國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science & Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

自動化事件腦力激盪：從 EARS 需求推導出受限制的前後文內容

Automate Event Storming Process to Derive Bounded Contexts from EARS Requirements

朱紹瑜

Shao-Yu Chu

指導教授: 李允中 博士

Advisor: Jonathan Lee, Ph.D.

中華民國 113 年 8 月

August, 2024

doi:10.6342/NTU202404153

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

自動化事件腦力激盪：從 EARS 需求推導出受限制的前後文內容

Automate Event Storming Process to Derive Bounded Contexts from EARS Requirements

本論文係朱紹瑜君（學號 R11922047）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 7 月 29 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 29 July 2024 have examined a Master's thesis entitled above presented by CHU, SHAO-YU (student ID: R11922047) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

李允中 (指導教授 Advisor)　　鄭卜壬　　蘇木春

王小捷　　簡宏宇

系主任/所長 Director: 陳祝嵩

Acknowledgements

首先，我要感謝李允中教授這兩年來的悉心指導，教導我運用系統性的方法分析問題、尋找解決方案，並在我做研究的過程中給予指引。接著，我要感謝臺灣大學軟體工程實驗室的同學：許容綮、蔡智冠、鍾昀諺、梁乃勻、王堃宇，在研究過程中互相討論與交流。此外，我也要感謝學妹鄭佳渝，在研究上給予我許多協助。同時，也感謝其他實驗室的學長姐、學弟妹、助理，以及家人的支持與幫助，讓我順利完成此篇論文。

doi:10.6342/NTU202404153

摘要

近年來，Event Storming 方法在軟體設計領域逐漸受到關注。然而，Event Storming 仰賴領域專家長時間參與，實施條件嚴苛。為了解決這個問題，我們提出了一種自動化的 Event Storming 流程，可用於從軟體需求中推導出 bounded contexts。該方法先從使用案例規格中提取 domain events，接著運用自然語言處理技術及外部知識庫的領域專業知識，找出領域模型中重要的 entities 及 attributes，進而依循 Domain-Driven Design 的概念將其聚合為 aggregates 和 bounded contexts。該方法有效減少了對領域專家參與的依賴，提高軟體設計的效率。

關鍵字：事件腦力激盪、領域驅動設計、軟體設計、自然語言處理、知識庫

doi:10.6342/NTU202404153

Abstract

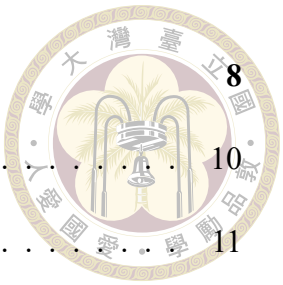
Event Storming has gained attention as an innovative method for software design. However, conducting Event Storming is challenging due to the need for extensive involvement from domain experts and stakeholders. In response to this challenge, we propose an automated Event Storming process aimed at deriving bounded contexts from software requirements. This process begins by extracting domain events from use case specifications. It then identifies the entities and attributes in the domain model by leveraging Natural Language Processing techniques and integrating domain knowledge sourced from external knowledge bases. These entities are subsequently aggregated into aggregates and bounded contexts in accordance with Domain-Driven Design principles. This method effectively reduces the dependency on domain experts' involvement and enhances the efficiency of software design.

Keywords: Event Storming, Domain-Driven Design, Software Design, Natural Language Processing, Knowledge Base

doi:10.6342/NTU202404153

Contents

doi:10.6342/NTU202404153



doi:10.6342/NTU202404153

doi:10.6342/NTU202404153

List of Figures

doi:10.6342/NTU202404153

doi:10.6342/NTU202404153

List of Tables

doi:10.6342/NTU202404153

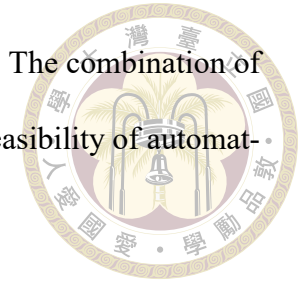
Chapter 1 Introduction

During the software design process, developers translate software requirements into comprehensive design representations, ensuring that each requirement is fully understood and fulfilled. This process is crucial within the software development lifecycle and significantly impacts the success of the software project. [6] Among the various software design approaches, Event Storming [4], introduced by Alberto Brandolini, is a novel strategy based on Domain-Driven Design (DDD) [8]. Through collaborative interactions between domain experts and developers in Event Storming workshops, they clarify business processes, define subdomain boundaries, and construct domain models.

However, Event Storming is time-consuming and expensive. It demands domain experts, developers, and stakeholders to participate in workshops that can span days. Moreover, software requirements are constantly evolving. It is impractical to bring the groups together for Event Storming workshops every time the requirements change. Therefore, there is a need to identify a more efficient approach to address requirement changes and ensure the agility of software design processes.

In typical software development lifecycles, software requirements are documented in natural language in the requirements analysis phase. These requirements clearly define business processes. Additionally, off-the-shelf knowledge bases contain comprehensive

doi:10.6342/NTU202404153



records of general human knowledge and domain-specific expertise. The combination of software requirements and external knowledge bases enhances the feasibility of automating the Event Storming process.

In previous work, Lin [17] leveraged the fundamental steps of Event Storming to gather domain events and subsequently derive bounded contexts. Yet, the approach still faces the following issues:

1. Although Lin's method has semi-automated Event Storming, it still requires human intervention when distinguishing entities and attributes.
2. The approach focuses on distinguishing entities and attributes. It does not differentiate between entities, aggregates, and bounded contexts.

To address these issues, we propose the following methods:

1. We analyze the semantic and syntactic information in domain events and utilize external knowledge bases to access domain knowledge, a method adopted from Lin's approach. This combined approach allows us to automatically distinguish between entities and attributes.
2. We aggregate entities into aggregates and identify bounded contexts by adhering to the definitions of these model elements in Domain-Driven Design.

This paper is organized as follows: Chapter 2 introduces the background and related works; Chapter 3 details the process of deriving bounded contexts from software requirements; Chapter 4 presents a case study on automatically deriving bounded contexts from

doi:10.6342/NTU202404153

the requirements of the Unified University Inventory System (UUIS) [2]; Chapter 5 summarizes the contributions of this work; and, finally, Chapter 6 outlines future research directions.

doi:10.6342/NTU202404153

Chapter 2 Related Work

2.1 Background Work

2.1.1 Domain-Driven Design

Domain-Driven Design (DDD) is a software design concept introduced by Eric Evans in his book *Domain-Driven Design: Tackling Complexity in the Heart of Software* [8]. It emphasizes that the design of software systems should align closely with the business logic of their respective domains. This philosophy enhances the maintainability of software systems.

The book introduces the principles of DDD and defines a vocabulary for domain modeling. The terms that will be used in this paper include:

Domain DDD defines domain as "A sphere of knowledge, influence, or activity".

Bounded Context Large-scale software projects often encompass extensive business domains, requiring collaboration among multiple teams. It is important that each team focuses on specific areas of the domain model, maintains consistency within their respective scopes, and understands their relationships with other teams. Thus, clear boundaries need to be defined. Consequently, bounded contexts are defined as "the

doi:10.6342/NTU202404153

delimited applicability of a particular model."

Aggregate An aggregate is a cluster of associated objects. Each aggregate has an entity serving as its aggregate root. The aggregate root is the only entry point to the aggregate. Any changes to the data within the aggregate must go through the aggregate root to ensure consistency.

Entity An entity is a concrete concept that encompasses essential business logic. Entity objects are identified by their identities rather than the values of their attributes.

2.1.2 Event Storming

Event Storming [4] is a software design technique proposed by Alberto Brandolini. It aims to clarify business processes and establish domain models through highly collaborative workshops. This approach is based on the principles of Domain-Driven Design and facilitates the joint design of complex software systems by domain experts and developers through the visualization of domain events and business processes.

In this work, we adopt the four fundamental steps of Event Storming:

1. Identify domain events arrange them chronologically.
2. Identify the commands and actors that trigger the domain events.
3. Group domain events and commands into aggregates.
4. Group aggregates into bounded contexts.

doi:10.6342/NTU202404153

2.1.3 EARS Requirements

Software requirements are often written in unstructured natural language, leading to issues such as ambiguity, complexity, omission, and duplication. The Easy Approach to Requirements Syntax (EARS) [18] addresses these issues by providing a structured requirement syntax. The six types of requirement templates are illustrated in Table 2.1.

Table 2.1: EARS Requirement Syntax

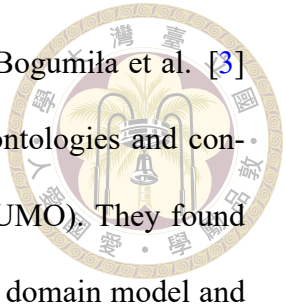
Requirements Type	Template
Ubiquitous Requirements	The <system name> shall <system response>.
Event-Driven Requirements	WHEN <optional preconditions> <trigger>, the <system name> shall <system response>.
Unwanted Requirements	IF <optional preconditions> <trigger>, THEN the <system name> shall <system response>.
State-Driven Requirements	WHILE <in a specific state>, the <system name> shall <system response>.
Optional Features Requirements	WHERE <feature is included>, the <system name> shall <system response>.
Compound Requirements	<Multiple conditions>, the <system name> SHALL <system response>.

2.2 Related Work

2.2.1 Leveraging Domain Knowledge in Domain Modeling

In software engineering, domain knowledge refers to the understanding of the environment in which the target system operates. Previous research indicates that domain knowledge is crucial for domain modeling. Cherfi et al. [23] proposed a method to enhance the quality of business process (BP) models by leveraging domain knowledge. The method aligns BP models with domain ontologies to identify defects in existing models

doi:10.6342/NTU202404153



and recommends improvements to enhance readability and quality. Bogumiła et al. [3] introduced a method for developing domain models using domain ontologies and conducted a case study with the Suggested Upper Merged Ontology (SUMO). They found that this approach helps maintain consistency between the developed domain model and the domain ontology.

