國立臺灣大學電資學院資訊工程研究所

碩士論文

Department of Computer Science & Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

藉由機器學習模型
自動將使用案例規格轉換為任務模型

Auto-Transform Use Case Specifications to Task Models Using Machine Learning Model

鍾昀諺

Yun-Yen Chung

指導教授: 李允中 博士

Advisor: Jonathan Lee Ph.D.

中華民國 113 年 7 月

July, 2024

doi:10.6342/NTU202404066

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

藉由機器學習模型自動將使用案例規格轉換為任務模型

Auto-Transform Use Case Specifications to Task Models Using Machine Learning Model

本論文係鍾昀諺君（學號 R11922145）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 CHUNG, YUN-YEN (student ID: R11922145) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

[illegible] （指導教授 Advisor）	劉建宏	[illegible]
[illegible]	王小璠	

系主任/所長 Director: 陳祝嵩

Acknowledgements

首先，我要衷心感謝我的導師，李允中教授。在整個兩年的研究過程中，李允中教授給予了我寶貴的指導和支持，無論是在研究主題的選擇、拆解問題的方法、程式架構設計，還是論文寫作方面，他的建議和指導都對我大有裨益。

其次，我要感謝軟體工程實驗室的所有成員，特別是許容綮、蔡智冠、王埅宇、朱紹瑜、梁乃勻、錢怡君以及學弟妹們。在與他們一起解決問題的過程中，我學到了很多寶貴的知識和技能，並在討論中得到了許多有價值的建議。

此外，我要感謝我的家人和朋友，感謝他們在這段時間裡的支持和鼓勵。你們的理解和支持使我能夠全身心地投入到研究中，克服了重重困難，最終完成了這篇論文。

doi:10.6342/NTU202404066

摘要

在軟體專案的開發過程中，我們通常會在收到需求後撰寫軟體規格書，然後進行軟體設計和實作。這整個流程非常耗時且需要大量人力。為了解決這一問題，我們致力於開發一個任務自動化系統，以實現整個流程的自動化。在這過程中，產生任務模型（Task Model）是一個關鍵步驟。任務模型是一種可以用來表示使用者操作和系統反應的工具，用於描述系統功能和交互行為。

我的主要任務是自動化從使用案例規格到任務模型生成的過程。由於使用案例規格中的任務具有多樣性和複雜性，很難通過規則的方法生成完全對應的任務模型資訊，因此機器學習技術是必不可少的。我們特別重視自然語言處理（NLP）的應用，因為我們的任務涉及到文本的預測和分類問題。生成的任務模型可以用於後續的前端介面設計和網路應用程式開發，從而提高整個開發流程的效率和質量。

關鍵字：任務模型、使用者需求、機器學習、使用案例規格

doi:10.6342/NTU202404066

Abstract

In the software project development process, we usually write software specification documents after getting the requirements, followed by software design and implementation. This entire process is very time-consuming and requires significant human resources. To solve this problem, we are committed to developing a task automation system to simplify the entire process. Generating task models is a key step in this process. Task models are tools used to represent user actions and system responses, which are essential for describing system functionalities and interaction behaviors.

My primary task is to automate the process from use case specifications to task model generation. Due to the diversity and complexity of tasks within use case specifications, it is challenging to generate fully corresponding task model information through rule-based methods. Consequently, machine learning techniques are essential. We particularly focus on the application of natural language processing (NLP) because our task involves text prediction and classification issues. The generated task models can be employed for sub-

doi:10.6342/NTU202404066

sequent front-end interface design and web application development, thereby enhancing the efficiency and quality of the entire development process.

Keywords: Task Model, User Requirements, Machine Learning, Use Case Specification

doi:10.6342/NTU202404066

Contents

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

List of Figures

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

doi:10.6342/NTU202404066

List of Tables

doi:10.6342/NTU202404066

Chapter 1 Introduction

In the development process of software projects, from requirement acquisition to the production of a complete web application, the entire workflow is highly time-consuming and labor-intensive. To address this issue, we are dedicated to developing a system that automates the entire process. This process includes converting user requirements into the EARs requirement style, then using rules to transform EARs requirements into Use Case Specifications. Subsequently, natural language processing models are employed to generate Task Models, which are ultimately used to generate the corresponding UI designs and clients.

This thesis focuses on the process of generating Task Models from Use Case Specifications. Due to the linguistic diversity of Use Case Specifications and the wide variety of information within Task Models, it is hard to create the needed content using only rule-based methods. Therefore, natural language processing techniques within machine learning models must be utilized to accomplish this task.

