "subjectType" : "person",
    "subjectURI" : "p002",
    "subjectName" : "Frank"
  },
  "object" : {
    "objectType" : "location",
    "objectName" : "HOME",
    "objectURI" : "location03",
    "fromTo" : null
  },
  "predicate" : {
    "predicateType" : "action",
    "predicateURI" : "action02",
    "predicateName" : "arrive"
  },
  "condition" : {
    "time" : "17-Jan-18 03:04:19"
  },
  "decision" : {
    "decisionName" : "TAKE"
  },
  "rid" : "0"
},...
```

Figure 20. An example of contextual information in Triples

6.1.3 From Triple to Context Object

Although the ultimate goal of the Triple data is to be transferred to context objects with CASMs and CSSMs, the Triple data is also responsible for creating or updating the whole context domain. So, I set the context domain information at the head of a Triple file. In the example of the Figure 21, the domain name and the domain URI is given so that the system can know which domain the file is related with.

```
{
  "infoType" : "any",
  "deviceType" : "smartphone",
  "deviceName" : "any",
  "deviceId" : "android",
  "domainName" : "Smart Campus",
  "domainId" : "5",
  "domainURI" : "test",
  "infoList" : [ {
    "subject" : {
      "subjectType" : "Student",
      "subjectURI" : "p004",
      "subjectName" : "Ted"
```

Figure 21. Head Information of a Triple File

Under each domain, there is a list of context category. Each domain is related with an ontology model which contains a list of classes. To differentiate with the term “class”, I use category. As shown in the Figure 22, for each context category in a “ccl” (Context Category List), the category type is the “class name”, and the category name can be related with a “sub-class”. The category name is mapped with the subject type in the Triple’s subject. For example, the subject type “Student” in the Figure 21 is mapped with the category name “Student” in the Figure 22. Under each category, if there is a new subject occurred, I add a new context object for it. The new object’s name is the subject name and the new object’s ID is the subject URI.

In the example of the context-aware elevator, I treat each person as a context object (entity). The location of a person is considered as an attribute of the person entity. Different locations are the states of the location attribute.

```
{
 "domainName" : "SmartCampus",
 "domainId" : "1",
 "domainURI" : "test",
 "ccl" : [ {
    "categoryType" : "Place",
    "categoryURI" : "cpl001",
    "categoryName" : "Office"
  },
  {
    "categoryType" : "Place",
    "categoryURI" : "cpl002",
    "categoryName" : "Classroom"
  },
  {
    "categoryType" : "Person",
    "categoryURI" : "cpe001",
    "categoryName" : "Student"
  },
  {
    "categoryType" : "Person",
    "categoryURI" : "cpe002",
    "categoryName" : "Professor"
  },
  {
    "categoryType" : "Device",
    "categoryURI" : "cd001",
    "categoryName" : "SmartElevator"
……
```

Figure 22. The Data for Domain Initialization

A Context Object is composed of the basic identification information, a list of context attributes, and a list of CASMs etc. Each context attribute is mapped to a CASM. The algorithm to transfer Triple data to the data of context objects in pseudo code is as follows:

Initialize the domain if the domain hasn't been initialized;

FOR there is a new triple data in the file;

Find the category name in the domain by subject type in Triple file;

IF the current triple data contains a new subject, add it as a new context object in the category;

ELSE update the related context object;

ENDIF;

ENDFOR;

The implementation details of how to process the triple data for building CASM and CSSM can be find in Appendix A. An example of a CASM in a data file can be found from the Figure 15 in the subsection 5.1.3. To build the CSSM, I also designed a triple for relationship registration, as shown in Figure 23. An entity's situation state is composed of its current attributes' states and its related entities' attributes states.