2.2.2 Mapping Natural Language to Programming Language

2.2.2.1 CodeBERT

Feng et al. [10] proposed CodeBERT, a bimodal pre-trained model designed for comprehending both natural language (NL) and programming language (PL). The model adopts a Transformer-based architecture and is pre-trained on bimodal data comprising NL-PL pairs and unimodal PL data. Empirical evaluations have shown that, after fine-tuning, the pre-trained model can be applied to tasks such as natural language code search and code documentation generation.

2.2.2.2 CodeGPT

Studies have shown the capabilities of large language models (LLMs) in generating code from natural language descriptions [1, 5, 15, 28]. This capability enables LLMs to assist engineers in both software design and implementation. CodeGPT [14] exemplifies a code assistant based on LLMs. It enhances developers' efficiency by integrating LLMs into integrated development environments (IDE), which allows developers to interact with the models, such as asking them to explain code or generate code based on requirements written in natural language.

doi:10.6342/NTU202404153

Chapter 3 The Process of Deriving Bounded Contexts

This section provides a detailed description of the automated process for deriving bounded contexts. The inputs and outputs of this process are as follows:

Input Requirements and Use Case Specification To ensure that the input is clear, precise, and complete, we recommend documenting the requirements using the Easy Approach to Requirements Syntax (EARS). For the use case specification, it should include the ID, actors, pre-conditions, post-conditions, basic flows, alternative flows, and exceptional flows for each use case.

Output Bounded Contexts The output of the process consists of the bounded contexts derived from the input requirements and use case specifications. Each bounded context includes one or more aggregates, with each aggregate comprising one or more entities and their associated attributes.

The overview of the process is shown in Figure 3.1. It consists of four major steps:

1. Collect Domain Events

   Following the principle of Event Storming, this step involves collecting domain

doi:10.6342/NTU202404153

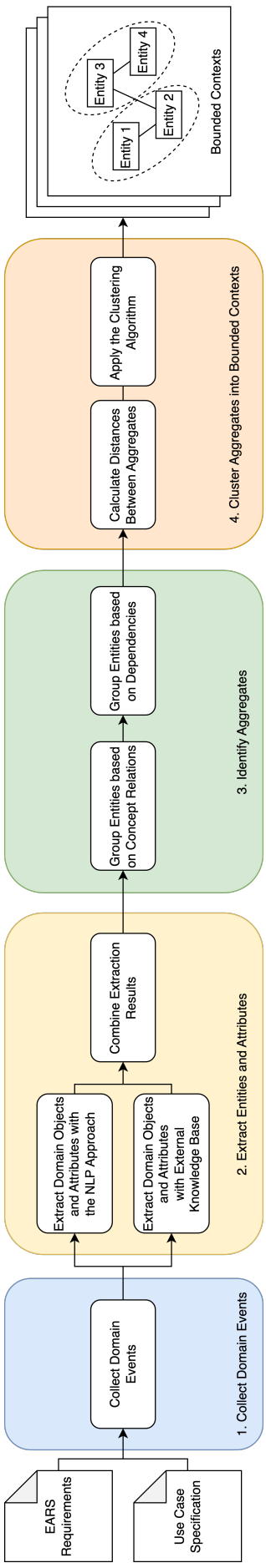


Figure 3.1: System Overview

doi:10.6342/NTU202404153

events from the requirements and the use case specification.

2. Extract Entities and Attributes

   This step involves extracting entities and attributes from the collected domain events by analyzing syntactic and semantic information as well as external knowledge bases.

3. Identify Aggregates

   This step further groups the extracted entities into aggregates and determines the root of each aggregate.

4. Cluster Aggregates into Bounded Contexts

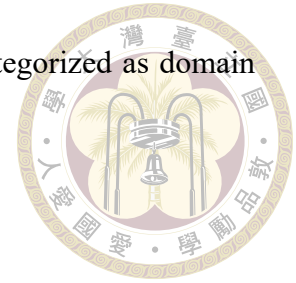
   This step calculates the distances between aggregates and clusters them into bounded contexts accordingly.

3.1 Collect Domain Events

In the beginning of an Event Storming session, all participants collaborate to "storm out" the domain events to gain a clear picture of the business process. These domain events are typically written in the past tense and arranged chronologically.

The descriptions in use case specifications naturally document the domain events, their corresponding commands, and the involved actors, arranging them in chronological order. For example, pre-conditions must precede the basic flows, while actions in basic flows, alternative flows, and exceptional flows must occur in the order of their serial numbers. Consequently, we extract domain events, commands, and actors from the use case specifications. Specifically, pre-conditions, basic flows, alternative flows, exceptional

doi:10.6342/NTU202404153



flows, and post-conditions from each use case are collected and categorized as domain events.

3.2 Extract Entities and Attributes

After determining the domain events and commands, participants in Event Storming sessions identify aggregates based on the responsibilities associated with each domain event and command. According to Domain-Driven Design, an aggregate consists of multiple entities. Entities are objects specific to the business area that represent concepts meaningful to domain experts. In this phase, we employ two approaches: the NLP-Based Approach (Section 3.2.1) and the Knowledge Graph-Based Approach (Section 3.2.2) to identify domain objects and their attributes. We then combine the results to derive entities (Section 3.2.3).

3.2.1 NLP-Based Approach

The natural language domain events collected from use case specifications contain substantial domain-specific information. For instance, the sentence “the user enters their username and password and presses the Login Button” implies that the system must track users’ usernames and passwords. In the NLP-Based Approach, we extract domain objects and attributes by analyzing the underlying information within these natural language statements.

doi:10.6342/NTU202404153

3.2.1.1 Syntactic Dependency Parsing

For syntactic analysis, we use the `DependencyParser` from the spaCy [12] `en_core_web_trf` model to perform syntactic dependency parsing. As illustrated in Figure 3.2, the parser segments sentences into multiple tokens. Each token generally corresponds to a word in the sentence and is associated with a head and a dependency label, except for the `ROOT` token. The dependencies among the tokens form a tree structure.

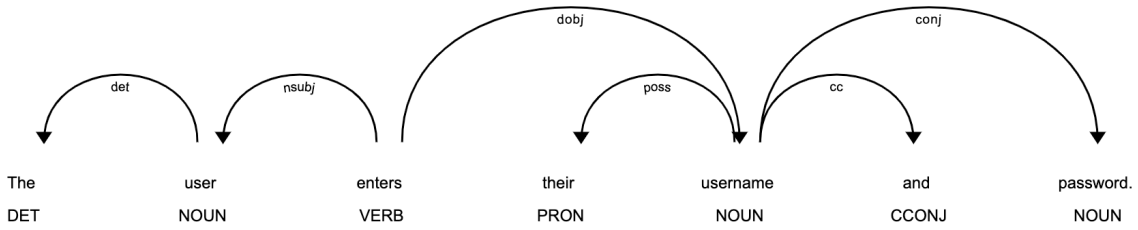


Figure 3.2: An example of the syntactic dependency parsing result. The token *enters* serves as the `ROOT` of this sentence. The token *user* depends on the token *enters*, with the dependency label `nsubj`.

3.2.1.2 Semantic Analysis

For semantic analysis, we apply the following techniques and tools:

Word Sense Disambiguation (WSD) WSD is a natural language processing technique used to determine the meaning of a word within a specific context. In this work, we employ PyWSD [25] to perform WSD.

Coreference Resolution To avoid repetition, pronouns are commonly used in natural language to refer to previously mentioned entities. However, this can make it challenging for machines to understand the actual referents. Therefore, natural language processing often employs Coreference Resolution techniques to identify words or

doi:10.6342/NTU202404153

phrases that refer to the same entity. In this work, we use the `CoreferenceResolver` from spacy-experimental [9] for Coreference Resolution.

Semantic Categorization To classify the semantics of words, we use the semantic categories provided by WordNet [20]. The WordNet database organizes all synonym sets (synsets) into 45 categories, which include 26 noun categories, 15 verb categories, 3 adjective categories, and 1 adverb category.

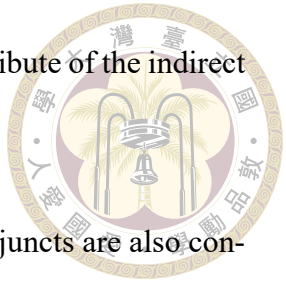
Sentiment Analysis Domain events are typically expressed in statements, which can be either positive or negative. To identify negative statements, we use regular expression pattern matching and employ the Valence Aware Dictionary and sEntiment Reasoner (VADER) [13] to perform sentiment analysis.

3.2.1.3 Rules

Combining the syntactic and semantic information in natural language, we compile the following nine rules for identifying domain objects and attributes. The process overview of the NLP-Based Approach is shown in Figure 3.3.

1. Proper nouns are domain objects.
2. The subject of a sentence, where the verb is an action verb (see Appendix A.1), is a domain object.
3. In a sentence with a direct object, if the verb of the sentence falls into the category of change-inducing verbs (see Appendix A.2), the subject is a domain object, and the direct object serves as attributes of the subject.
4. If a verb has both an indirect object and a direct object, the indirect object is consid-

doi:10.6342/NTU202404153



ered a domain object, while the direct object is viewed as an attribute of the indirect object.

5. If a noun is identified as a domain object or an attribute, its conjuncts are also considered domain objects or attributes of the same domain object, respectively.

6. If a noun phrase indicates possession, the possessor noun is a domain object, and the possessed noun is an attribute of the domain object.

7. If there is a phrase in the form of "X of Y",

   (a) If it matches the pattern "the X of Y is Z",

      i. If Z is a proper noun, Y and Z should be domain objects linked by the relationship X.

      ii. Otherwise, X should be an attribute of the domain object Y.