In Chapter 2, we discuss related work, including the T5 model, influence functions, Task Models, and ConcurTaskTree (CTT). Chapter 3 provides an overview of the information in Use Case Specifications and the data formats of the generated Task Models, as well as the elements that need to be predicted using machine learning. Chapter 4 details

doi:10.6342/NTU202404066

how the T5 model is trained to generate the required information and explains the entire automated Task Model generation process.

In Chapter 5, we use Influence Function techniques to identify high-quality and low-quality datasets during training and perform several experiments. Chapter 6 summarizes the research work and discusses future research directions. The appendix presents the complete output results of the Inventory project from Use Case Specifications to Task Models.

doi:10.6342/NTU202404066

Chapter 2 Related Work

In this chapter, the relevant technologies referenced and used in this research will be introduced, and their applications in the study will be explained.

2.1 Task Model

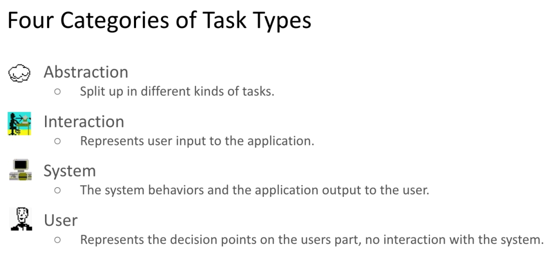
The Task Model is a tool used to describe user operations and system responses, playing a crucial role in both front-end design and back-end interactions. The Task Model clearly demonstrates user operation flows, providing designers with guidance for creating intuitive and efficient user interfaces, while also helping back-end developers understand and implement system functional requirements and data exchange methods. By applying the Task Model, front-end and back-end teams can collaborate effectively, improving communication efficiency and overall quality during the development process, thereby ensuring the final product meets user needs and provides a smooth user experience.

2.2 Concur Task Tree (CTT)

CTT (ConcurTaskTrees) is a visual representation of the Task Model, making it easy to understand and communicate. Its main features include a focus on activities, hierarchi-

doi:10.6342/NTU202404066

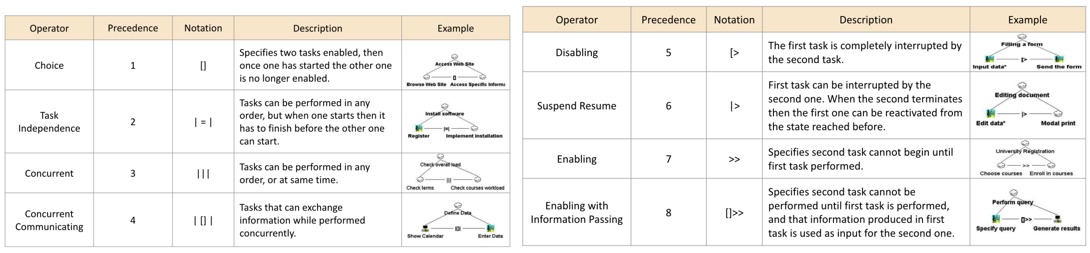
cal structure, graphical syntax, rich temporal operators, task allocation, and the attributes of objects and tasks. Through its hierarchical structure, CTT organizes tasks into layers, clearly displaying the hierarchical relationships and internal logic between tasks. Graphical syntax refers to the use of graphical elements to express and organize various tasks and their relationships within the Task Model. Each task is represented by a node, which can be divided into four types (Figure 2.1): User, System, Interaction, and Abstraction, with their meanings illustrated in the figure. Connections between nodes indicate the sequence and dependency of tasks, while temporal operators describe the temporal relationships between tasks. There are eight types of temporal operators (Figure 2.2), each representing different temporal relationships such as parallelism, sequence, and choice.



Four Categories of Task Types

- Abstraction
  - Split up in different kinds of tasks.
- Interaction
  - Represents user input to the application.
- System
  - The system behaviors and the application output to the user.
- User
  - Represents the decision points on the users part, no interaction with the system.