```
"infoType" : "relationshipRegistration",
"deviceType" : "smartphone",
"deviceName" : "any",
"deviceId" : "android",
"domainName" : "Smart Campus",
"domainId" : "5",
"domainURI" : "test",
"infoList" : [ {
  "subject" : {
    "subjectType" : "Professor",
    "subjectURI" : "p001",
    "subjectName" : "Mike"
  },
  "object" : {
    "objectType" : "Student",
    "objectName" : "Ted",
    "objectURI" : "p004"
  },
  "predicate" : {
    "predicateType" : "relationshipRegistration",
    "predicateURI" : "relation01",
    "predicateName" : "Advise"
  },
  "closeness" : 80,
  "rid" : "0"
}
]
```

Figure 23. An Example Data of Relationship Triple

6.2 Challenges of Using Contextual Data

In this section, I survey and meditate the challenges of using context data, especially in the perspective of testing. First of all, I survey and discuss the quality of contextual data. Secondly, I introduce the testing adequacy criteria of context-aware system. Context-aware adaptation refers to the ability of computing systems to adapt their behaviors or structures to highly dynamic environments without explicit intervention from users, with the ultimate aim of

improving the user experience of these computing systems [26]. Since adaptation is a core process of using context data for computing in context-aware software, I introduce the challenges related to adaptation, including Erroneous adaptation rules and continuous adaptation. Finally, I discuss the challenges of generating context for testing and introduce an open topic that new mechanisms are necessary for facilitating testing execution.

6.2.1 Quality of Contextual Data

Wang et al. [82] argue that the added capabilities of context-awareness introduce a distinct input space. Since context changes can affect software behavior at any point during the system execution, context as system data or testing data should be well studied and selected. However, context data retrieved from sensors usually have such characteristics as being inaccurate, inconsistent, and continuous which may increase the difficulty in selecting testing data.

Context Inaccuracy:

Sensor data can be inaccurate [76]. Such data should be well studied before using for testing. Traditional testing methods usually use accurate values as test cases. However, for testing context-aware applications, especially those obtaining data directly from sensors, it is reasonable for testing engineers to question the reliability of the data.

Vaninha et al. [76] illustrate the relationships between the context sources (sensors or software) and defect patterns. They show that context sources are closely related to faults of several types: incompleteness, inconsistency, sensor noise, slow sensing, granularity mismatch, problematic rule logic, and overlapping sensors. Each fault type is caused by one or more failures in context sources, such a Camera, GPS, or Wi-Fi. Table 7 (borrowed from [76]) shows the relationship between context sources and fault types, e.g., ambiguity, as one form of

incomplete, may be caused by errors in the context source of RFID/NFC, QR-CODE or Clock/Alarm.

	Incomplete			Sensor Noise				Slow Sensing		Overlapping	
Context-	Unavailability	Not Interpretable	Ambiguity	Incorrectness	False Reading	Instability	Unreliability	Out-of-Dateness	Wrong Interpretation	Concurrent Values	Unpredictable
Accelero meter	X			X	X			X	X		
Wi-Fi	X						X	X	X		X
Camera	X	X						X	X		
RFID/NF C	X	X	X	X	X						
QR-Code	X	X	X	X	X						
GPS	X			X	X		X	X	X		
Light Sensor	X			X	X			X	X		
Clock/Ala rm	X		X					X	X		
Calendar	X			X	X			X	X		
Gyroscope	X							X	X		

Table 7. Context-Sources in Combination with Defect Patterns [76]

The problem of inaccuracy in context data can cause a high-level defect called context inconsistency, which may relate to multiple context sources or is a defect in interpretation from context [76].

Context Inconsistency:

Context inconsistency occurs when there is at least one contradiction in a computation task's context [15]. It can be caused by sensor errors or sensor data inaccuracy [25] [34] [39] [76]. Asynchronous updating of context information can also cause the same problem [11]. As a result of the possible inconsistency of context, the application logic that rely on the context can lead to wrong behaviors or execution errors.

Context contradiction can be illustrated using the Wi-Fi access point (WAP) application where WAP can be used to detect the location of a device connected to it [25]. Suppose in a location identification service, WAP installed in each room of a building is supposed to detect the location of a person who is wearing a smart device. The smart device has a unique identification for each person. Context inconsistency may happen in the following situation: if WAP S1 installed in room R1 detects person P and claims that P is in R1 now and meanwhile WAP S2 embedded in room R2 detects the same person P and claims P is in R2. This type of inconsistency can happen in the following scenarios: rooms R1 and R2 are near each other or they are in the same coordinates of nearby floor.

Context-aware applications can get raw data from a single sensor or several sensors, and they can also get synthesized context data from middleware [70], which collects data from sensors as well. Raw data from a single sensor have great opportunities to exhibit inconsistency problems, however, data from a middleware, which does not apply consistency checking, may also experience inconsistency problems.

Chang et al. [15] try to solve the inconsistency problem using a framework for realizing dynamic context consistency management. Based on a semantic matching and inconsistency-triggering model, the framework can detect inconsistency problems. The framework also applies inconsistency resolution with proactive actions to context sources.

Continuous Context:

Continuous context is used in many context-aware applications [70]. Challenges may arise when applying boundary testing in a continuous context. A straightforward way of modeling continuous context is to directly convert it into discrete one by dividing it into different time windows [9] [70]. Hidasi et al. [9] demonstrate that much information will be lost if such modeling approach is used. This missing information can be the boundary values, which will greatly affect the effectiveness of testing with the technique of boundary value analysis. To build better models for continuous context, Hidasis et al. propose fuzzy modeling approaches. The fuzzy modeling method advocates that context-state is not only associated with the interval it belongs to but is also influenced by its relative location in the interval and neighboring intervals. Thus, a better understanding of the event or context-state with respect to a specific interval can be achieved, in which way the information loss of boundary values can be complemented.

For context data collection, Chen et al. [63] define snapshot as the union of all sensing values at a particular timestamp. The act of collecting multiple continuous snapshots is called continuous data collection (CDC) [62]. Their work focuses on challenges of network capacity, while Nath's [70] work concentrates on reducing sensing cost using a middleware approach when continuous context sensing is required.

6.2.2 Adequacy Criteria for Testing Context-aware Software

Testing adequacy criterion is usually defined as a rule or a collection of rules a test set should satisfy [57]. To measure how well a program is examined by a test suite, usually one or more criteria are used. A variety of testing adequacy criteria have been developed for traditional testing while only a few are suitable for testing context-aware software. According to the work of Lu et al. [33], there are three kinds of obstacles that hinder the effective application of standard data flow testing criteria to testing Context-aware Middleware-Centric (CM-Centric) software, namely,

1) *Context-aware faults*: faults in the triggering logics in the middleware;

2) *Environmental interplay*: environmental updates may happen anytime, and test set should be updated in time accordingly;

3) *Context-aware control flow*: it is difficult to enumerate every control flow trace of context changing for some situations.

Recent research is using special approaches to generate testing criteria for context-aware software [33] [34]. For instance, Lu et al. [33] have applied a data flow method to generate adequacy criteria for testing middleware-centric context-aware programs. Different from traditional variables, a context variable can be defined and updated via either an assignment or an environmental update. Therefore, a new definition of "definition (DEF) of variables" and "usages (USES) of variables" are given, as well as "update-use occurrences of variables", which refers to an occurrence of a context definition due to sensing of environmental contexts and a context use. Imitating the conventional def-use (DU) associations, the paper provides definitions of def-use associations for CM-Centric programs, as well as a definition for the pairwise DU

associations. Using the defined data flow associations, they generate novel test adequacy criteria to measure the quality of a test set for a CM-centric program.

6.2.3 Context-aware Adaptation

Context-aware adaptation refers to the ability of computing systems to adapt their behaviors or structures to highly dynamic environments without explicit intervention from users, with the ultimate aim of improving the user experience of these computing systems [26]. Context can be used by software through triggering the context adaptation rules. Adaptation rules, which are usually maintained, evaluated and applied by adaptation manager of a context-aware system, define a significant portion of an application's behavior [51]. We can use an example of a car system to illustrate how an adaptation rule works. Suppose a car installed with an Autonomous-Driving System (ADS) needs to change lanes. The adaptation rules in ADS need to assure that the car can take this action only if the current context is safe for changing lanes. There should be some additional rules to define what is safe in a real driving environment, which ADS can use to check the safety. If ADS knows the context is safe, it will choose a way to react according to some other rules: changing to left lane or changing to right lane, and in what speed.

Adaptation Challenges:

Adaptation is the core process of using context for computing in context-aware software. In this subsection, I introduce Erroneous adaptation rules and continuous adaptation.

Adaptation rules can be erroneous. Realizing that adaptation rules play an important portion in middleware-based context-aware applications, the work of Sama et al. [51] is focused on fault detection in adaptation rules. In their approach, detection is driven by the requirement that the rules and its finite state machine satisfy the following properties: Determinism, State Liveness, Rule Liveness, Stability, and Reachability. For example, determinism requires that for

each state of the finite state machine and each possible assignment of values to the context variables in that state, the assignment of the value can only trigger at most one rule.

Continuous adaptation makes it hard to identify which adaptation rule have caused the faults, so it is difficult to set up an effective test oracle [17]. Xu et al. [17] suggest that for context-aware applications, the adaptation to the environmental changes may contain defects when the complexity of modeling all environmental changes is beyond a developer's ability. Such defects can cause failures to the adaptation and result in application crash or freezing. More importantly, they argue that tracking an obvious failure of the system back to the root cause in adaptation is generally difficult [17]. The reasons are as follows. Firstly, a failure is usually a consequence of multiply adaptations, and it is difficult to set up an effective test oracle. Secondly, when a failure happens, it is hard to collect all the context data because some of the data are from outside sensors. Thirdly, it is hard to repeat an observed failure. In their work, they propose a novel approach, called ADAM (adaptation modeling), to assist identifying defects in the context-aware adaptation.

6.2.4 Testing Execution

Testing execution refers to the process of executing a test plan, in which all the challenges meditated in above should be considered. It not only needs to consider making test plans to resolve aforementioned challenges, but also to realize them by creating novel tools or mechanisms. In this subsection, I discuss the challenges of generating context for testing and introduce an open topic that new mechanisms are necessary for facilitating testing execution.

Context Testing Data Generation:

Context can be complex and plenty of work has concentrated on context testing data generation. Two approaches can be used to provide context test information: real-world testing

and simulator testing. Real-world testing means to evaluate an application in real devices with multiple sensors and network conditions. Repeated real-world testing can be expensive in time and effort, sometimes even infeasible when context and environment are complex, e.g. aerospace. However, real-world testing is still highly recommended before the acceptance or commercialization of an application.

Simulator testing can be an alternative when real-world testing is expensive or unpractical, and it is a frequently used approach [27] [52] [67] [76]. Designers need a set of models and tools that aim to achieve the objective of "design for reality". In real-world, as I have discussed, context derived from sensors can be inaccurate, inconsistent, and continuous. Besides, sensor reading and network connections may strongly depend on the providers of sensors and networks. Thus, it is very challengeable to build a well-equipped simulator. Eleanor et al. [27] propose a testing platform for the user-centered design and evaluation of context-aware services by using a 3D virtual reality simulation to show the environment to users and generate the simulated environment's context. They recognize that to simulate the sensors is very difficult.

Adoption of New Mechanism:

Some new mechanisms have been adopted to facilitate context-aware software testing. Griebe et al. [75] use model transformation approach on context-enriched design-time system models to generate platform specific and technology specific test cases. For fulfilling testing criteria, Wang et al. [82] use a component of Context Interleave Generator to form potential context interleaving that may be of value a context-coverage criterion requires.

When an observed failure happens, repeating it is a common method to track to its original defect. Collecting all the runtime information can help to achieve this purpose. However, when data is from outside sensors, the task can be difficult [17]. Asynchronous updating of

context information can also lead to inconsistencies between external states and internal states. To our best knowledge, these problems have not been thoroughly discussed and new methods for resolving them needs to be explored.

6.3 Rubrics of Generating Synthesized Data for Testing Context-aware Systems

I discussed in previous subsections the general issues that need to be considered on using the contextual data in real-world context-aware software, including in context adaption, context-aware middleware and testing of context-aware software. The data from real-world sensors could be inaccurate and in poor quality.

During the design of the data for CSM engine, I examine those issues, and conclude several rubrics that need to be followed for the generation of high-level contextual data for the purpose of testing context-aware systems or middleware. The data quality and dimensionality rubrics are built to support for testing (simulating real-world data), reasoning (*consistency* and *accuracy*), data sharing (domain, *granularity*, *meaning and identification*), interoperability (*accuracy* and *identification) and* privacy protection (*identification* and *numerical*).

6.3.1 Data Quality Rubrics

Table 9 presents the rubrics regarding the data quality issues. The explanation for each feature is given. The features and purposes are related with the requirements discussed in our dissertation.

	Meaning	Purpose
Domain	Different Domains may share same context object name or attribute name, to avoid this, data should be designed with a domain	Data sharing; Interoperability;
Granularity	To share data through multiple applications or domains, the data should be in the same granularity level;	Data sharing; Interoperability;
Meaning	The words used in the context should have the same meaning so that it is easy to share data with other applications, and another domains' knowledge. The format of the data should be contracted.	Data sharing;
Identification	If a context is a common one, it can use the same URI. The identification can be designed especially for interoperability.	Data sharing; Interoperability; Privacy;
Consistency	If an entity is reported by one sensor or application in the one location during a time period, the other sensors or applications should report a same location during that same time period.	Reasoning;
Accuracy	The accuracy of data can support stricter computing and assure the consistency.	Reasoning; Interoperability;
Numerical	The data can be modeled with numerical, which could help with the data privacy mechanism.	Privacy;

Table 9: The Context Data Quality Rubrics

6.3.2 Data Dimensionality Rubrics

The rubrics regarding the context data dimensionality are demonstrated by four most commonly used context features [30]: "time", "location", "identity", and "activity". Each dimension is related with different data quality rubrics (Table 10).

	Meaning	Related Quality Features
Time	If a person arrives at a building at minute level, then the time should not be represented in a day level. Otherwise, the accuracy is lost, and it would be hard to support for reasoning. If a time sensor reports the person arrive at a building at one time slot and another time sensor report anther time slot, it will cause the reasoning logic to make wrong decisions and give pressure to the coding part, the coding part need to use more mechanisms to assure that there is no consistency issue.	Consistency; Granularity; Accuracy;
Location	If a person is in one building, however, another sensor or application reports that the person is in another building, it causes a consistency problem. A person's location can be described as in a province or in a country, which involves different location granularity. A location can have a unique URI.	Consistency; Granularity; Accuracy; Identification;
Identity (Entity)	Each entity in a context domain should have a unique identification. If the entities in a domain can be represented using numerals, like the mechanism of CASM, the privacy mechanism could be explored as discussed in Chapter 4.	Domain; Numerical; Identification;
Activity	An activity may be from different domain. For example, eating, a person's eating is different with a fish's eating. An activity usually is a synthesized contextual information from analyzing data of different sensors or applications. Each activity can have a unique URI for data sharing and interoperability.	Domain; Meaning; Identification;

Table 10: The Context Data Dimensionality Rubrics

6.4 How Our Testing Data Apply the Rubrics

This subsection describes how the testing data generated for CSM engine apply the rubrics of dimensionality and quality. The features of the testing data are *time, location, entity and action*.

1) ***Time***: the generated times follow the same data format so that it is easier to make comparison in the same *granularity*. The data provides a time interval between a person leaving a building and arriving at another building, and the time remaining in a building. This makes sure that the times are *consistent* with the timing of the real world, avoiding inconsistency such as a person leaving a building before the arrival time.
2) ***Location***: all locations are using *accurate* building names of a domain and they are in the same *granularity*. I assign each location a unique *identification* so that it is still practical if two locations in a domain share a same name as: coffee shop.
3) ***Entity***: In our testing data, the entity is a person. I give each person a URI (Uniform Resource Identifier) in case two persons in the domain share the same name (*identification*). After transferring the testing data to CASMs and CSSMs, an entity in the testing data could be represented by a numeral: the index of the entity in the context object list (refer to the example data of the CASM).
4) ***Action***: I use action in the testing data to represent the activity. The same as for entity and location, I assign each action a URI in case different actions share a same name (*identification*). Sometimes a car's running action is different with a person's running.

6.5 Conclusion

In this Chapter, I firstly describe the testing of CSM engine, which requires a process of transmitting from raw contextual data to context object in the engine. Secondly, I survey the challenges of using contextual data, especially in the perspective of testing context-aware software or middleware. Thirdly, I generate rubrics with respect of data quality and dimensionality to guide the generation of testing data for context-aware systems addressing the

inaccuracies and poor quality of data expected from real-world sensors. At last, I discuss how our design of testing data apply the rubrics.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Context-aware computing and context-aware software is an important research field especially with the pace of advancement in sensor technologies. Context-aware computing has many challenging issues. This dissertation addresses the context modeling for dynamic and implicit contextual information needed to support semi-automatic context reasoning. I proposed a state-based modeling and computing approach, designed a CSM engine, and used the contextual information of context-aware elevator scenarios to evaluate the CSM approach.

7.1 Contributions

In this dissertation, I presented CSM, a state-based approach to support context modeling and computing. The CSM treats state as a first-class entity for the modeling of context and allows data associated with the state allowing context reasoning. The implementation of a CSM engine demonstrates efficiency in terms of reasoning allowing the approach to scale. The primary contributions of this dissertation provide answers to the following research questions:

RQ1) Can states be used for modeling context?

States are demonstrated to be a rich model for context. This research provides a state-based context model and builds a mealy state machine style model, termed a Context State Machine (CSM), capable of modeling simple and complex contextual situations with data from multiple sensors involving multiple entities and constraints.

RQ2) Can CSM model for context-aware software development be used to demonstrate effectiveness for known context-aware problems from the literature?

The CSM model demonstrates effectiveness in a known context-aware problem from the literature: context-aware elevator. This research employs a CSM engine for the base situation of an elevator control system with limited entities and constraints supporting modeling and context reasoning. Thereby demonstrating the efficacy of using state machines to represent and reason regarding context.

RQ3) Can CSM model for context-aware software development scale to multiple entities and constraints?

This research extends the base situation from Research Question 2 to an elevator control system scenario with multiple entities and constraints. This research demonstrates that the CSM can be used to model context in complex scenarios and support the context reasoning across multiple entities, goals, and constraints.

RQ4) Can synthesized data for context aware systems be developed addressing the inaccuracies and poor quality of data expected from real-world sensors?

In order to address Research Question 4, synthesized data for context aware systems was developed addressing the inaccuracies and poor quality of data expected from real-world sensors. Data quality and data dimensionality rubrics are developed for context data in general. The rubrics are applied in the generation of synthetic data for the elevator control contexts in this dissertation.

7.2 Limitations and Future Work

In this subsection, I discuss some limitations of the research and discuss future work. Some future work is based on the implementation details and new methods.

7.2.1 Problem Space

Although the CSM approach can be used to model dynamic context information and the transitions among states of context attributes enhance the expressive force of this modeling method. I evaluate the CSM using scenarios for a context-aware elevator. While the context-aware elevator example is taken from the literature, the CSM method may only fit certain domains or situations. The measurement of fitness not may be related to the size of the domain, the frequency of updates, quantity of entity attributes, and, most importantly, the characteristics of the attributes. The CSM approach performs well with attributes that have steady states and recognizable patterns in the states and transitions. For other more dynamic domains, more research is needed on the fitness of the CSM approach.

In this research, I mainly use the states for locations. I can explore using states to model more human factors, such as persons' mood, emotions, opinion types, opinions, and so on. If the relations between those human factors and some situations are known, for example between a situation of healthy life style and an emotion, then the knowledge can be shared by the people who share the similar characteristics. As a result, it can lead to more context sharing.

7.2.2 Numbers of Entities and States

As discussed in chapter 3, as the numbers of entities and their states increase, it can cause exponential growth in the system. Three approaches can be taken to offset some of the growth. Firstly, more modular, hierarchical and well-structured state diagrams can be applied. Secondly, if an entity has many relationships, the system can prune the relationship graph to only consider the closest entities' situations with the closeness defined using the time the entities spend together or other features. Thirdly, if we focus on the most valuable states and their inter-relationships, the number of context states for some objects can become steady. Like a person

(object) can have locations (attributes) of "usually to go," it can only take values like office, lab, home, and gym.

7.2.3 Usage CSMs on Social Media Data and Big Data

In recent years, there has been a lot of research in the field of analyzing social media data to generate useful insights for various purposes [1] [41] [53]. Some research includes sentiment analysis, many researches use data analysis for detection of disasters such as "earthquake" and "landslide". As our work about finding the implicit relationship between hurricane severity and twitter data implies [64], social media data can be a reference for determining social activity status. This kind of relations makes social media data as a source of context for social activity, so that it can be a source of context for social-service related context-aware applications.

Implicit relationships in big data from various sources can be learned and extracted as context for usage by context-aware applications or middleware. To the best of our knowledge, big data learning for serving as context for context-aware computing has not been thoroughly explored and discussed. On one hand, context information within data (such as location and time) can be used in multiple ways during the process of data mining or machine learning [16] [61]. On the other hand, the generated results, usually classifiers or algorithms, can serve as context extraction methods.

Since CSM can help model dynamic context and reason implicit relationships through matrix computing mechanisms, how a CSM approach can benefit the research on social media data (big data) worthies the exploration. The CSM concepts, such as domain, category, object, attribute and state should be used to model the events, categories or features of social media. If the states of social media data are implicitly related with human behaviors or events, the

applications of CSM can help to identify the contextual relations through context reasoning so as to benefit special context-aware applications which take advantage of social media data.

7.2.4 A Distributed CSM Platform Architecture

For an individual person's daily life scale decision and prediction, it might be hard to conduct big data context computing due to lack of the ability to generate big data. Nowadays, big data analysis is being practiced by the organizations or companies, which already obtained and stored big data. Online merchant like Amazon collects each individual customer's data and can use them to generate "customer habits" for company's commercial interests. As for individual's data and for individual's interests, a data-sharing platform can be built for this purpose. Internet of things generates big data and allows people and things to be connected ubiquitously and context can be extracted within these connections. So that a future research can be to build a context-sharing platform in Internet of things era [46] [60].

In our dissertation research, I implement the state-based approach using a framework which treat states as the basic element and model context for only one domain. In the future, I can extend the framework so that it can support distributed computing for different domains and support data collecting and context sharing.

Social media data, such as images from Flickr, tweets form Twitter, and YouTube videos, can be a source of context, and one advantage is that the information can be extracted in automatic ways by using the social media companies' public interfaces. Hence, a distributed CSM platform can be built to support the context sharing and computing. The servers for image/textual information collection can pass information to the CSM building servers, as described in Figure 24. Different context state machine generation servers can communicate with

each other to exchange information. Servers for context-aware machine learning and data mining can utilize the context state machines, CSM tables, and historic data to make further computing.

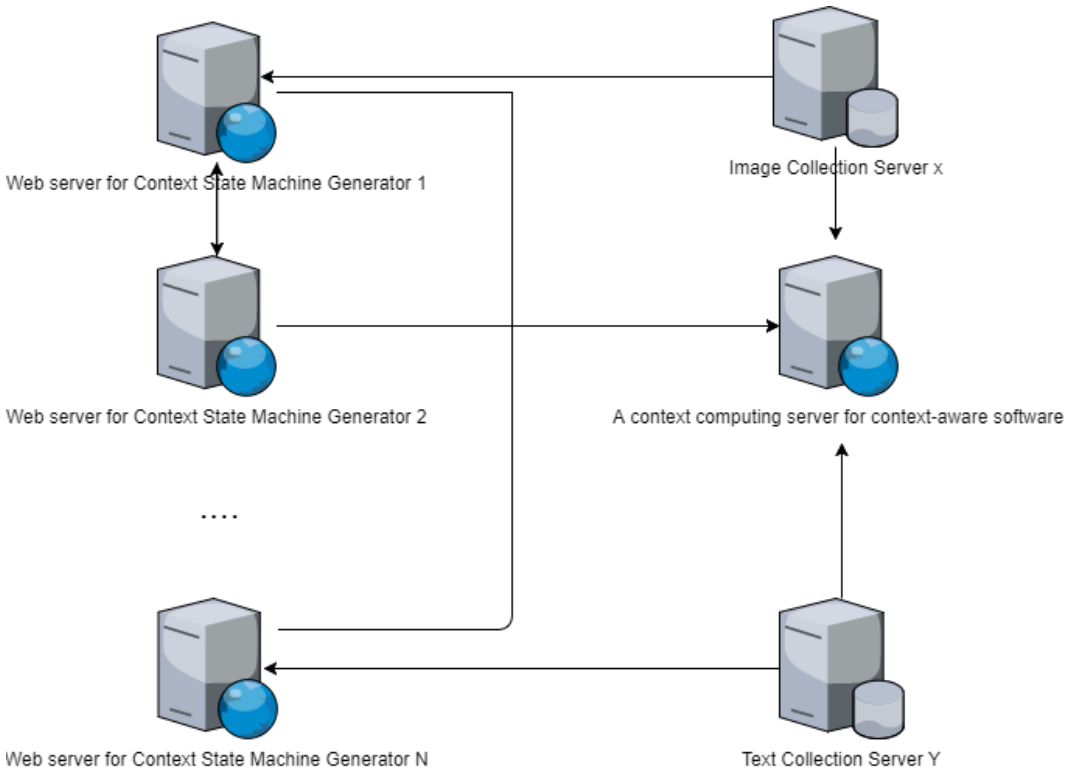

Figure 24. The Architecture of a Distributed CSM Platform

7.2.5 Privacy mechanism

Since the contextual information has been modeled to numerals in CSMs and using of meaningful words could be avoided. It is possible to explore using the CSMs as public keys and use the ontology models of a domain as private keys to assure the privacy during the context sharing or transmitting.

7.2.6 Some Other Implementation Options of CSM as Future Work

1) Since activity can be deduced from a combination of states [74], I can use activity as one dimension of a situation, instead of only using states and transitions in our experiments.

However, the using of activity cannot totally replace using of states and transitions because they are in different level of granularity.

2) The closeness of a relation between two entities could be defined using the constraints like “time spent together” or other features, as a result we can have an option to only consider some closest entities’ situations.
3) Similarity of entities can help in context sharing. If a person A who has not a CASM yet is similar to another person B who has a CASM, even if there is not enough data for A, the system can use the CASM of B temporarily.
4) Matrix similarity can be explored to measure the similarity of entities. Matrix similarity should include state similarity, and for the same states, there is data distribution similarity; Jaccard similarity [50] is a possible way to explore the matrix similarity. Another future work regarding matrices is to develop a matrix package for Java or other programming languages. The package would include the implementations of various aspects of matrix computing needed for CSM modeling and computing.
5) Priority can be used as a context to aide elevator scheduling. Besides using CSMs to supply the historical decision information for the context-aware elevator system discussed in chapter 4, the priorities of entities can be used as a contextual feature as input to the dispatch function. The priority can be calculated in two ways. Firstly, entities related to emergency responses like firefighters can be directly assigned with higher priorities. Secondly, CSMs or other context reasoning techniques can be used to determine whether an entity is in an urgent situation and needs a higher priority. For example, if a student is going to a building to take an exam and the exam has started, the student can be assigned with a higher priority to use the elevators.

7.3 Conclusion

Context-aware is an important research field with the advancement of sensor technologies with many challenging issues to be solved. This dissertation addresses the context modeling for dynamic and implicit contextual information needed to support semi-automatic context reasoning. I proposed a state-based modeling and computing approach, designed a CSM engine, and used the contextual information of context-aware elevator scenarios to evaluate the CSM engine.

Our research about the state-based approach consists of two parts: 1) a state-based modeling part. It is a state-based framework which put state as the first-class citizen. Based on the fact that the states of some context features can be prone to steady after collecting the overall data and give an analysis to them. Using the states, context can be modeled to support the state-based context computing, which leads to the second part of our research; 2) a state-based computing part. I use the word computing to differentiate this part with the basic modeling part, and our computing part specially refers to the implementation of finite state machines of contextual data to encapsulate the contextual data in the programming states, more specifically, in CSMs, as a way to model the steady states and the states transitions. I have shown how it can support semi-automatic context reasoning, which is the advantage of this modeling approach.

REFERENCES

[1] Aditi G., Hemank L., Ponnurangam K., and Anupam J. (2013). “Faking Sandy: characterizing and identifying fake images on Twitter during Hurricane Sandy.” In Proceedings of the 22nd International Conference on World Wide Web (WWW '13 Companion). ACM, New York, NY, USA, 729-736.

[2] Alexandros B., Nikos P., Babis M., Dimitris A. and Gregoris M. (2016). “A probabilistic model for context-aware proactive decision making.” 7th International Conference on Information, Intelligence, Systems & Applications (IISA), Chalkidiki, pp. 1-6.

[3] Amir P., Seng W. L., Arkady Z., Claudio B., and Bernard B. (2005). “An approach to data fusion for context-awareness.” In Proceedings of the 5th International Conference on Modeling and Using Context: LNAI 3554, pages 353–367, Paris.

[4] Anind K. D., Gregory D. A., Andrew W. (1999) “CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services.” Knowledge-Based Systems, 11 3-13.

[5] Antonio B., Annapaola M., Marco P., Heorhi R. (2017). “A Context-Aware Framework for Dynamic Composition of Process Fragments in the Internet of Services.” J. Internet Services and Applications, vol. 8, no. 1, pp. 6:1-6:23.

[6] Asad M. K., Noman A., Mohammad A., Taqdir A., Adil M. K., Seokhee J., Myunggwon H. and Sungyoung L. (2014). “Context Representation and Fusion: Advancements and Opportunities.” Sensors, 9628-9668.

[7] Asad M. K., Zeeshan P., Sungyoun L. and Young-Koo L. (2011). “Intelligent Healthcare Service Provisioning Using Ontology with Low-Level Sensory Data,” KSII Transactions on Internet and Information Systems, vol. 5, no. 11, pp. 2016-2034.

[8] Bachir C., Emmanuel B., and Noël C. (2013). “A graph-based context modeling approach.” International Conference on Smart Communications in Network Technologies (SaCoNeT), Paris, pp. 1-6.

[9] Balázs H. and Domonkos T. (2014). “Approximate modeling of continuous context in factorization algorithms.” In Proceedings of the 4th Workshop on Context-Awareness in Retrieval and Recommendation (CARR '14). ACM, New York, NY, USA, 3-9.

[10] Bettini, Claudio et al. (2010) “A survey of context modelling and reasoning techniques.” Pervasive and Mobile Computing 6: 161-180.

[11] Bill N. S., Marvin M. T. (1994) “Disseminating Active Map Information to Mobile Hosts.” IEEE Network, 8(5) 22-32.

[12] Bridget B. and Mohan K. (2010). “HyCoRE: Towards a generalized hierarchical hybrid context reasoning engine.” in Proc. 8th IEEE Int. Conf. Pervasive Comput. Commun. Workshops, Mannheim, Germany, pp. 30–36.

[13] Brown, P.J. (1996) “The Stick-e Document: a Framework for Creating Context-Aware Applications,” Electronic Publishing ’96 259-272.

[14] Business Dictionary, http://www.businessdictionary.com/definition/computer-modeling.html

[15] Chang X. and Shing-Chi C. (2005) “Inconsistency detection and resolution for contextaware middleware support.” In Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE-13). ACM, New York, NY, USA, 336-345.

[16] Chang X., Dacheng T., and Chao X. (2013) “A survey on multi-view learning.” Neural Comput. Appl., vol. 23, no. 7–8, pp. 2031–2038.

[17] Chang X., Shing-Chi C., Xiaoxing M., Chun C., Jian L. (2012) “Adam: Identifying defects in context-aware adaptation.” Journal of Systems and Software, Volume 85, Issue 12, Pages 2812-2828, ISSN 0164-1212.

[18] Charith P., Arkady Z., Peter C. and Dimitrios G. (2014) “Context Aware Computing for The Internet of Things: A Survey.” in IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 414-454, First Quarter.

[19] Cheverst K., Mitchell K., Davies N., (1999) “Design of an object model for a context sensitive tourist guide.” Comput. Graph. 23:883–891.

[20] Claudio B., Oliver B., Karen H., Jadwiga I., Daniela N., Anand R., Daniele R. (2010) “A survey of context modelling and reasoning techniques.” Pervasive and Mobile Computing, Volume 6, Issue 2, Pages 161-180, ISSN 1574-1192.

[21] Dale A. L., (2009) “A blueprint for higher-level fusion systems.” Information Fusion, Volume 10, Issue 1, Pages 6-24, ISSN 1566-2535.

[22] David H. (1987) “Statecharts: A visual formalism for complex systems.” Sci. Comput. Program. 8, 3, 231-274.

[23] Dejene E., Marian S., Lionel B., (2007) “An ontology-based approach to context modeling and reasoning in pervasive computing.” Pervasive Computing and Communications Workshops, PerCom Workshops '07, Fifth Annual IEEE International Conference on, pp. 14–19.

[24] Di Z., Hang Y., Jun W. (2011) “Research of the Middleware Based Quality Management for Context-Aware Pervasive Applications.” 2011 International Conference on Computer and Management (CAMAN), vol., no., pp.1, 4, 19-21.

[25] Dik Lun L., Qiuxia C. (2007) “A model-based WiFi localization method.” In Proceedings of the 2nd international conference on Scalable information systems (InfoScale '07). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, Article 40, 7 pages.

[26] Edwin J.Y. W., Alvin T.S. C. (2013) “CAMPUS: A middleware for automated context-aware adaptation decision making at run time.” Pervasive and Mobile Computing, Volume 9, Issue 1, Pages 35-56, ISSN 1574-1192.

[27] Eleanor O., David L., Kris M., and Simon D. (2006) “Rapid user-centred evaluation for context-aware systems.” In Proceedings of the 13th international conference on Interactive systems: Design, specification, and verification (DSVIS'06), Gavin Doherty and Ann Blandford (Eds.). Springer-Verlag, Berlin, Heidelberg, 220-233.

[28] Fatima M., Jalel Ben O., Abdellatif K., Essaid S. and Mohammed El K. (2016) “A Markov Chain Model for Integrating Context in Recommender Systems.” 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, pp. 1-6.

[29] George H. M. (1955) “A Method to Synthesizing Sequential Circuits.” Bell System Technical J, 1045-1079.

[30] Gregory D. A., Anind K. D., Peter J. B., Nigel D., Mark S., Pete S. (1999) “Towards a Better Understanding of Context and Context-Awareness.” HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. Publisher: Springer-Verlag.

[31] Hao Y. and Ted S. (2000) “Context-aware office assistant.” In Proceedings of the 5th international conference on Intelligent user interfaces (IUI '00). ACM, New York, NY, USA, 276-279.

[32] Heesoo M., Ju Yong C. and Kyoung Mu L. (2012) “Learning object relationships via graph-based context model.” 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, pp. 2727-2734.

[33] Heng L., Chan W.K., Tse T. H. (2006) “Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation,” SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering.

[34] Heng L., Chan W.K., Tse T.H. (2008) “Testing pervasive software in the presence of context inconsistency resolution services.” ICSE '08. ACM/IEEE 30th International Conference on Software Engineering, vol., no., pp.61,70, 10-18

[35] Hong, D., Schmidtke H.R., Woo W. (2007) “Linking context modelling and contextual reasoning.” In Proceedings of the 4th International Workshop on Modelling and Reasoning in Context, Roskilde, Denmark, 20–21.

[36] https://blogs.synopsys.com/configurablethoughts/2012/05/sensing-your-world/

[37] Interoperability https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=182763

[38] Jagdev B., and Philip M. (2014) “Towards Object-Oriented Context Modeling: Object-Oriented Relational Database Data Storage.” 2014 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, pp. 542-547.

[39] Jalal M., Jeffrey N., and Clemens D. (2014) “Home Location Identification of Twitter Users.” ACM Trans. Intell. Syst. Technol. 5, 3, Article 47, 21 pages.

[40] Jongyi H., Eui-Ho S., Junyoung K., SuYeon K. (2009) “Context-aware system for proactive personalized service based on context history.” Expert Systems with Applications, Volume 36, Issue 4, Pages 7448-7457, ISSN 0957-4174

[41] Jung-Hwan K. and Byung-Ro M. (2001) “Adaptive elevator group control with cameras.” in IEEE Transactions on Industrial Electronics, vol. 48, no. 2, pp. 377-382.

[42] Kristian Ellebæk K. (2007) “A survey of context-aware middleware.” In Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering (SE'07), W. Hasselbring (Ed.). ACTA Press, Anaheim, CA, USA, 148-155.

[43] Kwon O., Bahn H., Koh K. (2006) “A context-aware elevator scheduling system for smart apartment buildings.” In: Proceedings of the 1st international conference on advances in hybrid information technology: Jeju Island, Korea; 9–11. p. 362–72.

[44] Li X., Eckert M., Martinez J.-F., Rubio G. (2015) “Context Aware Middleware Architectures: Survey and Challenges.” Passaro VMN, ed. Sensors (Basel, Switzerland).

[45] Licia C., Wolfgang E., Cecilia M. (2003) “CARISMA: context-aware reflective middleware system for mobile applications.” IEEE Transactions on Software Engineering, vol.29, no.10, pp.929,945

[46] Lu T. and Neng W. (2010) “Future internet: The internet of things.” in 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), vol. 5, pp. V5–376–V5–380.

[47] Luis R.-B., Juan M.-G., Jose J. C.-S., Carlos S., Leonardo G. J. (2008) “Action recognition in video sequences using a mealy machine,” World Congress of Science, Engineering and Technology. 31:42–48.

[48] Matko K., Hrvoje K., Iva B., Mario K., Gordan J. (2011) “Android/OSGi-based Machine-to-Machine context-aware system.” Proceedings of the 11th International Conference on Telecommunications, ConTEL. 95 - 102.

[49] Matthias B., Schahram D., and Florian R. (2007) “A survey on context-aware systems.” Int. J. Ad Hoc Ubiquitous Comput. 2, 4, 263-277.

[50] McCormick W. P., Lyons N. I., and Hutcheson K. (1992) “Distributional properties of Jaccard's index of similarity.”Commun. Statist., Theory Meth.,21, No. 1, 51–68.

[51] Michele S., David S. R., Zhimin W., and Sebastian E., (2008) “Model-based fault detection in context-aware adaptive applications.” In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08/FSE-16). ACM, New York, NY, USA, 261-271.

[52] Minsu J., Jaehong K., Joo-Chan S., (2005) “Simulation framework for testing context-aware ubiquitous applications.” ICACT 2005. The 7th International Conference on Advanced Communication Technology, vol.2, no., pp.1337-1340, 0-0 0.

[53] Muhammad A., Wiratunga N., and Lothian R. (2015) “Context-aware sentiment analysis of social media.” In Advances in Social Media Analysis. Switzerland, pp. 87–104

[54] Nguyen T. V. and Choi D. (2008) "Context Reasoning Using Contextual Graph." *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, Sydney, QLD, pp. 488-493.

[55] Omer B. S., Erdogan D., Ahmet M. O. (2018) "Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey," in *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 1-27.

[56] Ovidiu V., Patrick G., and Peter F. (2009) “Internet of things strategic research roadmap.” The Cluster of European Research Projects, Tech. Rep.

[57] Paul A. and Jeff O. (2008) “Introduction to software testing.” Cambridge University Press.

[58] Petteri N., and Patrik F. (2004) “Reasoning in Context-Aware Systems.” Helsinki Inst. Inf. Technol., Espoo, Finland, pp. 1–6.

[59] Qin W., Shi Y., Suo Y. (2007) “Ontology-based context-aware middleware for smart spaces.” Tsinghua Science and Technology, vol.12, no.6, pp.707,713.

[60] Rajkumar Bu., Chee S. Y., Srikumar V. (2008) “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities.” HPCC '08. 10th IEEE International Conference on High Performance Computing and Communications, vol., no., pp.5-13, 25-27.

[61] Sakaki T., Okazaki M., and Matsuo Y. (2010) “Earthquake shakes twitter users: real-time event detection by social sensors.” In WWW.

[62] Shouling J., Jing (Selena) H., A. Selcuk U., Raheem B. and Yingshu L. (2013) “Cell-based snapshot and continuous data collection in wireless sensor networks.” ACM Trans. Sen. Netw. 9, 4, Article 47, 29 pages.

[63] Siyuan C., Shaojie T., Minsu H., Yu W. (2010) “Capacity of Data Collection in Arbitrary Wireless Sensor Networks.” in INFOCOM, 2010 Proceedings IEEE, vol., no., pp.1-5, 14-19.

[64] Songhui Y., Jyothsna K., Aibek M., Songqing Y., Randy S., (2018) “Using Twitter Data to Determine Hurricane Category: An Experiment.” ISCRAM.

[65] Songhui Y., Songqing Y. and Randy S. (2016) “A survey of testing context-aware software, challenges and resolutions.” Conference of SERP in WorldComp.

[66] Songhui Y., Songqing Y., and Randy S. (2017) “A State-based Approach to Context Modeling and Computing.” IEEE Ubiquitous Intelligence and Computing.

[67] Stefan T. and Jens T. (2009) “General method for testing context aware applications.” In Proceedings of the 6th international workshop on Managing ubiquitous communications and services (MUCS '09). ACM, New York, NY, USA, 3-8.

[68] Stefano C., Florian D., Maristella M., and Federico M. F. (2007) “Model-driven development of context-aware Web applications.” ACM Trans. Internet Technol. 7, 1, Article 2.

[69] Strang T. and Bauer C. (2007) "Context-Aware Elevator Scheduling." 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), Niagara Falls, Ont., pp. 276-281.

[70] Suman N. (2012) “ACE: exploiting correlation for energy-efficient and continuous context sensing.” In Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12). ACM, New York, NY, USA, 29-42.

[71] Sven M. and Andry R. (2003) “A survey of research on context-aware homes,” In Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003 - Volume 21 (ACSW Frontiers '03), Chris Johnson, Paul Montague, and Chris Steketee (Eds.), Vol. 21. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 159-168.

[72] Tam Van N., Wontaek L., Huy N., Deokjai C. and Chilwoo L. (2007) “Context Ontology Implementation for Smart Home.” Proc. The 2nd International Conference on Ubiquitous Information Technologies & Applications, Java Island, Indonesia.

[73] Tao G., Hung K. P., Da Q. Z., (2005) "A service-oriented middleware for building context-aware services." Journal of Network and Computer Applications, Volume 28, Issue 1, Pages 1-18, ISSN 1084-8045.

[74] Teixeira T., Jung D., Dublon G. and Savvides A. (2009) "Recognizing activities from context and arm pose using finite state machines." 2009 Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), Como, pp. 1-8.

[75] Tobias G., Volker G. (2014) "A model-based approach to test automation for context-aware mobile applications." In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14). ACM, New York, NY, USA, 420-427.

[76] Vaninha V., Konstantin H., and Michael H. (2015) "A context simulator as testing support for mobile apps." In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, New York, NY, USA, 535-541.

[77] Vanrompay, Y., Berbers Y. (2012) "A Methodological Approach to Quality of Future Context for Proactive Smart Systems." In: Andreev, S.; Balandin, S. & Koucheryavy, Y. (Eds.), Internet of Things, Smart Spaces, and Next Generation Networking, LNCS, 152-163.

[78] Wang X. H., Zhang D. Q., Gu T. and Pung H. K. (2004) "Ontology based context modeling and reasoning using OWL." IEEE Annual Conference on Pervasive Computing and Communications Workshops, Proceedings of the Second, pp. 18-22.

[79] Want R., Hopper A., Falcao V. and Gibbons J. (1992) "The active badge location system." ACM Transactions on Information Systems, Vol. 10, No. 1, pp.91–102.

[80] Wu, H., Yue, K., Liu, X., Pei, Y., & Li, B. (2015). "Context-Aware Recommendation via Graph-Based Contextual Modeling and Postfiltering." International Journal of Distributed Sensor Networks.

[81] Zhang D., Huang H., Lai C.F., Liang X., Zou Q., Guo M. (2013) "Survey on context-awareness in ubiquitous media." Multimed. Tools Appl. 67:179–211.

[82] Zhimin W., Sebastian E., David R., (2007) "Automated Generation of Context-Aware Tests." ICSE. 29th International Conference on Software Engineering, vol., no., pp.406, 415, 20-26.

APPENDIX

CSM IMPLEMENTATION

In this Appendix, the key packages of CSM implementation are shown, including the core module, matrix module, testing data generation module, and the module for CASM/CSSM assembling.

A.1 Core

A.1.1 ContextDomain

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  // this class is used to handle a Json file which contains a list of information that has not
7.  // been identified the CASMs.
8.  public class ContextDomain {
9.
10.     private String domainId;
11.     private String domainName;
12.     private String domainURI;
13.     private List<String> col_domain = new ArrayList<String>(); //use object URI to identify, a list of all entities
14.     private ContextRelationship cr = new ContextRelationship(); //for computing
15.     private List<ContextCategory> ccl = new ArrayList<ContextCategory>(); //for modeling
16.
17.
18.     public ContextDomain(){
19.     }
20.
21.     public ContextDomain(String domainId, String domainName, String domainURI){
22.         this.domainId=domainId;
23.         this.domainName=domainName;
24.         this.domainURI=domainURI;
25.     }
26.
27.     public void addContextCategory(ContextCategory cc){
28.         this.ccl.add(cc);
29.     }
30.
31.     public void addContextObjectURI(String uri){
32.         this.col_domain.add(uri);
33.         cr.addEnity(this.col_domain.size());
```

```
34.     }
35.
36.     /**
37.      * @return the col_domain
38.      */
39.     public List<String> getCol_domain() {
40.         return col_domain;
41.     }
42.
43.     /**
44.      * @param col_domain the col_domain to set
45.      */
46.     public void setCol_domain(List<String> col_domain) {
47.         this.col_domain = col_domain;
48.     }
49.
50.     /**
51.      * @return the cr
52.      */
53.     public ContextRelationship getCr() {
54.         return cr;
55.     }
56.
57.     /**
58.      * @param cr the cr to set
59.      */
60.     public void setCr(ContextRelationship cr) {
61.         this.cr = cr;
62.     }
63.
64.     /**
65.      * @return the ccl
66.      */
67.     public List<ContextCategory> getCcl() {
68.         return ccl;
69.     }
70.
71.     /**
72.      * @param ccl the ccl to set
73.      */
74.     public void setCcl(List<ContextCategory> ccl) {
75.         this.ccl = ccl;
76.     }
77.
78.     /**
79.      * @return the domainId
80.      */
81.     public String getDomainId() {
82.         return domainId;
83.     }
84.
85.     /**
86.      * @param domainId the domainId to set
87.      */
88.     public void setDomainId(String domainId) {
89.         this.domainId = domainId;
90.     }
91.
92.     /**
93.      * @return the domainName
94.      */
```

```
95.     public String getDomainName() {
96.         return domainName;
97.     }
98.
99.     /**
100.     * @param domainName the domainName to set
101.     */
102.    public void setDomainName(String domainName) {
103.        this.domainName = domainName;
104.    }
105.
106.    /**
107.     * @return the domainURI
108.     */
109.    public String getDomainURI() {
110.        return domainURI;
111.    }
112.
113.    /**
114.     * @param domainURI the domainURI to set
115.     */
116.    public void setDomainURI(String domainURI) {
117.        this.domainURI = domainURI;
118.    }
119.
120.
121. }
```