   (b) If X is a collective noun, view the phrase as Y and reanalyze the sentence.

   (c) Otherwise, Y should be a domain object, and X should be an attribute of Y.

8. If a noun has an appositive, use the appositive as the name of the domain object or the attribute and ignore the main noun.

9. If a sentence is a negative statement, do not use it to identify domain objects and attributes.

3.2.2 Knowledge Graph-Based Approach

Domain experts play a critical role in Event Storming. Their expertise in the domain ensures that teams can gain a comprehensive understanding of the business domain. The

doi:10.6342/NTU202404153

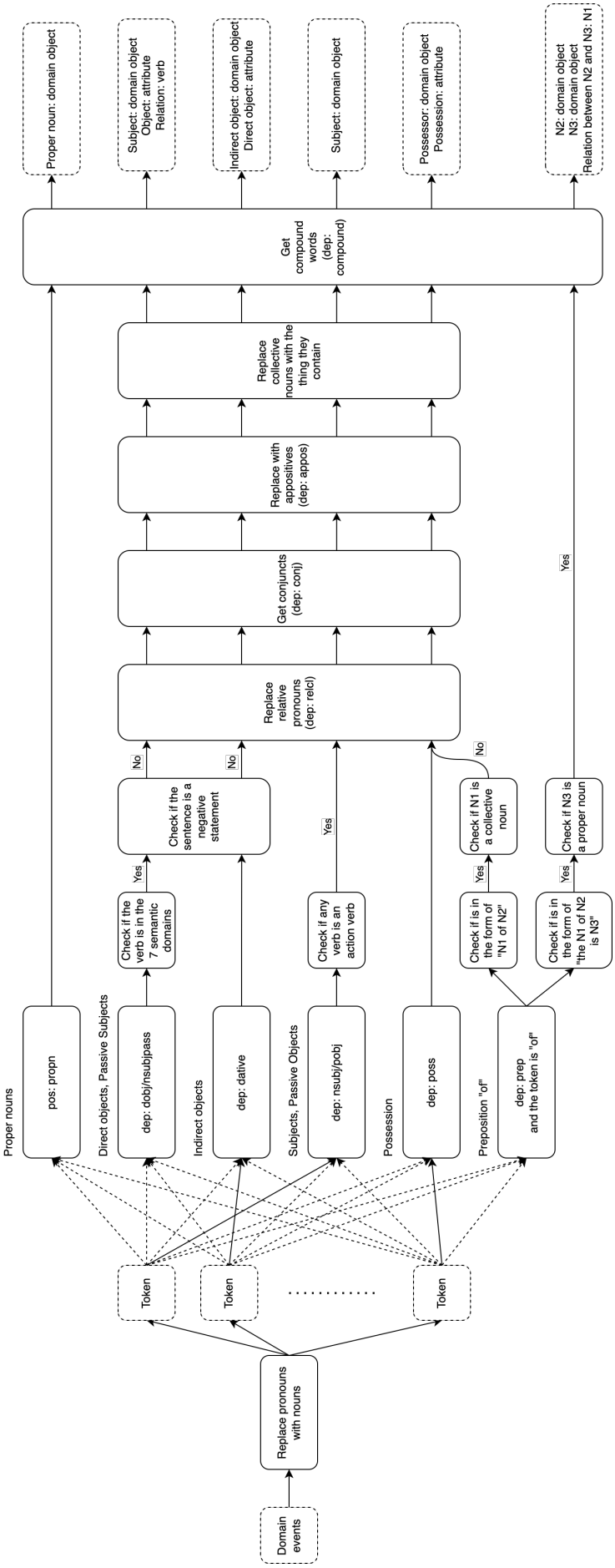


Figure 3.3: The process of the NLP-based approach.

doi:10.6342/NTU202404153

development and prevalence of knowledge bases document domain-specific knowledge. Leveraging these knowledge bases can significantly reduce teams' dependence on domain experts.

In this work, we leverage ConceptNet [24] as the source of domain knowledge for automated Event Storming. ConceptNet is a public, multilingual semantic network designed to assist machines in understanding human language. The network represents words and phrases as nodes, referred to as Concepts, with edges connecting these Concepts to describe their relationships (see Appendix A.2). For example, the edge $(computer, IsA, electronic\_device)$ indicates that a computer is an electronic device. We focus on the Concepts referring to English terms and the edges between them, of which there are approximately 1,500,000.

We identify domain objects and attributes with ConceptNet through the following three steps:

1. Group use cases based on their relationships.
2. Identify the domain objects and attributes within each use case group.
3. Determine which domain object each attribute is associated with.

3.2.2.1 Group Use Cases Based on Use Case Relationships

The business logic of related use cases often surround same sets of domain objects and attributes. Therefore, before identifying domain objects and attributes, we group use cases based on their relationships.

There are three types of use case relationships: include, extend, and generalization.

doi:10.6342/NTU202404153

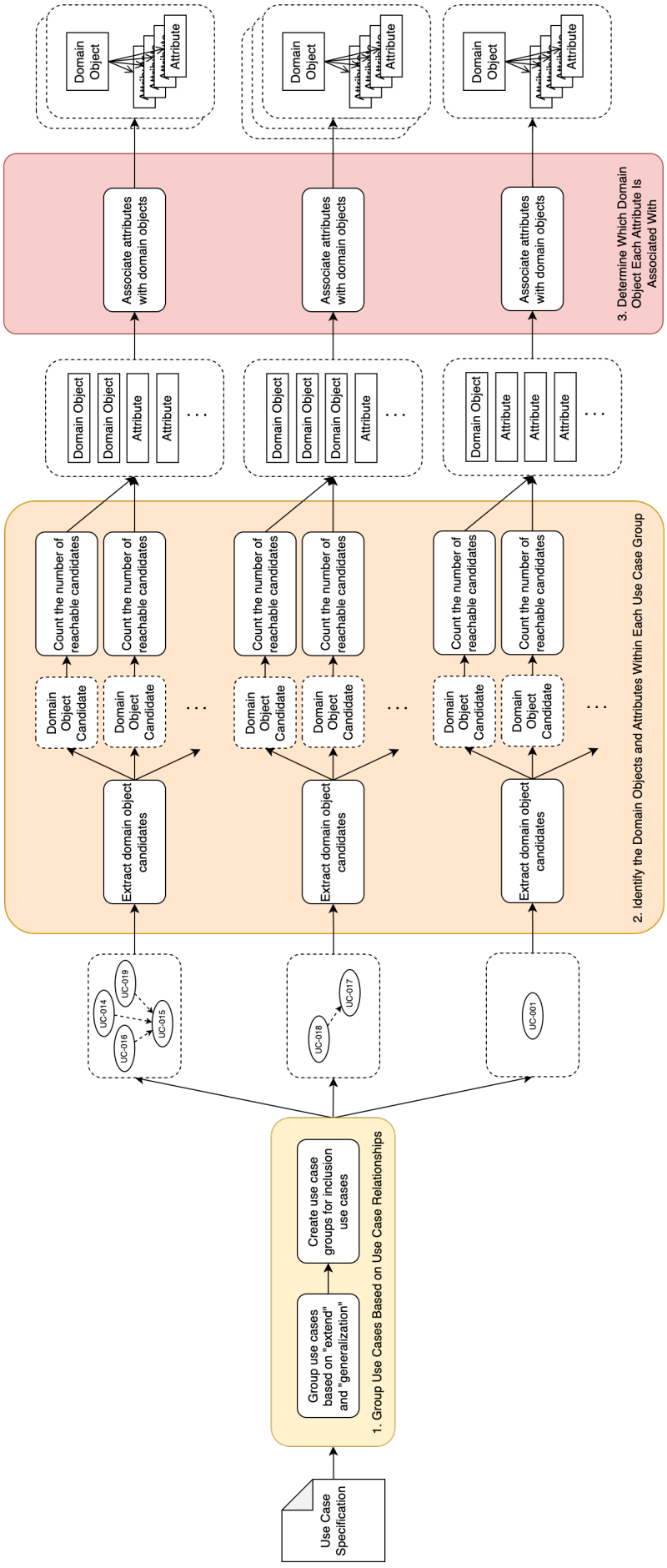
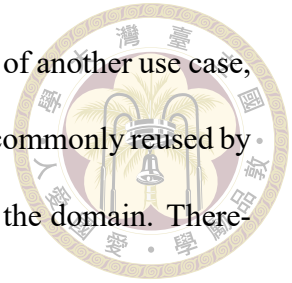


Figure 3.4: The process of the knowledge graph-based approach.

doi:10.6342/NTU202404153



Include For use case inclusion, a base use case includes the behavior of another use case, the included use case. Included use cases represent logic that is commonly reused by other use cases. They must contain domain objects relevant to the domain. Therefore, each included use case forms an individual group.

Extend For use case extension, the extension use case adds optional behaviors to the base use case. Both use cases should be in the same theme, so the two use cases should belong to the same group.

Generalization For use case generalization, the parent use case presents a goal, and all child use cases describe specific ways to achieve the same goal. Similarly, both use cases should be in the same theme, so the two use cases should belong to the same group.

Based on the above analysis, we first group the use cases according to their extend and generalization relationships. Then, we add each included use case as a new group containing only itself. Figure 3.5 shows an example of the grouping result. The steps for this stage are as follows:

1. Group use cases based on use case extension and generalization. Consider the relationship between use cases as a directed graph. The nodes represent the use cases, and there is an edge between two nodes if there is an “extend” or “generalization” relationship between the two use cases.

2. Construct groups based on use case inclusion. If an included use case is categorized as a use case group containing multiple use cases in the previous step, add a new group containing only the included use case itself.

doi:10.6342/NTU202404153

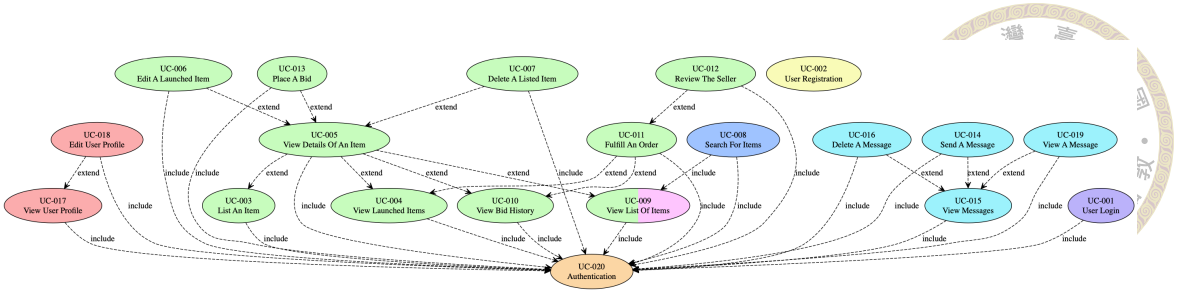


Figure 3.5: An example of the use case grouping result. Each color corresponds to a use case group. In this example, there are eight use case groups. Note that UC-009 appears in two groups: one group contains only UC-009 itself, while the other group includes UC-003, UC-004, UC-005, UC-006, UC-007, UC-009, UC-010, UC-011, UC-012, and UC-013.

3.2.2.2 Identify the Domain Objects and Attributes Within Each Use Case Group

In this step, we identify domain objects and attributes from the domain events of each use case group.

We begin by extracting nouns from the domain events as domain object candidates. Part-of-Speech (POS) tagging is performed using the `Tagger` from the spaCy [12] `en_core_web_lg` model. Words identified as `NOUN` or `PROPN` (Proper Noun) are selected as domain object candidates.

Next, we use ConceptNet [24] to distinguish domain object candidates into domain objects and attributes. For each use case group, we determine the number of candidates that each candidate can reach within two hops in ConceptNet. The domain object candidate(s) that can reach the highest number of candidates in each use case group are identified as the domain object(s), while the rest are categorized as attributes.

3.2.2.3 Determine Which Domain Object Each Attribute Is Associated With

Finally, we determine the domain object(s) associated with each attribute. For each attribute, if a domain object can reach it within two hops in ConceptNet, the domain object

doi:10.6342/NTU202404153

is linked to that attribute. If an attribute cannot be reached by any domain object, we consider it a domain object.

3.2.3 Combining the Extraction Results

Following the NLP-Based Approach and the Knowledge Graph-Based Approach, we combine the extraction results of the two approaches. The combination follows the rules below:

- If a noun is identified as a domain object in one approach and an attribute in the other, it should be transformed into a domain object.
- If a domain object does not have any attributes, transform it into an attribute.
- Discard attributes that are not referenced by any others.

After the combination, the resulting domain objects correspond to entities, while attributes correspond to value objects or attributes of entities.

3.3 Identify Aggregates

According to the definition in Domain-Driven Design, an aggregate is "a cluster of associated objects that are treated as a unit for the purpose of data changes." Each aggregate must have an entity that serves as the aggregate root. The aggregate root is the only entry point of the aggregate and is the only member of the aggregate that objects outside the aggregate are allowed to reference.

doi:10.6342/NTU202404153

Based on the definition, we cluster entities into aggregates based on the following properties:

- Entities connected with the "aggregation" and "generalization" relationships should belong to the same aggregate.
- The aggregate root should be the only member of the aggregate that objects outside the aggregate are allowed to hold a reference to.

The process of grouping entities into aggregates is as follows:

1. Let each entity be an aggregate containing itself. Let the only entity within each aggregate be the aggregate root.
2. If any two entities have relation "PartOf", "MadeOf", "HasProperty", or "IsA" in ConceptNet, merge the two corresponding aggregates. Let the aggregate root of the source of the relationship be the new aggregate root.
3. If any entity holds a reference to an entity from another aggregate that is not an aggregate root, merge the two corresponding aggregates into one. Let the root of the aggregate containing the referenced entity become the new aggregate root. Repeat this step iteratively until no entity holds any non-aggregate-root entity.

3.4 Cluster Aggregates into Bounded Contexts

Large-scale software projects typically require collaboration among multiple teams to be successfully developed. To ensure that each team can focus on specific parts of the system and maintain consistency within their respective scopes, Domain-Driven Design

doi:10.6342/NTU202404153

[8] introduced the concept of bounded contexts. It defines bounded contexts as "The delimited applicability of a particular model. Bounded contexts give team members a clear and shared understanding of what has to be consistent and what can develop independently."

A bounded context consists of one or more aggregates. Therefore, we further cluster the aggregates identified in Section 3.3 into bounded contexts. The process is illustrated in Figure 3.6.

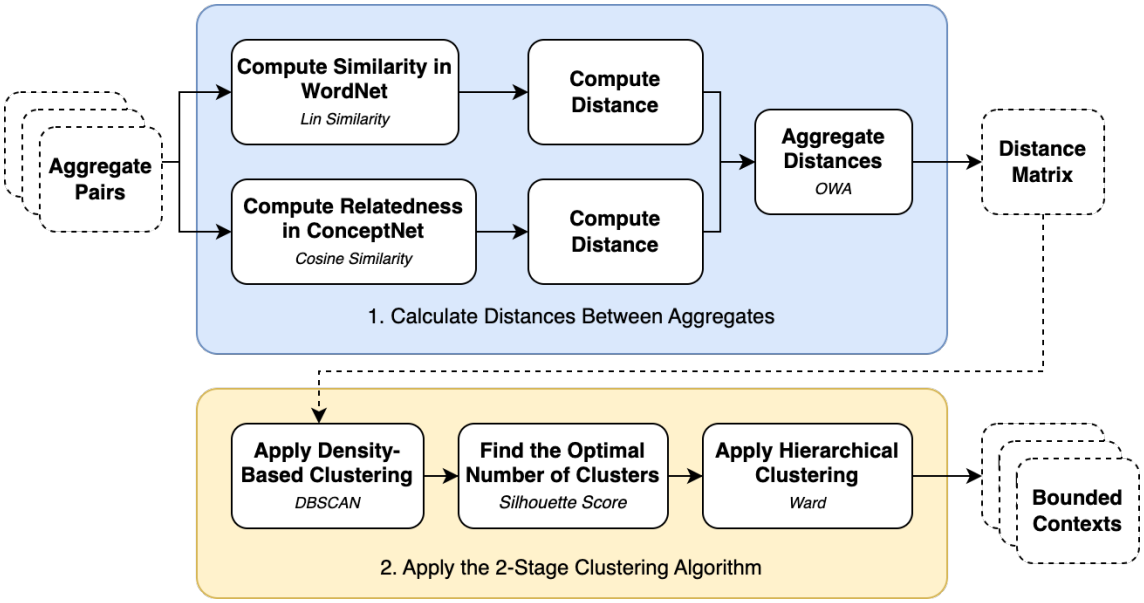
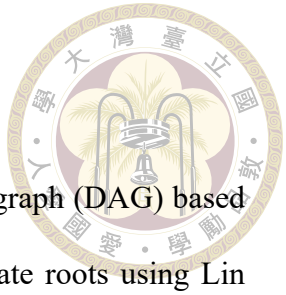


Figure 3.6: The process of clustering aggregates into bounded contexts.

3.4.1 Calculate Distances Between Aggregates

To cluster aggregates into bounded contexts, we use WordNet and ConceptNet to calculate the distances between aggregates based on their similarities.

doi:10.6342/NTU202404153



3.4.1.1 Similarity in WordNet

In WordNet, noun synsets are organized into a directed acyclic graph (DAG) based on Is-A relationships. We compute the similarities between aggregate roots using Lin Similarity [16], which bases on the DAG and information content (IC).

For calculating information content, we use the Brown Corpus [11], a large American English corpus comprising approximately 1,000,000 words from various themes and genres. Based on the Brown Corpus, the IC of a synset s is computed as follows:

$$IC(s) = -\log P(s) = -\log \frac{freq(s)}{N} = -\log \frac{\sum_{n \in words(s)} count(n)}{N}, \tag{3.1}$$

where $words(s)$ represents the set of words with the synset s and all its hyponyms; $freq(s)$ denotes the frequency of synset s, calculated as the total occurrence count of all words corresponding to synset s and its hyponyms in the corpus; N is the total number of noun occurrences in the corpus.

Based on Equation 3.1, the Lin Similarity of two aggregates is computed as follows:

$$sim_W(s_i, s_j) = \frac{2 \cdot IC(LCS_{s_i,s_j})}{IC(s_i) + IC(s_j)}, \tag{3.2}$$

where s_i and s_j represent the synsets of the two aggregate roots, respectively, and $\text{LCS}(s_i, s_j)$ is the Least Common Subsumer (LCS) of the two synsets in the DAG formed by Is-A relationships. The similarity ranges from 0 to 1, reaching its maximum when $s_i = s_j$.

doi:10.6342/NTU202404153

3.4.1.2 Relatedness in ConceptNet

To measure similarities between aggregates with ConceptNet, we utilize its Relatedness API. The API calculates the relatedness between two Concepts based on the Cosine similarity of their word embeddings:

$$sim_C(c_i, c_j) = \frac{emb_{c_i} \cdot emb_{c_j}}{\|emb_{c_i}\| \|emb_{c_j}\|}, \tag{3.3}$$

where c_i and c_j represent the Concepts corresponding to the two aggregate roots; emb_{c_i} and emb_{c_j} denote the two corresponding word embedding vectors, which are derived from a pruned ConceptNet grph, word2vec [19], and GloVe [21]. The similarity ranges from -1 to 1, reaching its maximum when $c_i = c_j$.