A.1.2 ContextCategory

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.Iterator;
5.  import java.util.List;
6.
7.  // this class is used to handle a Json file which contains a list of information that has not
8.  // been identified the CASMs.
9.  public class ContextCategory {
10.     private int id;
11.     private String categoryType;
12.     private String categoryName;
13.     private String categoryURI;
14.
15.     private List<ContextObject> col = new ArrayList<ContextObject>();
16.
17.
18.     public ContextCategory(){
19.     }
20.
21.     public ContextCategory(String categoryName, String categoryURI){
22.         this.categoryName=categoryName;
23.         this.categoryURI=categoryURI;
24.     }
25.
26.     public void addContextObject(ContextObject co){
27.         this.col.add(co);
```

```
28.     }
29.
30.     /**
31.      * @return the id
32.      */
33.     public int getId() {
34.         return id;
35.     }
36.
37.     /**
38.      * @param id the id to set
39.      */
40.     public void setId(int id) {
41.         this.id = id;
42.     }
43.
44.     /**
45.      * @return the col
46.      */
47.     public List<ContextObject> getCol() {
48.         return col;
49.     }
50.
51.     /**
52.      * @param col the col to set
53.      */
54.     public void setCol(List<ContextObject> col) {
55.         this.col = col;
56.     }
57.
58.     /**
59.      * @return the categoryType
60.      */
61.     public String getCategoryType() {
62.         return categoryType;
63.     }
64.
65.     /**
66.      * @param categoryType the categoryType to set
67.      */
68.     public void setCategoryType(String categoryType) {
69.         this.categoryType = categoryType;
70.     }
71.
72.     /**
73.      * @return the categoryName
74.      */
75.     public String getCategoryName() {
76.         return categoryName;
77.     }
78.
79.     /**
80.      * @param categoryName the categoryName to set
81.      */
82.     public void setCategoryName(String categoryName) {
83.         this.categoryName = categoryName;
84.     }
85.
86.     /**
87.      * @return the categoryURI
88.      */
```

```
89.     public String getCategoryURI() {
90.         return categoryURI;
91.     }
92.
93.     /**
94.      * @param categoryURI the categoryURI to set
95.      */
96.     public void setCategoryURI(String categoryURI) {
97.         this.categoryURI = categoryURI;
98.     }
99.
100.
101. }
```

A.1.3 ContextRelationship

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.HashMap;
5.  import java.util.List;
6.
7.  import context.matrix.RMatrixes;
8.  /**
9.   * @author syue2
10.  *
11.  *the third dimension, the possible point of the relationship grade can be 100.
12.  *There can be different contracts, we use 1-10 as one segment, and if the digit in ones is greater than 7 or 8, it
13.  *can be counted as significant enough to affect some decision related to that corresponding digit in tens.
14.  *
15.  *However, we only use one rule here, if the number is more that 80, then we count it as significant enough.
16.  */
17. public class ContextRelationship {
18.     private List<RelationshipPair> relationshipTypeList = new ArrayList<RelationshipPair>();
19.     private RMatrixes rMatrixes = new RMatrixes();
20.
21.     public ContextRelationship(){
22.
23.     }
24.
25.     /**
26.      * @return the rMatrixes
27.      */
28.     public RMatrixes getrMatrixes() {
29.         return rMatrixes;
30.     }
31.     /**
32.      * @param rMatrixes the rMatrixes to set
33.      */
34.     public void setrMatrixes(RMatrixes rMatrixes) {
35.         this.rMatrixes = rMatrixes;
36.     }
37.
38.     public void addEnity(int size){
39.
40.     }
41.
42.     /**
43.      * @return the relationshipTypeList
```

```
     */
    public List<RelationshipPair> getRelationshipTypeList() {
        return relationshipTypeList;
    }

    /**
     * @param relationshipTypeList the relationshipTypeList to set
     */
    public void setRelationshipTypeList(List<RelationshipPair> relationshipTypeList) {
        this.relationshipTypeList = relationshipTypeList;
    }



}
```