3.4.1.3 Aggregating the Metrics

First, we convert the similarities obtained from Sections 3.4.1.1 and 3.4.1.2 into distance values ranging between 0 and 1. The higher the similarity between aggregates, the closer their distance should be. Therefore, assuming the aggregate roots of two aggregates are r_i and r_j, corresponding to synsets s_{r_i} and s_{r_j}, the distance value is calculated as follows:

$$dis_W(r_i, r_j) = 1 - sim_W(s_{r_i}, s_{r_j})$$
$$dis_C(r_i, r_j) = -\frac{1}{2} \cdot sim_C(r_i, r_j) + \frac{1}{2}.$$

Next, we aggregate the two distance values with the ordered weighted averaging

doi:10.6342/NTU202404153

(OWA) operator [27]:

$$dis_{\text{OWA}}(r_i, r_j) = \sum_{k=1}^{2} w_k \cdot dis_k(r_i, r_j)\,,$$

where $dis_k(r_i, r_j)$ is the k^{th} largest distance values among $[dis_W(r_i, r_j), dis_C(r_i, r_j)]$. Here, we select the weights as $[w_1, w_2] = [0.8, 0.2]$.

3.4.2 Clustering Algorithm

In this stage, we employ a two-stage clustering algorithm to group aggregates into bounded contexts based on the distances computed in Section 3.4.1. The algorithm integrates a density-based method with a hierarchical clustering method and dynamically selects the optimal number of clusters.

3.4.2.1 Density-Based Clustering

First, we adopt the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [7] algorithm for the initial classification of all aggregates. This step allows us to effectively identify and remove outliers and preliminarily classify the non-outlier aggregates. Although the DBSCAN algorithm does not require the number of clusters to be predetermined, its performance heavily depends on the hyperparameters $MinPts$ and Eps. It is challenging to select an appropriate Eps for unknown data distributions. Therefore, in this stage, we only perform a preliminary classification of aggregates and dynamically adjust the number of clusters in subsequent steps.

doi:10.6342/NTU202404153

3.4.2.2 Find the Optimal Number of Clusters

Next, we further divide each cluster obtained from the DBSCAN algorithm into multiple subclusters. To dynamically select the optimal number of subclusters for each cluster, we use the Silhouette score [22] as the criterion for evaluating the quality of the clustering results. Since the goal of subclustering is to group similar data points together, we aim for data points within the same subcluster to be as close to each other as possible, and data points in different subclusters to be as far apart as possible. The Silhouette score measures such property.

To compute the Silhouette score of a subclustering result, we first calculate the Silhouette value of each data point, i.e., each aggregate, as follows:

1. Calculate the average distance of the data point to the other data points within the same subcluster.
$$a(i) = \frac{\sum_{j \in C_I, i \neq j} dis(i,j)}{|C_I| - 1}$$

2. Calculate the minimum average distance from the data point to the data points in other subclusters.
$$b(i) = \min_{J \neq I} \frac{\sum_{j \in C_J} dis(i,j)}{|C_J|}$$

3. Combine $a(i)$ and $b(i)$ into the Silhouette value $s(i)$.
$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max(a(i),b(i))} & , \text{ if } |C_I| > 1 \\ 0 & , \text{ otherwise}, \end{cases}$$

where $0 \leq s(i) \leq 1$. The Silhouette score is then computed as the average Silhouette value of all data points. A higher Silhouette score indicates a more optimal subclustering

doi:10.6342/NTU202404153

result.

We select the number of subclusters with the highest Silhouette score as the optimal number of subclusters.

3.4.2.3 Hierarchical Clustering

Finally, based on Sections 3.4.2.1 and 3.4.2.2, we divide the preliminary clusters into subclusters. We use Hierarchical Agglomerative Clustering (HAC) to split each cluster generated by DBSCAN into the optimal number of subclusters. Initially, each aggregate forms an individual subcluster. Subsequently, subclusters are merged recursively until the number of subclusters reaches the optimal count. During this process, merging is performed using Ward's method [26], which minimizes the sum of squared Euclidean distances from data points to the subcluster center.

After this final step, the aggregates within each subcluster form a bounded context.

doi:10.6342/NTU202404153

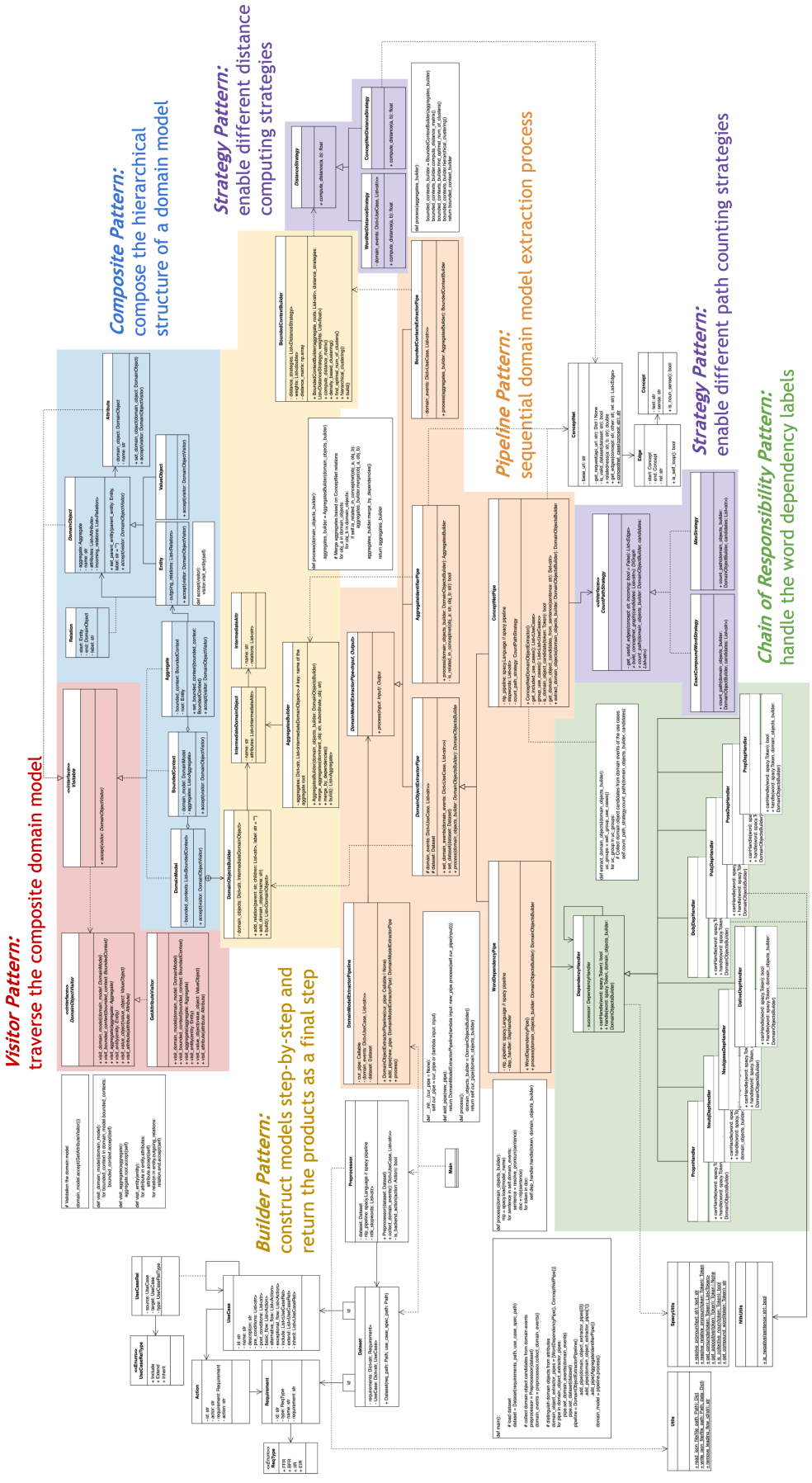


Figure 3.7: Class Diagram Design

doi:10.6342/NTU202404153

Chapter 4 Case Study: Unified University Inventory System

In this chapter, we present a case study on automatically deriving bounded contexts from the requirements of the Unified University Inventory System (UUIS) [2]. The requirement describes a university asset management system. The system integrates the databases of three departments and supports functions such as asset transfer, editing, modification, addition, borrowing requests, asset returning, location creation, request approval, authentication, and permission changes.

4.1 Input

The automated bounded contexts derivation process takes software requirements and use case specifications as its input. The UUIS includes 25 functional requirements across 11 categories (see Table 4.1) and 10 use cases (see Table 4.2).

doi:10.6342/NTU202404153

Table 4.1: Functional Requirements of the Unified University Inventory System

ID	Requirement Description
1	**Transferring Assets**
1.1	Within the same department: data base can be updated directly without any request.
1.2	Inter departments: request must be approved by a DA group member and faculty group member unless it came from a higher level group.
1.3	Inter faculties transfer: request can be made by any authorized user and approved by faculty group or higher level.
1.4	Transfer outside university should be approved by the university group.
2	**Editing Assets**
2.1	Any administrative level user or inventory user can edit an asset that belongs to its department; same thing for faculty user, or university user; in order to make modification if he is authorized to do it.
3	**Modifying Assets**
3.1	All fields of an edited asset can be modified except Ids.
3.2	A bulk entry file can be used.
4	**Adding Inventory Assets**
4.1	Any DA group member or authorized inventory group member asset is owned by the department.
4.2	Any faculty member can add all related departments inventory.
4.3	Any university group member can add all assets in the inventory.
4.4	A bulk entry can be used to add many assets.
5	**Creating Request to Borrow an Asset or Reserve a Location**
5.1	Request can be made by any authorized user.
5.2	After creation, a request remains pending, waiting to be approved by an administrative-level user who has the authority to do so.
6	**Returning Assets**
6.1	An inventory user should check the returned asset and update the inventory.
7	**Creating a New Location**

doi:10.6342/NTU202404153

7.1	IT group members can create a new space and modify floor structure when they receive an exception request from any administrative level.
8	**Approving Requests**
8.1	Any administration level or authorized inventory group member can display all pending requests waiting for approval from this level and approve those requests.
8.2	When request is treated, the user is notified by email.
8.3	Request is added to the waiting for execution list.
8.4	Inventory is updated when user receive the requested asset.
9	**Authentication**
9.1	Authentication is made by user name and a password for all users.
9.2	Administrative level works on administration computers.
10	**Changing Permission**
10.1	Any administrative lavel user can delegate another user to execute some or all his authorized actions. And this user acquires the role of inventory administrator.
11	**Output Reports**
11.1	Asset report by location.
11.2	Request report.
11.3	User permission user.