A.1.4 ContextObject

```
package context.core;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import context.utility.ListHelper;

public class ContextObject {
    private int id;
    private String objectName;
    private String objectURI;
    private ContextSituationState currentSituationState;
    private int currentSituationStateID;
    private List<ContextAttributeStateMachine> casml = new ArrayList<ContextAttributeStateMachine>();
    private ContextSituationStateMachine cssm = new ContextSituationStateMachine(); //a situation is a combination of attributes of multiple objects
    //as for now, we only consider the related persons' location information in situation
    //TODO: to extend the context attribute, we can add context attribute type URI, so there can be different type
    //now we only consider one type

    //context attribute list
    private List<ContextAttribute> cal = new ArrayList<ContextAttribute>();

    //context situation state list
    private List<ContextSituationState> cssl = new ArrayList<ContextSituationState>();

    public ContextObject(){

    }

    public ContextObject(String objectName, String objectURI){
        this.objectName = objectName;
        this.objectURI = objectURI;
    }

    /**
```

```
37.      * this method is important
38.      * It get the current object state, which will be used by combining the related objects' states,
39.      * to form a situation for this object
40.      *
41.      * The object state include all the current states of the attributes
42.      */
43.     public void getCurrentObjectStateForSituation(){
44.
45.     }
46.
47.     /**
48.      * @return the objectURI
49.      */
50.     public String getObjectURI() {
51.         return objectURI;
52.     }
53.     /**
54.      * @param objectURI the objectURI to set
55.      */
56.     public void setObjectURI(String objectURI) {
57.         this.objectURI = objectURI;
58.     }
59.     /**
60.      * @return the id
61.      */
62.     public int getId() {
63.         return id;
64.     }
65.
66.     /**
67.      * @param id the id to set
68.      */
69.     public void setId(int id) {
70.         this.id = id;
71.     }
72.
73.     /**
74.      * @return the objectName
75.      */
76.     public String getObjectName() {
77.         return objectName;
78.     }
79.     /**
80.      * @param objectName the objectName to set
81.      */
82.     public void setObjectName(String objectName) {
83.         this.objectName = objectName;
84.     }
85.     /**
86.      * @return the casml
87.      */
88.     public List<ContextAttributeStateMachine> getCasml() {
89.         return casml;
90.     }
91.     /**
92.      * @param casml the casml to set
93.      */
94.     public void setCasml(List<ContextAttributeStateMachine> casml) {
95.         this.casml = casml;
96.     }
97.
```

```
98.
99.     /**
100.     * @return the cssm
101.     */
102.    public ContextSituationStateMachine getCssm() {
103.        return cssm;
104.    }
105.
106.    /**
107.     * @param cssm the cssm to set
108.     */
109.    public void setCssm(ContextSituationStateMachine cssm) {
110.        this.cssm = cssm;
111.    }
112.
113.    /**
114.     * @return the cal
115.     */
116.    public List<ContextAttribute> getCal() {
117.        return cal;
118.    }
119.    /**
120.     * @param cal the cal to set
121.     */
122.    public void setCal(List<ContextAttribute> cal) {
123.        this.cal = cal;
124.    }
125.
126.
127.    /**
128.     * @return the cssl
129.     */
130.    public List<ContextSituationState> getCssl() {
131.        return cssl;
132.    }
133.
134.    /**
135.     * @param cssl the cssl to set
136.     */
137.    public void setCssl(List<ContextSituationState> cssl) {
138.        this.cssl = cssl;
139.    }
140.
141.    /**
142.     * @return the currentSituationState
143.     */
144.    public ContextSituationState getCurrentSituationState() {
145.        return currentSituationState;
146.    }
147.
148.    /**
149.     * @param currentSituationState the currentSituationState to set
150.     */
151.    public void setCurrentSituationState(ContextSituationState currentSituationState) {
152.        this.currentSituationState = currentSituationState;
153.    }
154.
155.    /**
156.     * @return the currentSituationStateID
157.     */
158.    public int getCurrentSituationStateID() {
```

```
159.        return currentSituationStateID;
160.    }
161.
162.    /**
163.     * @param currentSituationStateID the currentSituationStateID to set
164.     */
165.    public void setCurrentSituationStateID(int currentSituationStateID) {
166.        this.currentSituationStateID = currentSituationStateID;
167.    }
168.
169.    public void addAttributeStateMachine(ContextAttributeStateMachine casm){
170.        casml.add(casm);
171.    }
172.
173.    public void addContextAttribute(ContextAttribute ca){
174.        cal.add(ca);
175.    }
176.
177.    public void addContextSituationStateToList(ContextSituationState css){
178.        cssl.add(css);
179.    }
180.
181.    /**
182.     * if return -1, then not exist, if the return value if more than -1, then it is the index of the css
183.     * @param css
184.     * @return
185.     */
186.    public int existAndIndexInCSSL(ContextSituationState css){
187.        int exist = -1;
188.        Iterator it = this.cssl.iterator();
189.        int tag =0;
190.        label1:
191.            while(it.hasNext()){
192.                ContextSituationState csstmp = (ContextSituationState)it.next();
193.                int size1= css.getContextAttributeSnapshotList().size();
194.                int size2= csstmp.getContextAttributeSnapshotList().size();
195.                if(size1 == size2){
196.                    Iterator it2 = css.getContextAttributeSnapshotList().iterator();
197.                    int tag2=0;
198.                    while(it2.hasNext()){
199.                        ContextAttributeSnapshot casnap = (ContextAttributeSnapshot)it2.next();
200.                        if(!ListHelper.inTheSnapshotList(casnap, csstmp.getContextAttributeSnapshotList())){
201.                            break;
202.                        }else{
203.                            if(tag2==size1-1){
204.                                exist=tag;
205.                                break label1;
206.                            }
207.                        }
208.                        tag2++;
209.                    }
210.                }
211.                tag++;
212.            }
213.        return exist;
214.    }
215.
216.    public void prettyPrint(){
217.        System.out.println("The state machine is of : "+this.objectName+".");
218.        int size=0;
219.        if(casml!=null){
```

```
            size=casml.size();
        }
        System.out.println("It has "+size+" ContextAttributeStateMachines.");
        if(size>0){
            System.out.println("They are:");
            Iterator<ContextAttributeStateMachine> it=casml.iterator();
            while(it.hasNext()){
                ContextAttributeStateMachine temp = (ContextAttributeStateMachine)it.next();
                temp.prettyPrint();
            }
        }

    }

}
```

A.1.5 ContextAttributeState

```
package context.core;

import java.util.Map;

public class ContextAttributeState {
    //?what is this attribute
    //private Map<ContextAttributeState, Integer> countNumberMap;
    private String stateURI;
    private String stateName;
    private boolean steadyStatus; // true or false

    public ContextAttributeState(){

    }

    public ContextAttributeState(String stateName, String stateURI){
        this.stateURI=stateURI;
        this.stateName=stateName;
    }

    public float getPossibility(ContextAttributeState current, ContextAttributeState target){
        return 1.00f;
    }

    /**
     * @return the stateURI
     */
    public String getStateURI() {
        return stateURI;
    }

    /**
     * @param stateURI the stateURI to set
     */
    public void setStateURI(String stateURI) {
        this.stateURI = stateURI;
    }

    /**
```

```
40.      * @return the stateName
41.      */
42.     public String getStateName() {
43.         return stateName;
44.     }
45.
46.     /**
47.      * @param stateName the stateName to set
48.      */
49.     public void setStateName(String stateName) {
50.         this.stateName = stateName;
51.     }
52.
53.     /**
54.      * @return the steadyStatus
55.      */
56.     public boolean isSteadyStatus() {
57.         return steadyStatus;
58.     }
59.
60.     /**
61.      * @param steadyStatus the steadyStatus to set
62.      */
63.     public void setSteadyStatus(boolean steadyStatus) {
64.         this.steadyStatus = steadyStatus;
65.     }
66.
67. }
```

A.1.6 ContextSituationState

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.List;
5.
6.  import context.convertion.Object;
7.
8.  //?Will this class extends the object in conversion
9.  public class ContextSituationState {
10.     private int id; // since the situation vary a lot. So we don't use URI but only use id to distinguish them.
11.     private boolean steadyStatus; // true or false
12.     private List<ContextAttributeSnapshot> contextAttributeSnapshotList = new ArrayList<ContextAttributeSnapshot>
    ();
13.
14.     public ContextSituationState(){
15.     }
16.
17.     public ContextSituationState(boolean steadyStatus){
18.         this.setSteadyStatus(steadyStatus);
19.     }
20.
21.     /**
22.      * @return the steadyStatus
23.      */
24.     public boolean isSteadyStatus() {
25.         return steadyStatus;
```

```
26.     }
27.
28.     /**
29.      * @param steadyStatus the steadyStatus to set
30.      */
31.     public void setSteadyStatus(boolean steadyStatus) {
32.         this.steadyStatus = steadyStatus;
33.     }
34.
35.     /**
36.      * @return the contextAttributeSnapshotList
37.      */
38.     public List<ContextAttributeSnapshot> getContextAttributeSnapshotList() {
39.         return contextAttributeSnapshotList;
40.     }
41.
42.     /**
43.      * @param contextAttributeSnapshotList the contextAttributeSnapshotList to set
44.      */
45.     public void setContextAttributeSnapshotList(
46.             List<ContextAttributeSnapshot> contextAttributeSnapshotList) {
47.         this.contextAttributeSnapshotList = contextAttributeSnapshotList;
48.     }
49.
50.     public void addContextAttributeSnapshotToList(ContextAttributeSnapshot casnap){
51.         this.contextAttributeSnapshotList.add(casnap);
52.     }
53.
54.     /**
55.      * @return the id
56.      */
57.     public int getId() {
58.         return id;
59.     }
60.
61.     /**
62.      * @param id the id to set
63.      */
64.     public void setId(int id) {
65.         this.id = id;
66.     }
67. }
```

A.1.7 ContextAttributeStateMachine

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.HashMap;
5.  import java.util.List;
6.  import java.util.Map;
7.  import java.util.Map.Entry;
8.  import java.util.Objects;
9.
10. import context.matrix.Matrixes;
11.
12. public class ContextAttributeStateMachine{
```

```
13.     private Matrixes matrixes = new Matrixes(); //use this structure to store the core data; matrix for state transitions, an
    d matrix for prediction
14.     //private ContextAttribute ca = new ContextAttribute(); //contains context attribute name and a list of states
15.     private HashMap<String, Integer> hmstates = new HashMap<String, Integer>(); //a key value map  like (home, 1);
    ##the string here can be extended(replaced) using the state class
16.     private HashMap<String, Integer> hmdecisions = new HashMap<String, Integer>(); //a key value map  like (take, 1
    )
17.     private int currentState = -1; //initial value is -1
18.     private String lastTransition; //a string string map  like (coffee shop, home, "01")
19.
20.     public ContextAttributeStateMachine(){
21.
22.     }
23.
24.     //match input to a specific transition
25.     public Transition matchTransition(Object input, List inputList){
26.         return new Transition(new ContextCategory());
27.     }
28.
29.     public void prettyPrint(){
30.
31.     }
32.
33.     /**
34.      * @return the transitionList
35.      */
36.     public List<Transition> getTransitionList() {
37.         List<Transition> l = new ArrayList<Transition>();
38.         return l;
39.     }
40.
41.     public static <T, E> T getKeyByValue(Map<T, E> map, E value) {
42.         for (Entry<T, E> entry : map.entrySet()) {
43.             if (Objects.equals(value, entry.getValue())) {
44.                 return entry.getKey();
45.             }
46.         }
47.         return null;
48.     }
49.
50.     public String extractCurrentStateName(){
51.         return getKeyByValue(hmstates, currentState);
52.     }
53.
54.     /**
55.      * @return the matrixes
56.      */
57.     public Matrixes getMatrixes() {
58.         return matrixes;
59.     }
60.
61.     /**
62.      * @param matrixes the matrixes to set
63.      */
64.     public void setMatrixes(Matrixes matrixes) {
65.         this.matrixes = matrixes;
66.     }
67.
68.     /**
69.      * @return the hmstates
70.      */
```

```
71.     public HashMap<String, Integer> getHmstates() {
72.         return hmstates;
73.     }
74.
75.     /**
76.      * @param hmstates the hmstates to set
77.      */
78.     public void setHmstates(HashMap<String, Integer> hmstates) {
79.         this.hmstates = hmstates;
80.     }
81.
82.     public void addtoHashMapStates(int place, String stateURI){
83.         this.hmstates.put(stateURI, place);
84.     }
85.
86.     /**
87.      * @return the currentState
88.      */
89.     public int getCurrentState() {
90.         return currentState;
91.     }
92.
93.     /**
94.      * @param currentState the currentState to set
95.      */
96.     public void setCurrentState(int currentState) {
97.         this.currentState = currentState;
98.     }
99.
100.    /**
101.     * @return the hmdecisions
102.     */
103.    public HashMap<String, Integer> getHmdecisions() {
104.        return hmdecisions;
105.    }
106.
107.    /**
108.     * @param hmdecisions the hmdecisions to set
109.     */
110.    public void setHmdecisions(HashMap<String, Integer> hmdecisions) {
111.        this.hmdecisions = hmdecisions;
112.    }
113.
114.    /**
115.     * @return the lastTransition
116.     */
117.    public String getLastTransition() {
118.        return lastTransition;
119.    }
120.
121.    /**
122.     * @param lastTransition the lastTransition to set
123.     */
124.    public void setLastTransition(String lastTransition) {
125.        this.lastTransition = lastTransition;
126.    }
127. }
```

A.1.8 ContextSituationStateMachine

```
1.  package context.core;
2.
3.  import java.util.ArrayList;
4.  import java.util.HashMap;
5.  import java.util.List;
6.
7.  import context.matrix.Matrixes;
8.  import context.matrix.SMatrixes;
9.
10. public class ContextSituationStateMachine{
11.     private SMatrixes smatrixes = new SMatrixes(); //use this structure to store the core data; matrix for state transitions
    , and matrix for prediction
12.     //private ContextAttribute ca = new ContextAttribute(); //contains context attribute name and a list of states
13.     private HashMap<String, Integer> hmstates = new HashMap<String, Integer>(); //a key value map  like (home, 1);
    ##the string here can be extended(replaced) using the state class
14.     private HashMap<String, Integer> hmdecisions = new HashMap<String, Integer>(); //a key value map  like (take, 1
    )
15.     private int currentState = -1; //initial value is -1
16.     private String lastTransition; //a string string map  like (coffee shop, home, "01")
17.
18.     public ContextSituationStateMachine(){
19.
20.     }
21.
22.     //match input to a specific transition
23.     public Transition matchTransition(Object input, List inputList){
24.         return new Transition(new ContextCategory());
25.     }
26.
27.     public void prettyPrint(){
28.
29.     }
30.
31.     /**
32.      * @return the smatrixes
33.      */
34.     public SMatrixes getSmatrixes() {
35.         return smatrixes;
36.     }
37.
38.     /**
39.      * @param smatrixes the smatrixes to set
40.      */
41.     public void setSmatrixes(SMatrixes smatrixes) {
42.         this.smatrixes = smatrixes;
43.     }
44.
45.     /**
46.      * @return the transitionList
47.      */
48.     public List<Transition> getTransitionList() {
49.         List<Transition> l = new ArrayList<Transition>();
50.         return l;
51.     }
52.
53.     /**
54.      * @return the hmstates
```

```
55.      */
56.     public HashMap<String, Integer> getHmstates() {
57.         return hmstates;
58.     }
59.
60.     /**
61.      * @param hmstates the hmstates to set
62.      */
63.     public void setHmstates(HashMap<String, Integer> hmstates) {
64.         this.hmstates = hmstates;
65.     }
66.
67.     public void addtoHashMapStates(int place, String stateName){
68.         this.hmstates.put(stateName, place);
69.     }
70.
71.     /**
72.      * @return the currentState
73.      */
74.     public int getCurrentState() {
75.         return currentState;
76.     }
77.
78.     /**
79.      * @param currentState the currentState to set
80.      */
81.     public void setCurrentState(int currentState) {
82.         this.currentState = currentState;
83.     }
84.
85.     /**
86.      * @return the hmdecisions
87.      */
88.     public HashMap<String, Integer> getHmdecisions() {
89.         return hmdecisions;
90.     }
91.
92.     /**
93.      * @param hmdecisions the hmdecisions to set
94.      */
95.     public void setHmdecisions(HashMap<String, Integer> hmdecisions) {
96.         this.hmdecisions = hmdecisions;
97.     }
98.
99.     /**
100.     * @return the lastTransition
101.     */
102.    public String getLastTransition() {
103.        return lastTransition;
104.    }
105.
106.    /**
107.     * @param lastTransition the lastTransition to set
108.     */
109.    public void setLastTransition(String lastTransition) {
110.        this.lastTransition = lastTransition;
111.    }
112. }
```

A.2 Matrix

A.2.1 Matrixes for CASM

```
1.  package context.matrix;
2.
3.  import java.util.Vector;
4.
5.  public class Matrixes {
6.      //Using vector for 2D array. It is threadsafe.
7.      //The important idea about matrix is that we can use deep matrix to represent the multiple level conditions such as sta
    te(transition)/decision
8.      private Vector<Vector<Integer>>  casmMatrix= new Vector<Vector<Integer>>();
9.      private Vector<Vector<Double>>  predicationMatrix= new Vector<Vector<Double>>();
10.     private Vector<Vector<String>>  functionMatrix= new Vector<Vector<String>>(); //for decisions
11.     private Vector<Vector<Vector<Integer>>>  decisionMatrix= new Vector<Vector<Vector<Integer>>>(); //decision
    can be injected from the outside and dynamically build the decision matrix
12.
13.     //the spot in the new column which is related to the preState will be set to 1
14.     public Vector addState(int preState){
15.         //handle initial
16.         if(casmMatrix.size()==0||preState==-1){
17.             Vector<Integer> r=new Vector<>();
18.             r.add(0);
19.             casmMatrix.add(r);
20.
21.             return casmMatrix;
22.         }
23.
24.         //add a row of matrix.size
25.         Vector<Integer> r=new Vector<>();
26.         for(int i=0;i<casmMatrix.size();i++){
27.             r.add(0);
28.         }
29.         casmMatrix.add(r);
30.
31.         //add column to all rows
32.         for(int j=0;j<casmMatrix.size();j++){
33.             r=casmMatrix.get(j);
34.             if(j==preState){
35.                 r.add(1);
36.             } else{
37.                 r.add(0);
38.             }
39.         }
40.         return casmMatrix;
41.     }
42.
43.     public Vector addStateForDeicision(int preState, int decision){ //if decision is 1, then it is take, if decision is 0, then
    it is pass, this can be extensible later by refactoring
44.         //handle initial
45.         if(decisionMatrix.size()==0||preState==-1){
46.             //handle decisionMatrix 2019-01-15
47.             Vector<Vector<Integer>> rd=new Vector<Vector<Integer>>();
48.             Vector<Integer> rd2=new Vector<>();
49.             if(decision == 1){
```

```
50.                 rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
51.                 rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
52.             }else{
53.                 rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
54.                 rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
55.             }
56.             rd.add(rd2);
57.             decisionMatrix.add(rd);
58.
59.             return decisionMatrix;
60.         }
61.
62.
63.         //handle decisionMatrix 2019-01-15
64.         //add a row of matrix.size
65.         Vector<Vector<Integer>> rd=new Vector<Vector<Integer>>();
66.         for(int i=0;i<decisionMatrix.size();i++){
67.             Vector<Integer> rd2=new Vector<>();
68.             rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
69.             rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
70.             rd.add(rd2);
71.         }
72.         decisionMatrix.add(rd);
73.
74.         //add column to the all rows
75.         //handle decisionMatrix 2019-01-15
76.         for(int j=0;j<decisionMatrix.size();j++){
77.             if(j==preState){
78.                 Vector<Integer> rd2=new Vector<>();
79.                 if(decision == 1){
80.                     rd2.add(1); //the first one is for "take" in the case of elevator "take/pass"
81.                     rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
82.                 }else{
83.                     rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
84.                     rd2.add(1); //the second number is for counting the pass in the case of elevator "take/pass"
85.                 }
86.                 decisionMatrix.get(j).add(rd2);
87.             } else{
88.                 Vector<Integer> rd2=new Vector<>();
89.                 rd2.add(0);
90.                 rd2.add(0);
91.                 decisionMatrix.get(j).add(rd2);
92.             }
93.         }
94.
95.         return decisionMatrix;
96.     }
97.
98.     //todo
99.     public Vector updateHmStates(int row, int column){
100.        casmMatrix.get(row).set(column, casmMatrix.get(row).get(column).intValue()+1);
101.        return casmMatrix;
102.    }
103.
104.    //handle update decisionMatrix 2019-01-15
105.    public Vector updateDecisionMatrixStates(int row, int column, int decision){
106.        if(decision == 1){
107.            decisionMatrix.get(row).get(column).set(0, decisionMatrix.get(row).get(column).get(0).intValue()+1);
108.        }else{
109.            decisionMatrix.get(row).get(column).set(1, decisionMatrix.get(row).get(column).get(1).intValue()+1);
110.        }
```

```
111.        return decisionMatrix;
112.    }
113.
114.
115.    public Vector<Vector<Integer>> getCasmMatrix() {
116.        return casmMatrix;
117.    }
118.
119.    public void setCasmMatrix(Vector<Vector<Integer>> casmMatrix) {
120.        this.casmMatrix = casmMatrix;
121.    }
122.
123.    public Vector<Vector<Double>> getPredicationMatrix() {
124.        return predicationMatrix;
125.    }
126.
127.    public void setPredicationMatrix(Vector<Vector<Double>> predicationMatrix) {
128.        this.predicationMatrix = predicationMatrix;
129.    }
130.
131.    /**
132.     * @return the functionMatrix
133.     */
134.    public Vector<Vector<String>> getFunctionMatrix() {
135.        return functionMatrix;
136.    }
137.
138.    /**
139.     * @param functionMatrix the functionMatrix to set
140.     */
141.    public void setFunctionMatrix(Vector<Vector<String>> functionMatrix) {
142.        this.functionMatrix = functionMatrix;
143.    }
144.
145.    /**
146.     * @return the decisionMatrix
147.     */
148.    public Vector<Vector<Vector<Integer>>> getDecisionMatrix() {
149.        return decisionMatrix;
150.    }
151.
152.    /**
153.     * @param decisionMatrix the decisionMatrix to set
154.     */
155.    public void setDecisionMatrix(Vector<Vector<Vector<Integer>>> decisionMatrix) {
156.        this.decisionMatrix = decisionMatrix;
157.    }
158. }
```

A.2.2 Matrixes for CSSM

```
1.  package context.matrix;
2.
3.  import java.util.Vector;
4.
5.  public class SMatrixes {
6.      //Using vector for 2D array. It is threadsafe.
```

```
7.      //The important idea about matrix is that we can use deep matrix to represent the multiple level conditions such as sta
    te(transition)/decision
8.      private Vector<Vector<Integer>>  cssmMatrix= new Vector<Vector<Integer>>();
9.      private Vector<Vector<Double>>  predicationMatrix= new Vector<Vector<Double>>();
10.     private Vector<Vector<String>>  functionMatrix= new Vector<Vector<String>>(); //for decisions
11.     private Vector<Vector<Vector<Integer>>>  decisionMatrix= new Vector<Vector<Vector<Integer>>>(); //decision
    can be injected from the outside and dynamically build the decision matrix
12.
13.     //the spot in the new column which is related to the preState will be set to 1
14.     public Vector addState(int preState){
15.         //handle initial
16.         if(cssmMatrix.size()==0||preState==-1){
17.             Vector<Integer> r=new Vector<>();
18.             r.add(0);
19.             cssmMatrix.add(r);
20.
21.             return cssmMatrix;
22.         }
23.
24.         //add a row of matrix.size
25.         Vector<Integer> r=new Vector<>();
26.         for(int i=0;i<cssmMatrix.size();i++){
27.             r.add(0);
28.         }
29.         cssmMatrix.add(r);
30.
31.         //add column to all rows
32.         for(int j=0;j<cssmMatrix.size();j++){
33.             r=cssmMatrix.get(j);
34.             if(j==preState){
35.                 cssmMatrix.get(j).add(1);
36.             } else{
37.                 cssmMatrix.get(j).add(0);
38.             }
39.         }
40.
41.         return cssmMatrix;
42.     }
43.
44.     public Vector addStateForDeicision(int preState, int decision){ //if decision is 1, then it is take, if decision is 0, then
    it is pass, this can be extensible later by refactoring
45.         //handle initial
46.         if(decisionMatrix.size()==0||preState==-1){
47.             //handle decisionMatrix 2019-01-15
48.             Vector<Vector<Integer>> rd=new Vector<Vector<Integer>>();
49.             Vector<Integer> rd2=new Vector<>();
50.             if(decision == 1){
51.                 rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
52.                 rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
53.             }else{
54.                 rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
55.                 rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
56.             }
57.             rd.add(rd2);
58.             decisionMatrix.add(rd);
59.
60.             return decisionMatrix;
61.         }
62.
63.
64.         //handle decisionMatrix 2019-01-15
```

```
65.         //add a row of matrix.size
66.         Vector<Vector<Integer>> rd=new Vector<Vector<Integer>>();
67.         for(int i=0;i<decisionMatrix.size();i++){
68.             Vector<Integer> rd2=new Vector<>();
69.             rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
70.             rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
71.             rd.add(rd2);
72.         }
73.         decisionMatrix.add(rd);
74.
75.         //add column to the all rows
76.         //handle decisionMatrix 2019-01-15
77.         for(int j=0;j<decisionMatrix.size();j++){
78.             if(j==preState){
79.                 Vector<Integer> rd2=new Vector<>();
80.                 if(decision == 1){
81.                     rd2.add(1); //the first one is for "take" in the case of elevator "take/pass"
82.                     rd2.add(0); //the second number is for counting the pass in the case of elevator "take/pass"
83.                 }else{
84.                     rd2.add(0); //the first one is for "take" in the case of elevator "take/pass"
85.                     rd2.add(1); //the second number is for counting the pass in the case of elevator "take/pass"
86.                 }
87.                 decisionMatrix.get(j).add(rd2);
88.             } else{
89.                 Vector<Integer> rd2=new Vector<>();
90.                 rd2.add(0);
91.                 rd2.add(0);
92.                 decisionMatrix.get(j).add(rd2);
93.             }
94.         }
95.
96.         return decisionMatrix;
97.     }
98.
99.     //todo
100.    public Vector updateHmStates(int row, int column){
101.        cssmMatrix.get(row).set(column, cssmMatrix.get(row).get(column).intValue()+1);
102.        return cssmMatrix;
103.    }
104.
105.    //handle update decisionMatrix 2019-01-15
106.    public Vector updateDecisionMatrixStates(int row, int column, int decision){
107.        if(decision == 1){
108.            decisionMatrix.get(row).get(column).set(0, decisionMatrix.get(row).get(column).get(0).intValue()+1);
109.        }else{
110.            decisionMatrix.get(row).get(column).set(1, decisionMatrix.get(row).get(column).get(1).intValue()+1);
111.        }
112.        return decisionMatrix;
113.    }
114.
115.
116.    public Vector<Vector<Integer>> getCssmMatrix() {
117.        return cssmMatrix;
118.    }
119.
120.    public void setCssmMatrix(Vector<Vector<Integer>> cssmMatrix) {
121.        this.cssmMatrix = cssmMatrix;
122.    }
123.
124.    public Vector<Vector<Double>> getPredicationMatrix() {
125.        return predicationMatrix;
```

```
126.    }
127.
128.    public void setPredicationMatrix(Vector<Vector<Double>> predicationMatrix) {
129.        this.predicationMatrix = predicationMatrix;
130.    }
131.
132.    /**
133.     * @return the functionMatrix
134.     */
135.    public Vector<Vector<String>> getFunctionMatrix() {
136.        return functionMatrix;
137.    }
138.
139.    /**
140.     * @param functionMatrix the functionMatrix to set
141.     */
142.    public void setFunctionMatrix(Vector<Vector<String>> functionMatrix) {
143.        this.functionMatrix = functionMatrix;
144.    }
145.
146.    /**
147.     * @return the decisionMatrix
148.     */
149.    public Vector<Vector<Vector<Integer>>> getDecisionMatrix() {
150.        return decisionMatrix;
151.    }
152.
153.    /**
154.     * @param decisionMatrix the decisionMatrix to set
155.     */
156.    public void setDecisionMatrix(Vector<Vector<Vector<Integer>>> decisionMatrix) {
157.        this.decisionMatrix = decisionMatrix;
158.    }
159. }
```

A.2.3 Matrixes for Relationship

```
1.  package context.matrix;
2.
3.  import java.util.Vector;
4.
5.  public class RMatrixes {
6.      //Using vector for 3D array. It is threadsafe.
7.      //The important idea about rmatrix is that we can use deep rmatrix to represent the multiple level conditions such as s
    tate(transition)/decision
8.      private Vector<Vector<Integer>>  rmatrix= new Vector<Vector<Integer>>();
9.      private Vector<Vector<Integer>>  rClosenessMatrix= new Vector<Vector<Integer>>();
10.     //the spot in the new column which is related to the preState will be set to 1
11.     public Vector addState(int preState){
12.         //handle initial   if prestate is 0, then it means for rmatrix
13.         Vector<Integer> r1=new Vector<>();
14.         Vector<Integer> r2=new Vector<>();
15.         if(rmatrix.size()==0||preState==-1){
16.             r1.add(-1);
```

```
17.             r2.add(-1);
18.             rmatrix.add(r1);
19.             rClosenessMatrix.add(r2);
20.             return rmatrix;
21.         }
22.
23.         //add a row of rmatrix.size
24.         Vector<Integer> r3=new Vector<>();
25.         Vector<Integer> r4=new Vector<>();
26.         for(int i=0;i<rmatrix.size();i++){
27.             r3.add(-1);
28.             r4.add(-1);
29.         }
30.
31.         rmatrix.add(r3);
32.         rClosenessMatrix.addElement(r4);
33.
34.         //add column to all rows
35.         for(int j=0;j<rmatrix.size();j++){
36.             rmatrix.get(j).add(-1);
37.             rClosenessMatrix.get(j).add(-1);
38.         }
39.
40.         return rmatrix;
41.     }
42.
43.     //todo
44.     public Vector updateHmStates(int row, int column){
45.         rmatrix.get(row).set(column, rmatrix.get(row).get(column).intValue()+1);
46.         return rmatrix;
47.     }
48.
49.     /**
50.      * @return the rmatrix
51.      */
52.     public Vector<Vector<Integer>> getRmatrix() {
53.         return rmatrix;
54.     }
55.
56.     /**
57.      * @param rmatrix the rmatrix to set
58.      */
59.     public void setRmatrix(Vector<Vector<Integer>> rmatrix) {
60.         this.rmatrix = rmatrix;
61.     }
62.
63.     /**
64.      * @return the rClosenessMatrix
65.      */
66.     public Vector<Vector<Integer>> getrClosenessMatrix() {
67.         return rClosenessMatrix;
68.     }
69.
70.     /**
71.      * @param rClosenessMatrix the rClosenessMatrix to set
72.      */
73.     public void setrClosenessMatrix(Vector<Vector<Integer>> rClosenessMatrix) {
74.         this.rClosenessMatrix = rClosenessMatrix;
75.     }
76. }
```

A.3 Testing Data Generation

A.3.1 Main Entry for Testing Data Generation

```
1.  package context.utility;
2.
3.  import java.io.BufferedWriter;
4.  import java.io.FileWriter;
5.  import java.io.IOException;
6.  import java.text.DateFormat;
7.  import java.text.ParseException;
8.  import java.text.SimpleDateFormat;
9.  import java.util.ArrayList;
10. import java.util.Arrays;
11. import java.util.Calendar;
12. import java.util.Collections;
13. import java.util.HashMap;
14. import java.util.HashSet;
15. import java.util.Iterator;
16. import java.util.List;
17. import java.util.Map;
18. import java.util.Random;
19. import java.util.Set;
20. import java.time.Duration;
21. import java.time.Instant;
22.
23. /**
24.  * This class is used to generate random context and situation
25.  * @author syue2
26.  *
27.  */
28. public class RandomGenerator {
29.     // class variable
30.     final static String lexicon = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
31.
32.     final static java.util.Random rand = new java.util.Random();
33.
34.     // consider using a Map<String,Boolean> to say whether the identifier is being used or not
35.     final static Set<String> identifiers = new HashSet<String>();
36.     final static Map<String, String> location = new HashMap<String, String>();
37.
38.     public static void main (String args[]) throws ParseException, IOException{
39.
40.         //location.put("BuildingEntrance", "location00");
41.         //location.put("OnWayToElevator", "location01");
42.         //location.put("AtTheFrontOfElevator", "location02");
43.         location.put("HOME", "location03");
44.         location.put("GYM", "location04");
45.         location.put("DINNINGHALL", "location05");
46.         location.put("COFFEESHOP", "location06");
47.         location.put("ElevatorBuilding", "location07");
48.
49.         BufferedWriter writer = new BufferedWriter(new FileWriter("config/samplefile1.txt"));
50.         //System.out.println(randomIdentifier());
51.         TimeHelper th = new TimeHelper("17-Jan-18 02:10:15");
52.         PersonHelper ph = new PersonHelper();
53.         Instant start = Instant.now();
54.         for(int i=0; i<1000000; i++){
55.             Iterator<String> it = ph.getNextPerson().iterator();
```

```
            //String[] s= randomLocation();
             writer.write(i                  //whole ID
                    +", "+it.next()          //person ID subject URI
                    +", "+it.next()          //name
                    +", "+it.next()       //person type
                    +", "+th.getNextDate()     //date
                    +", "+randomResult()       //decision
                    +", "+it.next()          //action
                    +", "+it.next()          //predicate URI
                    +", "+"action"        //location type
                    +", "+it.next()    //location URI
                    +", "+it.next()    //location NAME
                    //+", "+"person"          //person type
                    +", "+"location");        //location type
             writer.newLine();
          }
          Instant end = Instant.now();
          Duration timeElapsed = Duration.between(start, end);
          System.out.println("Time taken: "+ timeElapsed.toMillis() +" milliseconds");
          writer.close();
       }

       //https://stackoverflow.com/questions/5025651/java-randomly-generate-distinct-names
       private static String randomIdentifier() {
          StringBuilder builder = new StringBuilder();
          while(builder.toString().length() == 0) {
             int length = rand.nextInt(5)+5;
             for(int i = 0; i < length; i++) {
                builder.append(lexicon.charAt(rand.nextInt(lexicon.length())));
             }
             if(identifiers.contains(builder.toString())) {
                builder = new StringBuilder();
             }
          }
          return builder.toString();
       }


       //http://wcstitbits.blogspot.com/2012/05/generating-random-datetime-between-two.html
       public static String randomTime() throws ParseException{
          DateFormat formatter = new SimpleDateFormat("dd-MMM-yy HH:mm:ss");
          Calendar cal=Calendar.getInstance();
          String str_date1="17-Nov-18 02:10:15";
          String str_date2="27-Nov-18 02:10:20";

          cal.setTime(formatter.parse(str_date1));
          Long value1 = cal.getTimeInMillis();

          cal.setTime(formatter.parse(str_date2));
          Long value2 = cal.getTimeInMillis();

          long value3 = (long)(value1 + Math.random()*(value2 - value1));
          cal.setTimeInMillis(value3);
          return formatter.format(cal.getTime());
       }

       //We only consider three types of location here.
       enum Location
       {
          HOME, GYM, DINNINGHALL, COFFEESHOP, ElevatorBuilding;
       }
```

```
117.
118.    private static final List<Location> VALUES =
119.    Collections.unmodifiableList(Arrays.asList(Location.values()));
120.    private static final int enumSize = VALUES.size();
121.    private static final int size = location.size();
122.    private static final Random RANDOM = new Random();
123.
124.    public static String[] randomLocation(){
125.
126.        String tmp = VALUES.get(RANDOM.nextInt(enumSize)).toString();
127.        Object randomName = location.get(tmp);
128.        //System.out.print(randomName);
129.        String[] s = {randomName.toString(),tmp};
130.        return s;
131.
132.    }
133.
134.    //either take(the elevator) or pass(the elevator)
135.    enum RESULT
136.    {
137.        TAKE, PASS;
138.    }
139.    private static final List<RESULT> VALUES2 =
140.    Collections.unmodifiableList(Arrays.asList(RESULT.values()));
141.    private static final int SIZE2 = VALUES2.size();
142.
143.    public static RESULT randomResult(){
144.        return VALUES2.get(RANDOM.nextInt(SIZE2));
145.    }
146.
147.  //either take(the elevator) or pass(the elevator)
148.    enum NAME
149.    {
150.        Mike, Frank, Joshua, Ted, Alice;
151.    }
152.    private static final List<NAME> VALUES3 =
153.    Collections.unmodifiableList(Arrays.asList(NAME.values()));
154.    private static final int SIZE3 = VALUES3.size();
155.
156.    public static NAME randomName(){
157.        return VALUES3.get(RANDOM.nextInt(SIZE3));
158.    }
159.
160.    //to insert a record to a CSM, the program need to
161.
162. }
```