Table 4.2: Use Case Specification of the Unified University Inventory System

ID	Name	Description
MOD.UC	Modify Use Case	The use case describes the modification that the Inventory Admin can do.
EDT.UC	Edit Use Case	The use case describes the edit operation that the Inventory Admin can do.
ANI.UC	Add New Asset Use Case	The use case describes the operation of adding a new asset to the inventory.
CRQ.UC	Create Request Use Case	The use case describes the activity of creating a new request that the User can do.

doi:10.6342/NTU202404153

RTI.UC	Returning Asset	The use case describes the returning asset update that the Inventory Admin can do.
APR.UC	Approving Request	The use case describes the approving of requests that an Inventory Admin can do.
CHP.UC	Change Permission Use Case	The use case describes the modification that Department, Faculty, or University Administrator can do to user permissions.
ATH.UC	Authentication Use Case	The use case describes the authentication.
SRCH.UC	Search	The use case describes the search operation that only authorized users can do.
REP.UC	Create Reports Use Case	The use case describes the creation of reports that the User can do.

4.2 Domain Events

First, the process extracts each use case's pre-conditions, basic flow, alternative flows, exceptional flows, and post-conditions as domain events.

Here, 123 domain events are extracted from the UUIS use case specification, with 76 being unique. For example, 19 domain events are extracted from the Edit Use Case, as shown in Table 4.3.

Table 4.3: Domain Events of the Edit Use Case

	Pre-Conditions
1	Inventory Admin is authenticated.
	Basic Flow
1	The use case begins when Inventory Admin start searching for an asset.
2	Inventory Admin edits the asset.
3	Inventory Admin modifies asset properties.

doi:10.6342/NTU202404153

	Alternative Flow
Cond	Administrator or authorized inventory user is waiting for approval list or waiting for execution list.
1	Inventory user or Admin edits the asset.
2	Inventory user or Admin modifies asset properties.
	Exceptional Flow
1.1	Inventory Admin searches for the asset.
1.2	Inventory Admin edits the asset.
1.3	Inventory Admin finds that the asset is out of inventory.
1.4	Message error is displayed because the asset cannot be modified.
2.1	Inventory Admin searches for the asset.
2.2	Inventory Admin edits the asset.
2.3	Inventory Admin does not have sufficient privileges to edit the asset.
2.4	Message error is displayed.
3.1	Inventory Admin searches for the asset.
3.2	No asset is found.
3.3	Message error is displayed.
	Post-Conditions
1	The system state change according to the modification.

4.3 Entities and Attributes

Next, we extract entities and attributes from the domain events with both the NLP-Based Approach and the Knowledge Graph-Based Approach.

The entities and attributes obtained through the NLP-Based Approach are shown in Figure 4.1. For example, the entity `USER` has attributes such as `username`, `password`, and `permission`, which align with the descriptions in the UUIS requirements.

doi:10.6342/NTU202404153

However, there are still some issues with the extraction results. For example, `INVENTORY ADMIN` and `ADMIN` refer to the same concept but are identified as two distinct entities. `ADMIN` originates from the domain events "Inventory user or Admin edits the asset" and "Inventory user or Admin modifies asset properties." In these sentences, the subject could be understood as "Inventory user and Admin" or "Inventory user and Inventory Admin." The latter aligns better with the requirements description, but the NLP-Based Approach identifies it as the former due to ambiguity.

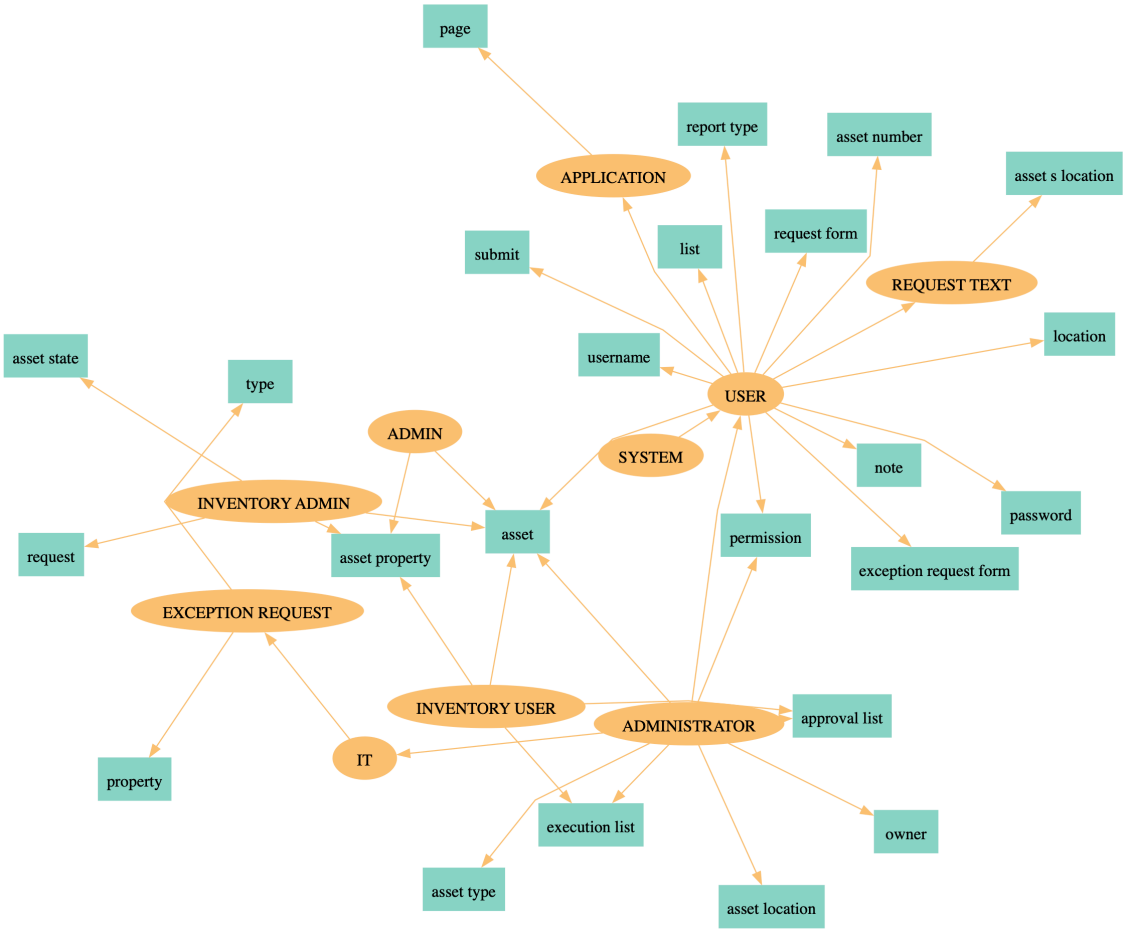


Figure 4.1: The result of the NLP-based entity/attribute extraction. The ellipses denote the entities, while the rectangles show the attributes. The edges show the dependencies.

The Knowledge Graph-Based Approach also extends the relations between entities and attributes based on ConceptNet. Specifically, it extracts five additional relations, as shown in Table 4.4.

doi:10.6342/NTU202404153

Table 4.4: Relations Added by the Knowledge Graph-Based Approach

Relation	According to ConceptNet Relation(s)
SYSTEM → ADMIN	(system, RelatedTo, admin)
SYSTEM → ADMINISTRATOR	(system, RelatedTo, admin) (admin, Synonym, administrator)
SYSTEM → IT	(system, RelatedTo, information_technology) (information_technology, Synonym, it)
SYSTEM → authentication	(system, RelatedTo, network) (network, RelatedTo, authentication)
SYSTEM → inventory	(system, RelatedTo, system_pull) (system_pull, RelatedTo, inventory)

Combining the NLP-Based Approach and the Knowledge Graph-Based Approach, the entities and attributes extracted in this phase are shown in Figure 4.2.

4.4 Aggregates

Then, we group entities into aggregates with the algorithm mentioned in 3.3. In this case, there are no "PartOf," "MadeOf," "HasProperty," or "IsA" relationships between any two entities in ConceptNet. Additionally, terms such as INVENTORY ADMIN, INVENTORY USER, EXCEPTION REQUEST, and REQUEST TEXT are specific to the UUIS system and do not correspond to any concepts in ConceptNet. Therefore, each entity forms its own aggregate, resulting in a total of 10 aggregates (see Figure 4.3).

4.5 Bounded Contexts

Finally, we cluster aggregates into bounded contexts using the method described in 3.4. The method calculates the distances between aggregate pairs with WordNet Similarity and ConceptNet Relatedness, and then applies the two-stage clustering algorithm outlined

doi:10.6342/NTU202404153

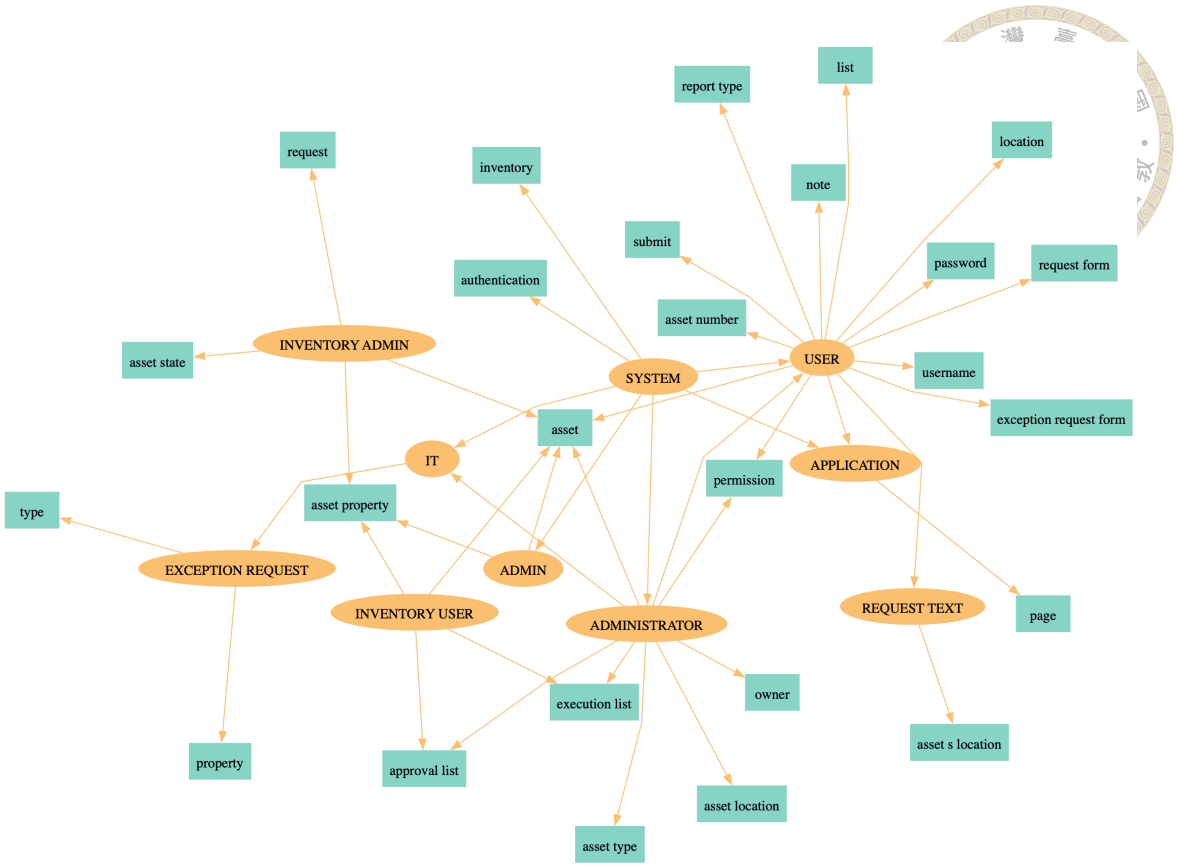


Figure 4.2: The result of the entity/attribute extraction. The ellipses denote the entities, while the rectangles denote the attributes. The edges show the dependencies.

in Section 3.4.2. The distances between aggregates are illustrated in Figure 4.4. Following the distance values, the 10 aggregates are clustered into five subclusters.

- `USER`, `ADMINISTRATOR`, `ADMIN`
- `INVENTORY USER`, `INVENTORY ADMIN`
- `REQUEST TEXT`, `APPLICATION`, `EXCEPTION REQUEST`
- `SYSTEM`
- `IT`

The resulting domain model is shown as 4.5.

doi:10.6342/NTU202404153

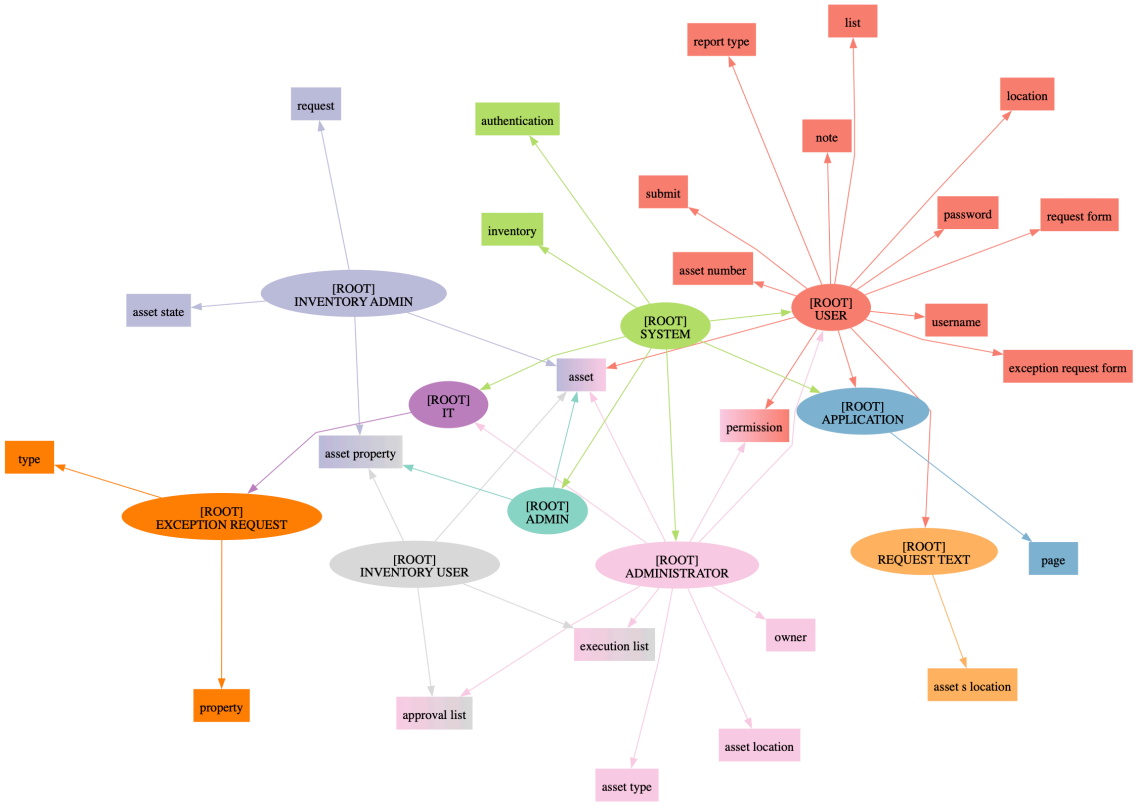


Figure 4.3: The result of the aggregate identification. The ellipses denote the entities, while the rectangles denote the attributes. Entities with the same color belong to the same aggregate. [ROOT] denotes the aggregate roots.

doi:10.6342/NTU202404153

Figure 4.4: The clustering result. Each data point corresponds to an aggregate. Aggregates shown in the same color belong to the same group. The coordinates of each data point are obtained by translating the distance matrix into points on a two-dimensional Cartesian space via multidimensional scaling (MDS).

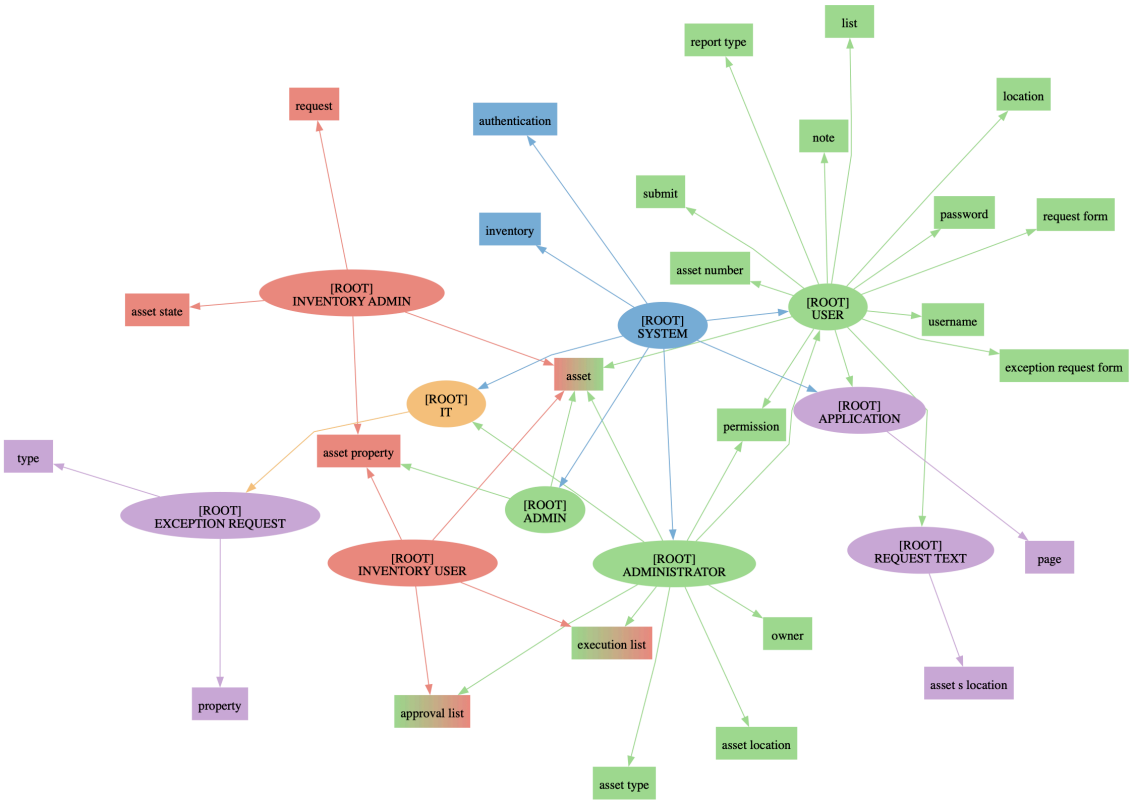


Figure 4.5: The result of the clustered bounded contexts. Entities with the same color belong to the same bounded context.

doi:10.6342/NTU202404153

Chapter 5 Conclusion

This work presents an automated Event Storming approach that derives bounded contexts from software requirements and use case specifications. The approach first extracts domain events from the requirements and the use case specifications. Then, it uses natural language processing techniques and external knowledge bases to extract the entities and attributes implicit in these domain events. Next, it groups entities into aggregates and then bounded contexts based on their properties as illustrated in Domain-Driven Design. By leveraging semantic and syntactic information from the software requirements and the ConceptNet knowledge base, this approach reduces reliance on domain experts as typically required for Event Storming sessions, thereby ensuring the agility of the software design process.

doi:10.6342/NTU202404153

Chapter 6 Future Work

In the future, we plan to develop quantitative evaluation methods to assess the extraction of entities, aggregates, and bounded contexts in the automated Event Storming process from a modularity perspective.