A.3.2 PersonHelper.java

```
1.    package context.utility;
2.
3.    import java.util.ArrayList;
4.    import java.util.Arrays;
5.    import java.util.Collections;
6.    import java.util.HashMap;
7.    import java.util.HashSet;
8.    import java.util.List;
```

```
9.  import java.util.Map;
10. import java.util.Random;
11. import java.util.Set;
12.
13. import context.utility.RandomGenerator.Location;
14. import context.utility.RandomGenerator.RESULT;
15.
16. public class PersonHelper {
17.     Set <List<String>> personLocation=new HashSet<List<String>>();
18.     final static Map<String, String> location = new HashMap<String, String>();
19.
20.     enum NAME
21.     {
22.         Mike, Frank, Joshua, Ted, Alice;
23.     }
24.      enum Location
25.     {
26.         HOME, GYM, DINNINGHALL, COFFEESHOP, SEC;
27.     }
28.
29.     private static final List<Location> VALUES =
30.             Collections.unmodifiableList(Arrays.asList(Location.values()));
31.     private static final int enumSize = VALUES.size();
32.     private static final int size = location.size();
33.     private static final Random RANDOM = new Random();
34.
35.     PersonHelper(){
36.         location.put("HOME", "location03");
37.         location.put("GYM", "location04");
38.         location.put("DINNINGHALL", "location05");
39.         location.put("COFFEESHOP", "location06");
40.         location.put("SEC", "location07");
41.         List<String> a = new ArrayList<String>();
42.         String[] s = randomLocation();
43.         a.add("p001");
44.         a.add("Mike");
45.         a.add("Professor");
46.         a.add("leave");
47.         a.add("action01");
48.         a.add(s[0]); //uri of location
49.         a.add(s[1]); //name of location
50.         personLocation.add(a);
51.         a = new ArrayList<String>();
52.         s = randomLocation();
53.         a.add("p002");
54.         a.add("Frank");
55.         a.add("Student");
56.         a.add("leave");
57.         a.add("action01");
58.         a.add(s[0]); //uri of location
59.         a.add(s[1]); //name of location
60.         personLocation.add(a);
61.         a = new ArrayList<String>();
62.         s = randomLocation();
63.         a.add("p003");
64.         a.add("Joshua");
65.         a.add("Professor");
66.         a.add("leave");
67.         a.add("action01");
68.         a.add(s[0]); //uri of location
69.         a.add(s[1]); //name of location
```

```
70.         personLocation.add(a);
71.         a = new ArrayList<String>();
72.         s = randomLocation();
73.         a.add("p004");
74.         a.add("Ted");
75.         a.add("Student");
76.         a.add("leave");
77.         a.add("action01");
78.         a.add(s[0]); //uri of location
79.         a.add(s[1]); //name of location
80.         personLocation.add(a);
81.         a = new ArrayList<String>();
82.         s = randomLocation();
83.         a.add("p005");
84.         a.add("Alice");
85.         a.add("Student");
86.         a.add("leave");
87.         a.add("action01");
88.         a.add(s[0]); //uri of location
89.         a.add(s[1]); //name of location
90.         personLocation.add(a);
91.     }
92.
93.
94.     public List getNextPerson(){
95.
96.
97.         int size = personLocation.size();
98.         int item = new Random().nextInt(size); // In real life, the Random object should be rather more shared than this
99.         int i = 0;
100.        List<String> temp1 = new ArrayList<String>();
101.        List<String> temp2 = new ArrayList<String>();
102.        for(List obj : personLocation) //this step is assign temp1 and temp2 with obj
103.        {
104.            if (i == item){
105.                temp1 = obj;
106.                temp2 = obj;
107.            }
108.            i++;
109.        }
110.        if(temp1.contains("leave")){
111.            String[] s;
112.            if("GYM".equals(temp1.get(3))){
113.                System.out.println("test");
114.                s = randomLocationWithout_GYM();
115.            }else if("DINNINGHALL".equals(temp1.get(3))){
116.                s = randomLocationWithout_DINNINGHALL();
117.            }else if("COFFEESHOP".equals(temp1.get(3))){
118.                s = randomLocationWithout_COFFEESHOP();
119.            }else if("SEC".equals(temp1.get(3))){
120.                s = randomLocationWithout_SEC();
121.            }else if("HOME".equals(temp1.get(3))){
122.                s = randomLocationWithout_HOME();
123.            }else{
124.                s = randomLocation();
125.            }
126.            temp1.remove("leave");
127.            temp1.remove("action01");
128.            temp1.remove(3); //location URI
129.            temp1.remove(3); //location name
130.            temp1.add("arrive");
```

```
131.            temp1.add("action02");
132.            temp1.add(s[0]);
133.            temp1.add(s[1]);
134.        }else{
135.            String s2=temp1.get(4);
136.            String s3 =temp1.get(5);
137.            temp1.remove("arrive");
138.            temp1.remove("action02");
139.            temp1.remove(3); //location URI
140.            temp1.remove(3); //location name
141.            temp1.add("leave");
142.            temp1.add("action01");
143.            temp1.add(s2);
144.            temp1.add(s3);
145.        }
146.        personLocation.remove(temp2);
147.        personLocation.add(temp1);
148.        return temp2;
149.    }
150.
151.    //We only consider three types of location here.
152.
153.    public static String[] randomLocation(){
154.
155.        String tmp = VALUES.get(RANDOM.nextInt(enumSize)).toString();
156.        Object randomName = location.get(tmp);
157.        //System.out.print(randomName);
158.        String[] s = {randomName.toString(),tmp};
159.        return s;
160.
161.    }
162.
163.    enum LocationWithout_HOME
164.    {
165.        GYM, DINNINGHALL, COFFEESHOP, SEC;
166.    }
167.    private static final List<LocationWithout_HOME> VALUESWithout_HOME =
168.            Collections.unmodifiableList(Arrays.asList(LocationWithout_HOME.values()));
169.    private static final int enumSizeWithout_HOME = VALUESWithout_HOME.size();
170.    public static String[] randomLocationWithout_HOME(){
171.        String tmp = VALUESWithout_HOME.get(RANDOM.nextInt(enumSizeWithout_HOME)).toString();
172.        Object randomName = location.get(tmp);
173.        //System.out.print(randomName);
174.        String[] s = {randomName.toString(),tmp};
175.        return s;
176.    }
177.
178.    enum LocationWithout_GYM
179.    {
180.        HOME, DINNINGHALL, COFFEESHOP, SEC;
181.    }
182.    private static final List<LocationWithout_GYM> VALUESWithout_GYM =
183.            Collections.unmodifiableList(Arrays.asList(LocationWithout_GYM.values()));
184.    private static final int enumSizeWithout_GYM = VALUESWithout_GYM.size();
185.    public static String[] randomLocationWithout_GYM(){
186.        String tmp = VALUESWithout_GYM.get(RANDOM.nextInt(enumSizeWithout_GYM)).toString();
187.        Object randomName = location.get(tmp);
188.        //System.out.print(randomName);
189.        String[] s = {randomName.toString(),tmp};
190.        return s;
191.    }
```

```
192.
193.    enum LocationWithout_COFFEESHOP
194.    {
195.        HOME, DINNINGHALL, GYM, SEC;
196.    }
197.    private static final List<LocationWithout_COFFEESHOP> VALUESWithout_COFFEESHOP =
198.            Collections.unmodifiableList(Arrays.asList(LocationWithout_COFFEESHOP.values()));
199.    private static final int enumSizeWithout_COFFEESHOP = VALUESWithout_COFFEESHOP.size();
200.    public static String[] randomLocationWithout_COFFEESHOP(){
201.        String tmp = VALUESWithout_COFFEESHOP.get(RANDOM.nextInt(enumSizeWithout_COFFEESHOP)).toStr
    ing();
202.        Object randomName = location.get(tmp);
203.        //System.out.print(randomName);
204.        String[] s = {randomName.toString(),tmp};
205.        return s;
206.    }
207.
208.    enum LocationWithout_SEC
209.    {
210.        HOME, DINNINGHALL, GYM, COFFEESHOP;
211.    }
212.    private static final List<LocationWithout_SEC> VALUESWithout_SEC =
213.            Collections.unmodifiableList(Arrays.asList(LocationWithout_SEC.values()));
214.    private static final int enumSizeWithout_SEC = VALUESWithout_SEC.size();
215.    public static String[] randomLocationWithout_SEC(){
216.        String tmp = VALUESWithout_SEC.get(RANDOM.nextInt(enumSizeWithout_SEC)).toString();
217.        Object randomName = location.get(tmp);
218.        //System.out.print(randomName);
219.        String[] s = {randomName.toString(),tmp};
220.        return s;
221.    }
222.
223.    enum LocationWithout_DINNINGHALL
224.    {
225.        HOME, SEC, GYM, COFFEESHOP;
226.    }
227.    private static final List<LocationWithout_DINNINGHALL> VALUESWithout_DINNINGHALL =
228.            Collections.unmodifiableList(Arrays.asList(LocationWithout_DINNINGHALL.values()));
229.    private static final int enumSizeWithout_DINNINGHALL = VALUESWithout_DINNINGHALL.size();
230.    public static String[] randomLocationWithout_DINNINGHALL(){
231.        String tmp = VALUESWithout_DINNINGHALL.get(RANDOM.nextInt(enumSizeWithout_DINNINGHALL)).t
    oString();
232.        Object randomName = location.get(tmp);
233.        //System.out.print(randomName);
234.        String[] s = {randomName.toString(),tmp};
235.        return s;
236.    }
237.
238. }
```

A.3.3 ListHelper.java

```
1.  package context.utility;
2.
3.  import java.util.Iterator;
4.  import java.util.List;
5.
```

```
6.  import context.core.ContextAttributeSnapshot;
7.  import context.core.ContextSituationState;
8.
9.  public class ListHelper {
10.
11.     //String, and template
12.     public static boolean notInList(Object o, List list){
13.         Iterator it = list.iterator();
14.         while(it.hasNext()){
15.             Object tmp = (Object)it.next();
16.             if(o.toString().equals(tmp.toString())){
17.                 return true;
18.             }
19.         }
20.         return false;
21.     }
22.
23.     /**
24.      * Only is used to compare the contextAttributeSnapshotList
25.      * @param s1 it is the new one
26.      * @param s2 it is the old one
27.      * @return
28.      */
29.     public static boolean cssEqual(ContextSituationState s1, ContextSituationState s2){
30.         boolean tag = true;
31.         Iterator it = s1.getContextAttributeSnapshotList().iterator();
32.         while(it.hasNext()){
33.             ContextAttributeSnapshot tmp = (ContextAttributeSnapshot)it.next();
34.             if(inTheSnapshotList(tmp, s2.getContextAttributeSnapshotList())){
35.                 continue;
36.             }else{
37.                 tag = false;
38.                 break;
39.             }
40.         }
41.         return tag;
42.     }
43.
44.     /**
45.      *
46.      * @param s1
47.      * @param snaplist
48.      * @return
49.      */
50.     public static boolean inTheSnapshotList(ContextAttributeSnapshot s1, List<ContextAttributeSnapshot> snaplist){
51.         boolean tag = false;
52.         Iterator it = snaplist.iterator();
53.         while(it.hasNext()){
54.             ContextAttributeSnapshot tmp = (ContextAttributeSnapshot)it.next();
55.             //System.out.print(tmp.getStateURI()+"    ");
56.             //System.out.println(s1.getStateURI());
57.             if(tmp.getStateURI()!=null&&tmp.getAttributeURI().equals(s1.getAttributeURI())
58.                     &&tmp.getObjectURI().equals(s1.getObjectURI())
59.                     &&tmp.getStateURI().equals(s1.getStateURI())){
60.                 tag=true;
61.             }
62.         }
63.         return tag;
64.     }
65. }
```

A.3.4 ConvertFileToClassToJson.java

```
1.  package context.convertion;
2.
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.io.PrintWriter;
8.  import java.time.Duration;
9.  import java.time.Instant;
10. import java.util.ArrayList;
11. import java.util.List;
12.
13. import com.fasterxml.jackson.databind.ObjectMapper;
14. import com.fasterxml.jackson.databind.ObjectWriter;
15.
16. public class ConvertFileToClassToJson {
17.     public static void main (String args[]) throws FileNotFoundException, IOException, InterruptedException{
18.         // Read data from file
19.         FileReader fr;
20.         // List to collect PersonLocation objects
21.         List<Triple> personLocations = new ArrayList<Triple>();
22.         Instant start = Instant.now();
23.         try (BufferedReader br = new BufferedReader(new FileReader("config/samplefile1.txt"))) {
24.             // Read file line by line
25.             String line = "";
26.             int tmp = 0;
27.             while ((line = br.readLine()) != null) {
28.                 //Thread.sleep(1);
29.                 // Parse line to extract individual fields
30.                 String[] data = parseLine(line);
31.
32.                 // Create new Employee object
33.                 Triple personLocation = new Triple();
34.                 personLocation.setRID(data[0].trim());
35.                 Subject subject = new Subject();
36.                 subject.setSubjectURI(data[1].trim());
37.                 subject.setSubjectName(data[2].trim());
38.                 subject.setSubjectType(data[3].trim());
39.                 personLocation.setSubject(subject);
40.                 Predicate predicate = new Predicate();
41.                 predicate.setPredicateName(data[6].trim());
42.                 predicate.setPredicateURI(data[7].trim());
43.                 predicate.setPredicateType(data[8].trim());
44.                 personLocation.setPredicate(predicate);
45.                 Object object = new Object();
46.                 object.setObjectURI(data[9].trim());
47.                 object.setObjectName(data[10].trim());
48.                 object.setObjectType(data[11].trim());
49.                 personLocation.setObject(object);
50.                 Condition condition = new Condition();
51.                 condition.setTime(data[4].trim());
52.                 personLocation.setCondition(condition);
53.                 Decision decision = new Decision();
54.                 decision.setDecisionName(data[5].trim());
55.                 personLocation.setDecision(decision);
56.
57.                 // Add object to list
```

```
58.                 personLocations.add(personLocation);
59.                 //System.out.println(personLocation.getDecision());
60.                 //System.gc();
61.                 if(tmp == 200000){
62.                     break;
63.                 }
64.                 tmp++;
65.             }
66.             System.out.println("this is 1");
67.             System.gc();
68.             while ((line = br.readLine()) != null) {
69.                 //Thread.sleep(1);
70.                 // Parse line to extract individual fields
71.                 String[] data = parseLine(line);
72.
73.                 // Create new Employee object
74.                 Triple personLocation = new Triple();
75.                 personLocation.setRID(data[0].trim());
76.                 Subject subject = new Subject();
77.                 subject.setSubjectURI(data[1].trim());
78.                 subject.setSubjectName(data[2].trim());
79.                 subject.setSubjectType(data[3].trim());
80.                 personLocation.setSubject(subject);
81.                 Predicate predicate = new Predicate();
82.                 predicate.setPredicateName(data[6].trim());
83.                 predicate.setPredicateURI(data[7].trim());
84.                 predicate.setPredicateType(data[8].trim());
85.                 personLocation.setPredicate(predicate);
86.                 Object object = new Object();
87.                 object.setObjectURI(data[9].trim());
88.                 object.setObjectName(data[10].trim());
89.                 object.setObjectType(data[11].trim());
90.                 personLocation.setObject(object);
91.                 Condition condition = new Condition();
92.                 condition.setTime(data[4].trim());
93.                 personLocation.setCondition(condition);
94.                 Decision decision = new Decision();
95.                 decision.setDecisionName(data[5].trim());
96.                 personLocation.setDecision(decision);
97.
98.                 // Add object to list
99.                 personLocations.add(personLocation);
100.                //System.out.println(personLocation.getDecision());
101.                //System.gc();
102.                if(tmp == 400000){
103.                    break;
104.                }
105.            }
106.            System.out.println("this is 2");
107.            System.gc();
108.            while ((line = br.readLine()) != null) {
109.                //Thread.sleep(1);
110.                // Parse line to extract individual fields
111.                String[] data = parseLine(line);
112.
113.                // Create new Employee object
114.                Triple personLocation = new Triple();
115.                personLocation.setRID(data[0].trim());
116.                Subject subject = new Subject();
117.                subject.setSubjectURI(data[1].trim());
118.                subject.setSubjectName(data[2].trim());
```

```
119.            subject.setSubjectType(data[3].trim());
120.            personLocation.setSubject(subject);
121.            Predicate predicate = new Predicate();
122.            predicate.setPredicateName(data[6].trim());
123.            predicate.setPredicateURI(data[7].trim());
124.            predicate.setPredicateType(data[8].trim());
125.            personLocation.setPredicate(predicate);
126.            Object object = new Object();
127.            object.setObjectURI(data[9].trim());
128.            object.setObjectName(data[10].trim());
129.            object.setObjectType(data[11].trim());
130.            personLocation.setObject(object);
131.            Condition condition = new Condition();
132.            condition.setTime(data[4].trim());
133.            personLocation.setCondition(condition);
134.            Decision decision = new Decision();
135.            decision.setDecisionName(data[5].trim());
136.            personLocation.setDecision(decision);
137.
138.            // Add object to list
139.            personLocations.add(personLocation);
140.            //System.out.println(personLocation.getDecision());
141.            //System.gc();
142.            if(tmp == 600000){
143.               break;
144.            }
145.          }
146.          System.out.println("this is 3");
147.          System.gc();
148.          while ((line = br.readLine()) != null) {
149.             //Thread.sleep(1);
150.            // Parse line to extract individual fields
151.            String[] data = parseLine(line);
152.
153.            // Create new Employee object
154.            Triple personLocation = new Triple();
155.            personLocation.setRID(data[0].trim());
156.            Subject subject = new Subject();
157.            subject.setSubjectURI(data[1].trim());
158.            subject.setSubjectName(data[2].trim());
159.            subject.setSubjectType(data[3].trim());
160.            personLocation.setSubject(subject);
161.            Predicate predicate = new Predicate();
162.            predicate.setPredicateName(data[6].trim());
163.            predicate.setPredicateURI(data[7].trim());
164.            predicate.setPredicateType(data[8].trim());
165.            personLocation.setPredicate(predicate);
166.            Object object = new Object();
167.            object.setObjectURI(data[9].trim());
168.            object.setObjectName(data[10].trim());
169.            object.setObjectType(data[11].trim());
170.            personLocation.setObject(object);
171.            Condition condition = new Condition();
172.            condition.setTime(data[4].trim());
173.            personLocation.setCondition(condition);
174.            Decision decision = new Decision();
175.            decision.setDecisionName(data[5].trim());
176.            personLocation.setDecision(decision);
177.
178.            // Add object to list
179.            personLocations.add(personLocation);
```

```
180.            //System.out.println(personLocation.getDecision());
181.            //System.gc();
182.            if(tmp == 800000){
183.               break;
184.            }
185.         }
186.         System.out.println("this is 4");
187.         System.gc();
188.         while ((line = br.readLine()) != null) {
189.            //Thread.sleep(1);
190.            // Parse line to extract individual fields
191.            String[] data = parseLine(line);
192.
193.            // Create new Employee object
194.            Triple personLocation = new Triple();
195.            personLocation.setRID(data[0].trim());
196.            Subject subject = new Subject();
197.            subject.setSubjectURI(data[1].trim());
198.            subject.setSubjectName(data[2].trim());
199.            subject.setSubjectType(data[3].trim());
200.            personLocation.setSubject(subject);
201.            Predicate predicate = new Predicate();
202.            predicate.setPredicateName(data[6].trim());
203.            predicate.setPredicateURI(data[7].trim());
204.            predicate.setPredicateType(data[8].trim());
205.            personLocation.setPredicate(predicate);
206.            Object object = new Object();
207.            object.setObjectURI(data[9].trim());
208.            object.setObjectName(data[10].trim());
209.            object.setObjectType(data[11].trim());
210.            personLocation.setObject(object);
211.            Condition condition = new Condition();
212.            condition.setTime(data[4].trim());
213.            personLocation.setCondition(condition);
214.            Decision decision = new Decision();
215.            decision.setDecisionName(data[5].trim());
216.            personLocation.setDecision(decision);
217.
218.            // Add object to list
219.            personLocations.add(personLocation);
220.            //System.out.println(personLocation.getDecision());
221.            //System.gc();
222.            if(tmp == 1200000){
223.               break;
224.            }
225.         }
226.         System.out.println("this is 5");
227.
228.         // Further process your Employee objects...
229.      }
230.
231.      UpperLevelInfo uli = new UpperLevelInfo();
232.      uli.setDeviceId("android");
233.      uli.setDeviceName("any");
234.      uli.setDeviceType("smartphone");
235.      uli.setInfoList(personLocations);
236.      uli.setInfoType("any");
237.      uli.setDomainName("Smart Campus");
238.      uli.setDomainId("5");
239.      uli.setDomainURI("test");
240.
```

```
241.        //https://www.journaldev.com/2324/jackson-json-java-parser-api-example-tutorial
242.        ObjectWriter ow = new ObjectMapper().writer().withDefaultPrettyPrinter();
243.        String json = ow.writeValueAsString(uli);
244.        //System.out.print(json);
245.
246.        try (PrintWriter out = new PrintWriter("config/samplefile2.json")) {
247.            out.println(json);
248.        }
249.        Instant end = Instant.now();
250.        Duration timeElapsed = Duration.between(start, end);
251.        System.out.println("Time taken: "+ timeElapsed.toMillis() +" milliseconds");
252.    }
253.
254.    public static String[] parseLine(String line){
255.        String[] split = line.split(",");
256.        return  split;
257.    }
258.
259. }
```