Additionally, we plan to incorporate more external knowledge bases to expand the range of supported domains. Currently, we only utilize ConceptNet and WordNet as external knowledge sources. However, each knowledge base has its limitations, and relying on a limited number of knowledge bases restricts the domains that can be covered. Therefore, we aim to integrate additional external knowledge bases to broaden the scope of knowledge coverage.

doi:10.6342/NTU202404153

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[2] A. Alhazmi, A. Al-Sharawi, B. Liu, D. Oliveira, K. Sobh, M. Mayantz, R. de Bled, and Y. M. Zhang. Software requirements specification of the iufa's uuis – a team 4 comp5541-w10 project approach, 2010.

[3] H. Bogumiła, H. Zbigniew, T. Lech, and D. Iwona. Conceptual modeling using knowledge of domain ontology. In N. T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong, editors, Intelligent Information and Database Systems, pages 554–564, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[4] A. Brandolini. Introducing event storming. blog, Ziobrando' s Lair, 18, 2013.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Ponde, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. W. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, I. Babuschkin, S. Balaji, S. Jain, A. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Rad-

doi:10.6342/NTU202404153

ford, M. M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. ArXiv, abs/2107.03374, 2021.

[6] B. Curtis. Insights from empirical studies of the software design process. Future Generation Computer Systems, 7(2):139–149, 1992.

[7] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd, volume 96, pages 226–231, 1996.

[8] E. Evans. Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2004.

[9] Explosion. spacy-experimental: Cutting-edge experimental spacy components and features. https://github.com/explosion/spacy-experimental, Nov. 2023.

[10] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou. CodeBERT: A pre-trained model for programming and natural languages. In T. Cohn, Y. He, and Y. Liu, editors, Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1536–1547, Online, Nov. 2020. Association for Computational Linguistics.

[11] W. N. Francis and H. Kucera. Brown corpus manual. Technical report, Department of Linguistics, Brown University, 1979.

[12] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020.

[13] C. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment

doi:10.6342/NTU202404153

analysis of social media text. In Proceedings of the international AAAI conference on web and social media, volume 8, pages 216–225, 2014.

[14] J. Inc. Codegpt: Ai coding for developers. https://codegpt.co/, 2024.

[15] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d' Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. Competition-level code generation with alphacode. Science, 378(6624):1092–1097, 2022.

[16] D. Lin et al. An information-theoretic definition of similarity. In Icml, volume 98, pages 296–304, 1998.

[17] Y.-L. Lin. From requirements to microservice: A domain driven approach with machine learning. Master's thesis, National Taiwan University, 2023.

[18] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy approach to requirements syntax (ears). In 2009 17th IEEE International Requirements Engineering Conference, pages 317–322, 2009.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[20] G. A. Miller. Wordnet: a lexical database for english. Communications of the ACM, 38(11):39–41, 1995.

[21] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word rep-

doi:10.6342/NTU202404153

resentation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

[22] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 20:53–65, 1987.

[23] S. Si-Said Cherfi, S. Ayad, and I. Comyn-Wattiau. Improving business process model quality using domain ontologies. Journal on Data Semantics, 2:75–87, 2013.

[24] R. Speer, J. Chin, and C. Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge, 2017.

[25] L. Tan. Pywsd: Python implementations of word sense disambiguation (wsd) technologies [software]. https://github.com/alvations/pywsd, 2014.

[26] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. Journal of the American statistical association, 58(301):236–244, 1963.

[27] R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Transactions on Systems, Man, and Cybernetics, 18(1):183–190, 1988.

[28] D. Zan, B. Chen, D. Yang, Z. Lin, M. Kim, B. Guan, Y. Wang, W. Chen, and J.-G. Lou. CERT: Continual pre-training on sketches for library-oriented code generation. In The 2022 International Joint Conference on Artificial Intelligence, 2022.

doi:10.6342/NTU202404153

Appendix A — WordNet Lexical Semantic Categories

A.1 Action Verbs

Table A.1: WordNet Lexical Semantic Categories of Action Verbs

Category	Description	Example
verb.body	Verbs of grooming, dressing and bodily care	act
verb.change	Verbs of size, temperature change, intensifying, etc.	mutate
verb.cognition	Verbs of thinking, judging, analyzing, doubting	solve
verb.communication	Verbs of telling, asking, ordering, singing	ping
verb.competition	Verbs of fighting, athletic activities	tackle
verb.consumption	Verbs of eating and drinking	digest
verb.contact	Verbs of touching, hitting, tying, digging	hold
verb.creation	Verbs of sewing, baking, painting, performing	construct
verb.motion	Verbs of walking, flying, swimming	traverse
verb.perception	Verbs of seeing, hearing, feeling	experience
verb.possession	Verbs of buying, selling, owning	import
verb.social	Verbs of political and social activities and events	pursue
verb.weather	Verbs of raining, snowing, thawing, thundering	persist

doi:10.6342/NTU202404153

A.2 Change-Inducing Verbs

Table A.2: WordNet Lexical Semantic Categories of Change-Inducing Verbs

Category	Description	Example
verb.change	Verbs of size, temperature change, intensifying, etc.	mutate
verb.cognition	Verbs of thinking, judging, analyzing, doubting	solve
verb.communication	Verbs of telling, asking, ordering, singing	ping
verb.creation	Verbs of sewing, baking, painting, performing	construct
verb.possession	Verbs of buying, selling, owning	import
verb.stative	Verbs of being, having, spatial relations	become
verb.social	Verbs of political and social activities and events	pursue

doi:10.6342/NTU202404153

Appendix B — Relations in ConceptNet

Table B.3: Relations in ConceptNet

Relation	**Description**	**Symmetric or Asymmetric**	**Example**
IsA	A is a subtype or a specific instance of B; every A is a B.	Asymmetric	robin → bird
PartOf	A is a part of B.	Asymmetric	tire → car
HasA	B belongs to A, either as an inherent part or due to a social construct of possession. HasA is often the reverse of PartOf.	Asymmetric	car → 4_tires
AtLocation	A is a typical location for B, or A is the inherent location of B.	Asymmetric	computers → office
Synonym	A and B have very similar meanings.	Symmetric	daytime → day

doi:10.6342/NTU202404153

RelatedTo	The most general relation. There is some positive relationship between A and B, but ConceptNet can't determine what that relationship is based on the data.	Symmetric	trick ↔ magic
FormOf	A is an inflected form of B; B is the root word of A.	Symmetric	slept → sleep
UsedFor	A is used for B; the purpose of A is B.	Asymmetric	pool → swimming
CapableOf	Something that A can typically do is B.	Asymmetric	bottle → hold_water
Causes	A and B are events, and it is typical for A to cause B.	Asymmetric	exercise → sweat
HasSubevent	A and B are events, and B happens as a subevent of A.	Asymmetric	eating → chewing
HasFirstSubevent	A is an event that begins with subevent B. The first thing you do when you A is B.	Asymmetric	eat → wash_hands
HasLastSubevent	A is an event that concludes with subevent B. The last thing you do when you A is B.	Asymmetric	drink → swallow
HasPrerequisite	In order for A to happen, B needs to happen; B is a dependency of A.	Asymmetric	eat → find_food
HasProperty	A has B as a property; A can be described as B.	Asymmetric	circle → round

doi:10.6342/NTU202404153

MotivatedByGoal	Someone does A because they want result B; A is a step toward accomplishing the goal B.	Asymmetric	play → have_fun
Desires	A is a conscious entity that typically wants B.	Asymmetric	bird → fly
CreatedBy	B is a process or agent that creates A.	Asymmetric	cake → baking
Antonym	A and B are opposites in some relevant way, such as being opposite ends of a scale, or fundamentally similar things with a key difference between them.	Symmetric	hot ↔ cold
DistinctFrom	A and B are distinct member of a set; something that is A is not B.	Symmetric	blue ↔ red
DerivedFrom	A is a word or phrase that appears within B and contributes to B's meaning.	Asymmetric	billionaire → billion
SymbolOf	A symbolically represents B.	Asymmetric	trophy → victory
DefinedAs	A and B overlap considerably in meaning, and B is a more explanatory version of A.	Asymmetric	satan → devil
MannerOf	A is a specific way to do B. Similar to "IsA", but for verbs.	Asymmetric	document → record
LocatedNear	A and B are typically found near each other.	Symmetric	shore ↔ ocean

doi:10.6342/NTU202404153

HasContext	A is a word used in the context of B, which could be a topic area, technical field, or regional dialect.	Asymmetric	square → arithmetic
SimilarTo	A is similar to B.	Symmetric	patient ↔ unhurried
Etymologically-RelatedTo	A and B have a common origin.	Symmetric	doctor ↔ doc
Etymologically-DerivedFrom	A is derived from B.	Asymmetric	eyecatch → eye
CausesDesire	A makes someone want B.	Asymmetric	happiness → sing
MadeOf	A is made of B.	Asymmetric	book → paper
ReceivesAction	B can be done to A.	Asymmetric	books → read

doi:10.6342/NTU202404153