A.4 Context Domain Initialization

A.4.1 Preparing the Context Domain Data

```
1.  {
2.    "domainName" : "SmartCampus",
3.    "domainId" : "1",
4.    "domainURI" : "test",
5.    "ccl" : [ {
6.       "categoryType" : "Place",
7.       "categoryURI" : "cpl001",
8.       "categoryName" : "Office"
9.     },
10.    {
11.      "categoryType" : "Place",
12.      "categoryURI" : "cpl002",
13.      "categoryName" : "Classroom"
14.    },
15.    {
16.      "categoryType" : "Place",
17.      "categoryURI" : "cpl003",
18.      "categoryName" : "Building"
19.    },
20.    {
21.      "categoryType" : "Person",
22.      "categoryURI" : "cpe001",
23.      "categoryName" : "Student"
24.    },
25.    {
26.      "categoryType" : "Person",
27.      "categoryURI" : "cpe002",
28.      "categoryName" : "Professor"
29.    },
30.    {
31.      "categoryType" : "Device",
32.      "categoryURI" : "cd001",
```

```
      "categoryName" : "SmartElevator"
    },
    {
      "categoryType" : "Academic",
      "categoryURI" : "ca001",
      "categoryName" : "Course"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr001",
      "categoryName" : "Friend"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr002",
      "categoryName" : "Colleague"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr003",
      "categoryName" : "Advisor"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr004",
      "categoryName" : "Passenger"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr005",
      "categoryName" : "happensAt"
    },
    {
      "categoryType" : "Relationship",
      "categoryURI" : "cr000",
      "categoryName" : "Normal"
    },
    {
      "categoryType" : "Action",
      "categoryURI" : "caction001",
      "categoryName" : "Decision"
    },
    {
      "categoryType" : "Time",
      "categoryURI" : "ctime001",
      "categoryName" : "TimeStamp"
    },
    {
      "categoryType" : "Time",
      "categoryURI" : "ctime002",
      "categoryName" : "Hour24"
    },
    {
      "categoryType" : "Time",
      "categoryURI" : "ctime003",
      "categoryName" : "Day31"
    },
    {
      "categoryType" : "Time",
      "categoryURI" : "ctime004",
      "categoryName" : "Month12"
```

```
    },
    {
      "categoryType" : "Time",
      "categoryURI" : "ctime005",
      "categoryName" : "Year"
    }
    ]
}
```

A.4.2 Processing the Context Domain Initialization

```
package context.actuator;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import org.json.JSONException;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;

import context.core.ContextDomain;

public class DomainInitialization {
    public static void main(String args[]) throws JSONException, IOException{
        ObjectMapper mapper = new ObjectMapper();
        //JSON from file to Object
        ContextDomain cd = mapper.readValue(new File("config/domainInitialization.json"), ContextDomain.class);

        ObjectWriter ow = new ObjectMapper().writer().withDefaultPrettyPrinter();
        String json = ow.writeValueAsString(cd);
        File file = new File("dataFiles2");
        if(!file.exists()){
            file.mkdirs();
        }
        try (FileWriter fileWriter = new FileWriter("dataFiles/casmfile_domain_"+5+".json")) {
            fileWriter.write(json);
            System.out.println("Successfully Copied JSON Object to File...");
        }
    }


}
```

A.4.3 Data for Relationship Registration

```
{
  "infoType" : "relationshipRegistration",
  "deviceType" : "smartphone",
  "deviceName" : "any",
  "deviceId" : "android",
```

```
  "domainName" : "Smart Campus",
  "domainId" : "5",
  "domainURI" : "test",
  "infoList" : [ {
   "subject" : {
    "subjectType" : "Professor",
    "subjectURI" : "p001",
    "subjectName" : "Mike"
   },
   "object" : {
    "objectType" : "Student",
    "objectName" : "Ted",
    "objectURI" : "p004"
   },
   "predicate" : {
    "predicateType" : "relationshipRegistration",
    "predicateURI" : "relation01",
    "predicateName" : "Advise"
   },
   "closeness" : 80,
   "rid" : "0"
  }
  ]
}
```

A.4.4 Relationship Registration

```
package context.actuator;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;

import context.core.ContextDomain;
import context.core.RelationshipPair;

public class RelationshipRegistration {
    public static void main(String args[]) throws JSONException, IOException{
        ObjectMapper mapper = new ObjectMapper();
        //JSON from file to Object
        //ContextDomain cd1 = mapper.readValue(new File("config/relationshipRegistration.json"), ContextDomain.class
);
        JSONObject obj = JasonReader.readJson("config/relationshipRegistration.json");
        JSONArray a = obj.getJSONArray("infoList");

        //handle domain of the container
        //container contains all state machine for the input; context domain is the domain
        String domainId = obj.getString("domainId");

        ContextDomain cd1 = mapper.readValue(new File("dataFiles/casmfile_domain_"+5+".json"), ContextDomain.cla
ss);
```

```
31.
32.         //the first step is to register the relationship type; then assign the both people with the type;
33.         String relationName, relationURI, subjectURI, objectURI;
34.         int closeness = 0;
35.         for (int i = 0; i < a.length(); i++) {
36.             //handle context object
37.             JSONObject o = a.getJSONObject(i);
38.             relationName=o.getJSONObject("predicate").getString("predicateName");
39.             relationURI=o.getJSONObject("predicate").getString("predicateURI");
40.             Iterator iter = null;
41.             if(cd1.getCr().getRelationshipTypeList()!=null&&cd1.getCr().getRelationshipTypeList().size()>0){
42.                 iter= cd1.getCr().getRelationshipTypeList().iterator();
43.             }
44.             boolean tag = true;
45.             while(iter!=null&&iter.hasNext()){
46.                 RelationshipPair rp= (RelationshipPair)iter.next();
47.                 if (relationURI.equals(rp.getRelationURI())){
48.                     tag =false;
49.                     break;
50.                 }
51.             }
52.
53.             if(tag){//this type is a new type, not in the list yet
54.                 cd1.getCr().getRelationshipTypeList().add(new RelationshipPair(relationName, relationURI));
55.             }
56.
57.             //get the index of the new relationship type so that it would be the value of rmatrix
58.             Iterator it2 = cd1.getCr().getRelationshipTypeList().iterator();
59.             int index = -1;
60.             int tmp = -1;
61.             while(it2!=null&&it2.hasNext()){
62.                 tmp++;
63.                 RelationshipPair rp= (RelationshipPair)it2.next();
64.                 if (relationURI.equals(rp.getRelationURI())){
65.                     index = tmp; //this index is important
66.                 }
67.             }
68.             subjectURI = o.getJSONObject("subject").getString("subjectURI");
69.             objectURI = o.getJSONObject("object").getString("objectURI");
70.             int x = cd1.getCol_domain().indexOf(subjectURI);
71.             int y = cd1.getCol_domain().indexOf(objectURI);
72.
73.             cd1.getCr().getrMatrixes().getRmatrix().get(x).set(y, index);
74.             cd1.getCr().getrMatrixes().getRmatrix().get(y).set(x, index);
75.
76.             closeness = o.getInt("closeness");
77.             cd1.getCr().getrMatrixes().getrClosenessMatrix().get(x).set(y, closeness);
78.             cd1.getCr().getrMatrixes().getrClosenessMatrix().get(y).set(x, closeness);
79.         }
80.
81.         ObjectWriter ow = new ObjectMapper().writer().withDefaultPrettyPrinter();
82.         String json = ow.writeValueAsString(cd1);
83.         File file = new File("dataFiles2");
84.         if(!file.exists()){
85.             file.mkdirs();
86.         }
87.         try (FileWriter fileWriter = new FileWriter("dataFiles/casmfile_domain_"+5+".json")) {
88.             fileWriter.write(json);
89.             System.out.println("Successfully Copied JSON Object to File...");
90.         }
91.     }
```

```
92.
93. }
```

A.5 CASM/CSSM Assembling

A.5.1 CASM Assembling

```
1.  public static ContextDomain assembleCASM(JSONArray a, ContextDomain cd) throws JSONException{
2.
3.      //handle contextObject, ContextAttributeStateMachine List and ContextAttribute List
4.      ContextAttributeStateMachine casm;
5.      ContextCategory cc = new ContextCategory();
6.      ContextObject co;
7.      ContextAttribute ca;
8.      ContextAttributeState cas;
9.
10.     for (int i = 0; i < a.length(); i++) {
11.       //handle context object
12.       JSONObject o = a.getJSONObject(i);
13.
14.       //if the person leaves a place, then we do not consider this as a new object and a new place state, because arrive
    is the first
15.       if("leave".equals(o.getJSONObject("predicate").getString("predicateName"))){
16.         continue;
17.       }
18.
19.       //here to handle context category
20.       //1: check the existence of the context category, if not, give an error in terminal
21.       //2: if exist, then modify that category, add or update a context object of that category
22.       // subjectType/objectType/PredicateType are mapped to CategoryName , subject in triple is mapped to an object

23.       Iterator itcc = cd.getCcl().iterator();
24.       boolean bcc = true; //suppose its new category at first, if false later, then okay, otherwise, break and give error
25.       ContextCategory cotmpcc = new ContextCategory();
26.       String subjectTypeCategoryName = o.getJSONObject("subject").getString("subjectType");
27.       Iterator it1;
28.       while(itcc.hasNext()){
29.         cotmpcc = (ContextCategory)itcc.next();
30.         if(subjectTypeCategoryName.equals(cotmpcc.getCategoryName())){
31.           //then find the object in this category and update it
32.           cc = cotmpcc;
33.           bcc = false;
34.         }
35.       }
36.
37.       if(bcc==true){
38.         System.out.println("the category of "'+subjectTypeCategoryName+"' does not exist, please double check and
    configure to add that context categroy first");
39.         break;
40.       }
41.
42.       //System.out.println(o.getJSONObject("subject").getString("subjectName"));
43.       //TODO: add check to the json file that the two attributes can not be null
44.       co = new ContextObject(o.getJSONObject("subject").getString("subjectName"),
45.           o.getJSONObject("subject").getString("subjectURI"));
46.       int decision = "TAKE".equals(o.getJSONObject("decision").getString("decisionName"))?1:0;
```

```
47.
48.             it1 = cc.getCol().iterator();
49.             boolean b = false; // new object
50.             ContextObject cotmp = new ContextObject();
51.             while(it1.hasNext()){
52.                 cotmp = (ContextObject)it1.next();
53.                 if(cotmp.getObjectURI().equals(co.getObjectURI())){
54.                     b=true; // already has the object
55.                 }
56.             }
57.
58.             if(!b){//this situation is that the context object is a new one.
59.                 cd.getCol_domain().add(co.getObjectURI());//this is to add new object in context domain
60.                 cd.getCr().getrMatrixes().addState(0);
61.                 cc.addContextObject(co);
62.
63.                 //then the context attribute will be very new and its the type in the object
64.                 //???so context object is a real object but context attribute is a concept
65.                 //???how to draw in uml class diagram need to be considered
66.                 ca = new ContextAttribute(o.getJSONObject("object").getString("objectType"));
67.                 co.addContextAttribute(ca);//append the attribute to the list cal in the co.
68.
69.                 //handle the context attribute state
70.                 JSONObject o2 = a.getJSONObject(i);
71.                 //System.out.println(o.getJSONObject("subject").getString("subjectName"));
72.                 //TODO: add check to the json file that the two attributes can not be null
73.                 cas = new ContextAttributeState(o2.getJSONObject("object").getString("objectName"),
74.                         o2.getJSONObject("object").getString("objectURI"));
75.
76.                 Iterator it2 = ca.getContextAttributeStatesList()!=null?
77.                         ca.getContextAttributeStatesList().iterator():null;
78.                 boolean b2 = false; // new object state
79.                 ContextAttributeState castmp = new ContextAttributeState();
80.                 while(it2!=null&&it2.hasNext()){
81.                     castmp = (ContextAttributeState)it2.next();
82.                     if(castmp.getStateURI().equals(cas.getStateURI())){
83.                         b2=true; // already has the object state
84.                     }
85.                 }
86.
87.                 if(!b2){
88.                     ca.addContextAttributeStateToList(cas);
89.                 }
90.                 //the context attribute state machine will be very new too
91.                 casm = new ContextAttributeStateMachine();
92.                 //casm.setCa(ca);
93.
94.                 int col = 0;//used to note the incoming state and for the parameter of matrix
95.
96.                 //handle hmstates (to store the map of the name and the number of states)
97.                 //handle matrixes (to store the value of the casm)
98.                 if(casm.getHmstates().containsKey(cas.getStateURI())){ //then edit matrix
99.                     col = casm.getHmstates().get(cas.getStateURI());
100.                    casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
101.                    casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column, dec
    ision
102.                    casm.setCurrentState(col);
103.                } else{ //then add a state; here since the object is new, then the state should be the first one, and the size shoul
    d be zero
104.                    casm.addtoHashMapStates(0, cas.getStateURI());
105.                    casm.getMatrixes().addState(casm.getCurrentState()); // use current state to find the original state
```

```
106.                 casm.getMatrixes().addStateForDeicision(casm.getCurrentState(),decision);
107.                 casm.setCurrentState(0); //the first state of the first object
108.             }
109.             co.addAttributeStateMachine(casm);
110.
111.         }else {//here the context object is not new. and we just need to search and update
112.             Iterator it3 = cc.getCol().iterator();
113.             boolean b3 = false; // new object
114.             ContextObject cotmp3 = new ContextObject();
115.             while(it3.hasNext()){
116.                 cotmp3 = (ContextObject)it3.next();
117.                 if(cotmp3.getObjectURI().equals(co.getObjectURI())){
118.                     //found the object and update it
119.                     //TODO: Did not handle context attribute here
120.                     //ca = new ContextAttribute(o.getJSONObject("object").getString("objectType"));
121.                     Iterator it4=cotmp3.getCal()!=null?cotmp3.getCal().iterator():null;
122.                     ContextAttribute catmp;
123.                     int it4int=0, tag=0;
124.                     while(it4.hasNext()){
125.                         catmp = (ContextAttribute)it4.next();
126.                         if(catmp.getAttributeName().equals(o.getJSONObject("object").getString("objectType"))){
127.                             it4int=tag;
128.                         }
129.                         tag++;
130.                     }
131.                     ca=cotmp3.getCal().get(it4int);
132.
133.                     //handle the context attribute state
134.                     JSONObject o2 = a.getJSONObject(i);
135.                     //System.out.println(o.getJSONObject("subject").getString("subjectName"));
136.                     //TODO: add check to the json file that the two attributes can not be null
137.                     cas = new ContextAttributeState(o2.getJSONObject("object").getString("objectName"),
138.                             o2.getJSONObject("object").getString("objectURI"));
139.
140.                     Iterator it2 = ca.getContextAttributeStatesList()!=null?
141.                             ca.getContextAttributeStatesList().iterator():null;
142.                     boolean b2 = false; // new object state
143.                     ContextAttributeState castmp = new ContextAttributeState();
144.                     while(it2!=null&&it2.hasNext()){
145.                         castmp = (ContextAttributeState)it2.next();
146.                         if(castmp.getStateURI().equals(cas.getStateURI())){
147.                             b2=true; // already has the object state
148.                         }
149.                     }
150.
151.                     if(!b2){
152.                         //System.out.println("this is the place 1");
153.                         ca.addContextAttributeStateToList(cas);
154.                     }
155.                     //cotmp3.addContextAttribute(ca);
156.                     //the context attribute state machine will be very new too
157.                     casm = cotmp3.getCasml().get(it4int);
158.                     //casm.setCa(ca);
159.
160.                     int col = 0;//used to note the incoming state and for the parameter of matrix
161.
162.                     //handle hmstates (to store the map of the name and the number of states)
163.                     //handle matrixes (to store the value of the casm)
164.                     if(casm.getHmstates().containsKey(cas.getStateURI())){ //if the state has already been recorded, then ed
    it matrix
165.                         col = casm.getHmstates().get(cas.getStateURI());
```

```
                    //System.out.println("row:" + casm.getCurrentState());
                    //System.out.println("col:" + col);
                    casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
                    casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column
, decision
                    //TODO: important: from current state to a new state, actions can be triggered here
                    // and the actions can be stored in a decision form in decision matrix
                    casm.setCurrentState(col);
                } else{
                    int size = casm.getHmstates().size();
                    casm.addtoHashMapStates(size, cas.getStateURI());
                    casm.getMatrixes().addState(casm.getCurrentState()); // use current state to find the original state
                    casm.getMatrixes().addStateForDeicision(casm.getCurrentState(),decision);
                    casm.setCurrentState(size);
                }
                //cotmp3.addAttributeStateMachine(casm);
            }
        }
    }

    //handle context attribute list

}

return cd;
}
```

A.5.2 CSSM Assembling

```
//this method is used to build the CASM and CSSM together
public static ContextDomain assembleCASMCSSM(JSONArray a, ContextDomain cd) throws JSONException{

    //handle contextObject, ContextAttributeStateMachine List and ContextAttribute List
    ContextAttributeStateMachine casm;
    ContextSituationStateMachine cssm;
    ContextCategory cc = new ContextCategory();
    ContextObject co;
    ContextAttribute ca;
    ContextAttributeState cas;
    //System.out.println("position 1");
    for (int i = 0; i < a.length(); i++) {
        //handle context object
        JSONObject o = a.getJSONObject(i);

        //if the person leaves a place, then we do not consider this as a new object and a new place state, because arrive is t
he first
        // and in our prototype of the research, we only consider to build CASM for arrive(behavior) predicate.
        // To extend the framework, we can have behavior class in the framework, and the behavior can be a lot of types
        //This is one of the limitation of the research; another one is that we do not have the object adding and transition sh
ift
        if("leave".equals(o.getJSONObject("predicate").getString("predicateName"))){
            continue;
        }

        //here to handle context category
        //1: check the existence of the context category, if not, give an error in terminal
        //2: if exist, then modify that category, add or update a context object of that category
```

```
27.         // subjectType/objectType/PredicateType are mapped to CategoryName , subject in triple is mapped to an object
28.         Iterator itcc = cd.getCcl().iterator();
29.         boolean bcc = true; //suppose its new category at first, if false later, then okay, otherwise, break and give error
30.         ContextCategory cotmpcc = new ContextCategory();
31.         String subjectTypeCategoryName = o.getJSONObject("subject").getString("subjectType");
32.         while(itcc.hasNext()){
33.             cotmpcc = (ContextCategory)itcc.next();
34.             if(subjectTypeCategoryName.equals(cotmpcc.getCategoryName())){
35.                 //then find the object in this category and update it
36.                 cc = cotmpcc;
37.                 bcc = false;
38.             }
39.         }
40.
41.         if(bcc==true){
42.             System.out.println("the category of "'+subjectTypeCategoryName+"' does not exist, please double check and co
    nfigure to add that context categroy first");
43.             break;
44.         }
45.
46.         //System.out.println(o.getJSONObject("subject").getString("subjectName"));
47.         //TODO: add check to the json file that the two attributes can not be null
48.         co = new ContextObject(o.getJSONObject("subject").getString("subjectName"),
49.                 o.getJSONObject("subject").getString("subjectURI"));
50.         int decision = "TAKE".equals(o.getJSONObject("decision").getString("decisionName"))?1:0;
51.
52.         Iterator it1;
53.         it1 = cc.getCol().iterator();
54.         boolean b = false; // new object, not existing
55.         ContextObject cotmp = new ContextObject();
56.         while(it1.hasNext()){
57.             cotmp = (ContextObject)it1.next();
58.             if(cotmp.getObjectURI().equals(co.getObjectURI())){
59.                 b=true; // already has the object
60.             }
61.         }
62.
63.         //From here on, We need to find the context situation state machine and search and update or initialize
64.         //here the context object is not new. and we just need to search and update
65.         //This situation state machine is only for the decision for using elevator, so we can have a list of situation state mac
    hine for different purpose,
66.         //which could be in the future work
67.
68.         //System.out.println("position 1.5");
69.         //we need to get the related object's current state and put them into the context situation state
70.         if(!b){//this situation is that the context object is a new one.
71.             cd.getCol_domain().add(co.getObjectURI());//this is to add new object in context domain
72.             cd.getCr().getrMatrixes().addState(0);
73.             cc.addContextObject(co);
74.
75.             //then the context attribute will be very new and its the type in the object
76.             //???so context object is a real object but context attribute is a concept
77.             //???how to draw in uml class diagram need to be considered
78.             ca = new ContextAttribute(o.getJSONObject("object").getString("objectType"));
79.             co.addContextAttribute(ca);//append the attribute to the list cal in the co.
80.
81.             //handle the context attribute state
82.             JSONObject o2 = a.getJSONObject(i);
83.             //System.out.println(o.getJSONObject("subject").getString("subjectName"));
84.             //TODO: add check to the json file that the two attributes can not be null
85.             cas = new ContextAttributeState(o2.getJSONObject("object").getString("objectName"),
```

```
86.                 o2.getJSONObject("object").getString("objectURI"));
87.
88.             Iterator it2 = ca.getContextAttributeStatesList()!=null?
89.                 ca.getContextAttributeStatesList().iterator():null;
90.             boolean b2 = false; // new object state
91.             ContextAttributeState castmp = new ContextAttributeState();
92.             while(it2!=null&&it2.hasNext()){
93.                 castmp = (ContextAttributeState)it2.next();
94.                 if(castmp.getStateURI().equals(cas.getStateURI())){
95.                     b2=true; // already has the object state
96.                 }
97.             }
98.
99.             if(!b2){
100.                ca.addContextAttributeStateToList(cas);
101.            }
102.            //the context attribute state machine will be very new too
103.            casm = new ContextAttributeStateMachine();
104.            //casm.setCa(ca);
105.
106.            int col = 0;//used to note the incoming state and for the parameter of matrix
107.
108.            //handle hmstates (to store the map of the name and the number of states)
109.            //handle matrixes (to store the value of the casm)
110.            if(casm.getHmstates().containsKey(cas.getStateURI())){ //then edit matrix
111.                col = casm.getHmstates().get(cas.getStateURI());
112.                casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
113.                casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column, decisi
    on
114.                casm.setCurrentState(col);
115.            } else{ //then add a state; here since the object is new, then the state should be the first one, and the size should
    be zero
116.                casm.addtoHashMapStates(0, cas.getStateURI());
117.                casm.getMatrixes().addState(casm.getCurrentState()); // use current state to find the original state
118.                casm.getMatrixes().addStateForDeicision(casm.getCurrentState(),decision);
119.                casm.setCurrentState(0); //the first state of the first object
120.            }
121.            co.addAttributeStateMachine(casm);
122.
123.        }else {//here the context object is not new. and we just need to search and update
124.            Iterator it3 = cc.getCol().iterator();
125.            boolean b3 = false; // new object
126.            ContextObject cotmp3 = new ContextObject();
127.            while(it3.hasNext()){
128.                cotmp3 = (ContextObject)it3.next();
129.                if(cotmp3.getObjectURI().equals(co.getObjectURI())){
130.                    //found the object and update it
131.
132.                    //handle CSSM
133.                    ContextSituationState css1= new ContextSituationState();
134.                    String objectURI = o.getJSONObject("subject").getString("subjectURI");
135.                    String attributeURI = o.getJSONObject("object").getString("objectType"); //here we use objectType
136.                    String stateURI = o.getJSONObject("object").getString("objectURI");
137.                    ContextAttributeSnapshot casnap = new ContextAttributeSnapshot(objectURI, attributeURI, stateURI);
138.                    css1.addContextAttributeSnapshotToList(casnap);
139.
140.                    //find a list of related objects by using relation matrix
141.                    Iterator itobjects = cd.getCol_domain().iterator();
142.                    int index = 0;
143.                    int tag2 = 0;
144.                    int length = cd.getCol_domain().size();
```

```
145.                 while(itobjects.hasNext()){
146.                     String co2= (String)itobjects.next();
147.                     if(objectURI.equals(co2)){
148.                         index = tag2; //the index of the object found here, which is the matrix row
149.                     }
150.                     tag2++;
151.                 }
152.                 for (int ii = 0; ii<length; ii++){
153.                     if(cd.getCr().getrMatrixes().getrClosenessMatrix().get(index).get(ii)>=80){
154.                         //then the relation is good enough, we add the current state of this object
155.                         String thisObject = cd.getCol_domain().get(ii);
156.                         Iterator itSS1 = cd.getCcl().iterator();
157.                         while(itSS1.hasNext()){
158.                             ContextCategory ccSS1 = (ContextCategory)itSS1.next();
159.                             Iterator itSS2 = ccSS1.getCol().iterator();
160.                             while(itSS2.hasNext()){
161.                                 ContextObject coSS2 = (ContextObject)itSS2.next();
162.                                 if(thisObject.equals(coSS2.getObjectURI())){
163.                                     //then the object here in this category is our target now
164.                                     //we will add the state and the last transition to this situation
165.                                     //then we iterate its attributes
166.                                     Iterator itSS3 = coSS2.getCasml().iterator();
167.                                     Iterator itSS4 = coSS2.getCal().iterator();
168.                                     while(itSS3.hasNext()){
169.                                         ContextAttributeStateMachine casm4 = (ContextAttributeStateMachine)itSS3.next();
170.                                         ContextAttribute ca4 = (ContextAttribute)itSS4.next();
171.                                         ContextAttributeSnapshot casnap4 = new ContextAttributeSnapshot(coSS2.getObjectURI()
    , ca4.getAttributeName(),casm4.extractCurrentStateName() );
172.                                         //here the state should be the URI not the name
173.                                         if(!ListHelper.inTheSnapshotList(casnap4, css1.getContextAttributeSnapshotList())){
174.                                             css1.addContextAttributeSnapshotToList(casnap4);
175.                                         }
176.                                     }
177.                                 }
178.                             }
179.                         }
180.                     }
181.                 }
182.                 int existAndIndex = cotmp3.existAndIndexInCSSL(css1);
183.                 if(existAndIndex>-1){ //existing, so we just have the index, and use the index to set current
184.                     //casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
185.                     //casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column,
    decision
186.                     cotmp3.getCssm().getSmatrixes().updateHmStates(cotmp3.getCssm().getCurrentState(), existAndIndex)
    ;
187.                     cotmp3.getCssm().getSmatrixes().updateDecisionMatrixStates(cotmp3.getCssm().getCurrentState(), exi
    stAndIndex, decision);
188.                     cotmp3.getCssm().setCurrentState(existAndIndex);
189.                 }else{
190.                     cotmp3.addContextSituationStateToList(css1);
191.                     cotmp3.getCssm().getSmatrixes().addState(cotmp3.getCssm().getCurrentState());
192.                     cotmp3.getCssm().getSmatrixes().addStateForDeicision(cotmp3.getCssm().getCurrentState(), decision);

193.                     int currentIndex = cotmp3.getCssl().size()-1;
194.                     css1.setId(currentIndex);
195.                     cotmp3.getCssm().setCurrentState(currentIndex);
196.                 }
197.
198.
199.                 //after mainly handly CSSM
200.                 Iterator it4=cotmp3.getCal()!=null?cotmp3.getCal().iterator():null;
```

```
201.                ContextAttribute catmp;
202.                int it4int=0, tag=0;
203.                while(it4.hasNext()){
204.                   catmp = (ContextAttribute)it4.next();
205.                   if(catmp.getAttributeName().equals(o.getJSONObject("object").getString("objectType"))){
206.                      it4int=tag;
207.                   }
208.                   tag++;
209.                }
210.                ca=cotmp3.getCal().get(it4int);
211.
212.                //handle the context attribute state
213.                JSONObject o2 = a.getJSONObject(i);
214.                //System.out.println(o.getJSONObject("subject").getString("subjectName"));
215.                //TODO: add check to the json file that the two attributes can not be null
216.                cas = new ContextAttributeState(o2.getJSONObject("object").getString("objectName"),
217.                      o2.getJSONObject("object").getString("objectURI"));
218.
219.                Iterator it2 = ca.getContextAttributeStatesList()!=null?
220.                      ca.getContextAttributeStatesList().iterator():null;
221.                boolean b2 = false; // new object state
222.                ContextAttributeState castmp = new ContextAttributeState();
223.                while(it2!=null&&it2.hasNext()){
224.                   castmp = (ContextAttributeState)it2.next();
225.                   if(castmp.getStateURI().equals(cas.getStateURI())){
226.                      b2=true; // already has the object state
227.                   }
228.                }
229.
230.                if(!b2){
231.                   //System.out.println("this is the place 1");
232.                   ca.addContextAttributeStateToList(cas);
233.                }
234.                //cotmp3.addContextAttribute(ca);
235.                //the context attribute state machine will be very new too
236.                casm = cotmp3.getCasml().get(it4int);
237.                //casm.setCa(ca);
238.
239.                int col = 0;//used to note the incoming state and for the parameter of matrix
240.
241.                //handle hmstates (to store the map of the name and the number of states)
242.                //handle matrixes (to store the value of the casm)
243.                if(casm.getHmstates().containsKey(cas.getStateURI())){ //if the state has already been recorded, then edit
    matrix
244.                   col = casm.getHmstates().get(cas.getStateURI());
245.                   //System.out.println("row:" + casm.getCurrentState());
246.                   //System.out.println("col:" + col);
247.                   casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
248.                   casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column, d
    ecision
249.                   //TODO: important: from current state to a new state, actions can be triggered here
250.                   // and the actions can be stored in a decision form in decision matrix
251.                   casm.setCurrentState(col);
252.                } else{
253.                   int size = casm.getHmstates().size();
254.                   casm.addtoHashMapStates(size, cas.getStateURI());
255.                   casm.getMatrixes().addState(casm.getCurrentState()); // use current state to find the original state
256.                   casm.getMatrixes().addStateForDeicision(casm.getCurrentState(),decision);
257.                   casm.setCurrentState(size);
258.                }
259.                //cotmp3.addAttributeStateMachine(casm);
```

```
260.             }
261.           }
262.       }
263.
264.
265.       Iterator it3 = cc.getCol().iterator();
266.       boolean b3 = false; // new object
267.       ContextObject cotmp3 = new ContextObject();
268.
269.
270.       while(it3.hasNext()){
271.          cotmp3 = (ContextObject)it3.next();
272.          if(cotmp3.getObjectURI().equals(co.getObjectURI())){
273.             //found the object and use it
274.             //we get the current attribute URI, and the current state URI, and the object URI
275.             Iterator it4=cotmp3.getCal()!=null?cotmp3.getCal().iterator():null;
276.             ContextAttribute catmp;
277.             int it4int=0, tag=0;
278.             while(it4.hasNext()){
279.                catmp = (ContextAttribute)it4.next();
280.                if(catmp.getAttributeName().equals(o.getJSONObject("object").getString("objectType"))){
281.                   it4int=tag;
282.                }
283.                tag++;
284.             }
285.             ca=cotmp3.getCal().get(it4int);
286.
287.             //handle the context attribute state
288.             JSONObject o2 = a.getJSONObject(i);
289.             //System.out.println(o.getJSONObject("subject").getString("subjectName"));
290.             //TODO: add check to the json file that the two attributes can not be null
291.             cas = new ContextAttributeState(o2.getJSONObject("object").getString("objectName"),
292.                   o2.getJSONObject("object").getString("objectURI"));
293.
294.             Iterator it2 = ca.getContextAttributeStatesList()!=null?
295.                   ca.getContextAttributeStatesList().iterator():null;
296.             boolean b2 = false; // new object state
297.             ContextAttributeState castmp = new ContextAttributeState();
298.             while(it2!=null&&it2.hasNext()){
299.                castmp = (ContextAttributeState)it2.next();
300.                if(castmp.getStateURI().equals(cas.getStateURI())){
301.                   b2=true; // already has the object state
302.                }
303.             }
304.
305.             if(!b2){
306.                //System.out.println("this is the place 1");
307.                ca.addContextAttributeStateToList(cas);
308.             }
309.             //cotmp3.addContextAttribute(ca);
310.             //the context attribute state machine will be very new too
311.             casm = cotmp3.getCasml().get(it4int);
312.             //casm.setCa(ca);
313.
314.             int col = 0;//used to note the incoming state and for the parameter of matrix
315.
316.             //handle hmstates (to store the map of the name and the number of states)
317.             //handle matrixes (to store the value of the casm)
318.             if(casm.getHmstates().containsKey(cas.getStateURI())){ //if the state has already been recorded, then edit ma
    trix
319.                col = casm.getHmstates().get(cas.getStateURI());
```

```
320.                 //System.out.println("row:" + casm.getCurrentState());
321.                 //System.out.println("col:" + col);
322.                 casm.getMatrixes().updateHmStates(casm.getCurrentState(), col);
323.                 casm.getMatrixes().updateDecisionMatrixStates(casm.getCurrentState(), col, decision); //row, column, dec
    ision
324.                 //TODO: important: from current state to a new state, actions can be triggered here
325.                 // and the actions can be stored in a decision form in decision matrix
326.                 casm.setCurrentState(col);
327.             } else{
328.                 int size = casm.getHmstates().size();
329.                 casm.addtoHashMapStates(size, cas.getStateURI());
330.                 casm.getMatrixes().addState(casm.getCurrentState()); // use current state to find the original state
331.                 casm.getMatrixes().addStateForDeicision(casm.getCurrentState(),decision);
332.                 casm.setCurrentState(size);
333.             }
334.             //cotmp3.addAttributeStateMachine(casm);
335.         }
336.     }
337.     //handle context attribute list
338.
339.   }
340.
341.   return cd;
342. }
```