Figure 2.1: Task Categories Summary



Operator	Precedence	Notation	Description	Example
Choice	1	[]	Specifies two tasks enabled, then once one has started the other one is no longer enabled.	Access Web Site; Browse Web Site [] Access Specific Informa
Task Independence	2	| = |	Tasks can be performed in any order, but when one starts then it has to finish before the other one can start.	Install software; Register |=| Implement installation
Concurrent	3	| | |	Tasks can be performed in any order, or at same time.	Check overall load; Check terms ||| Check courses workload
Concurrent Communicating	4	| [] |	Tasks that can exchange information while performed concurrently.	Define Data; Show Calendar |[]| Enter Data
Disabling	5	[>	The first task is completely interrupted by the second task.	Filling a form; Input data* [> Send the form
Suspend Resume	6	|>	First task can be interrupted by the second one. When the second terminates then the first one can be reactivated from the state reached before.	Editing document; Edit data* |> Modal print
Enabling	7	>>	Specifies second task cannot begin until first task performed.	University Registration; Choose courses >> Enroll in courses
Enabling with Information Passing	8	[]>>	Specifies second task cannot be performed until first task is performed, and that information produced in first task is used as input for the second one.	Perform query; Specify query []>> Generate results

Figure 2.2: Temporal Operator Summary

2.3 T5 Model

T5 (Text-to-Text Transfer Transformer) [6] is a model that turns all NLP tasks into a ”text-to-text” format, where both the input and output are text. T5 is based on the Transformer architecture, consisting of an encoder and a decoder. It is fine-tuned on the pre-trained model to adapt to various application scenarios. The same model and architecture can be used to solve different tasks, including **text classification**, sentiment analysis, machine translation, and **summarization**. For summarization, T5 works very well, creating clear and coherent text summaries ideal for academic research and content selection. Additionally, T5 is highly effective in text classification tasks, categorizing text based on content, which can be applied to sentiment analysis, spam detection, and topic classification. These features make T5 a powerful and flexible tool, demonstrating excellent performance across a variety of NLP tasks.

2.4 Influence Function

The influence function was initially introduced in robust statistics and was reintroduced to deep learning by Koh and Liang (2017) [3]. They demonstrated the practicality of influence functions in identifying data points that significantly impact model parameter estimates. The basic formula for the influence function is as follows:

$$I(z, z') = -\nabla_\theta L(z, \hat{\theta}) H_\theta^{-1} \nabla_\theta L(z', \hat{\theta}) \tag{2.1}$$

where，$L(z, \hat{\theta})$is loss function，H_θ is the Hessian matrix of the parameters θ ，z and z' are the training sample and test sample, respectively.

Specifically, the influence function can help compute the impact value of all training data points on a particular test data point, thereby identifying data points that are harmful or beneficial to the model.

This is very useful in many applications, such as understanding model behavior, han-

doi:10.6342/NTU202404066

dling adversarial training samples, debugging domain mismatches, and correcting mislabeled examples.

Based on these foundations, using the influence function to identify and remove harmful training samples for the T5 model's generation tasks aims to enhance the model's robustness and output quality.

doi:10.6342/NTU202404066

Chapter 3 Task Model Generation

3.1 Task Model Introduction

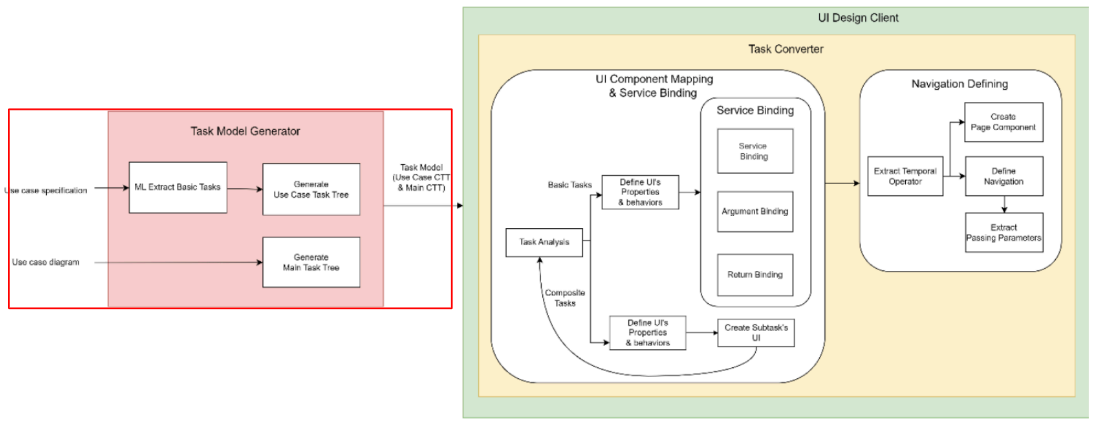
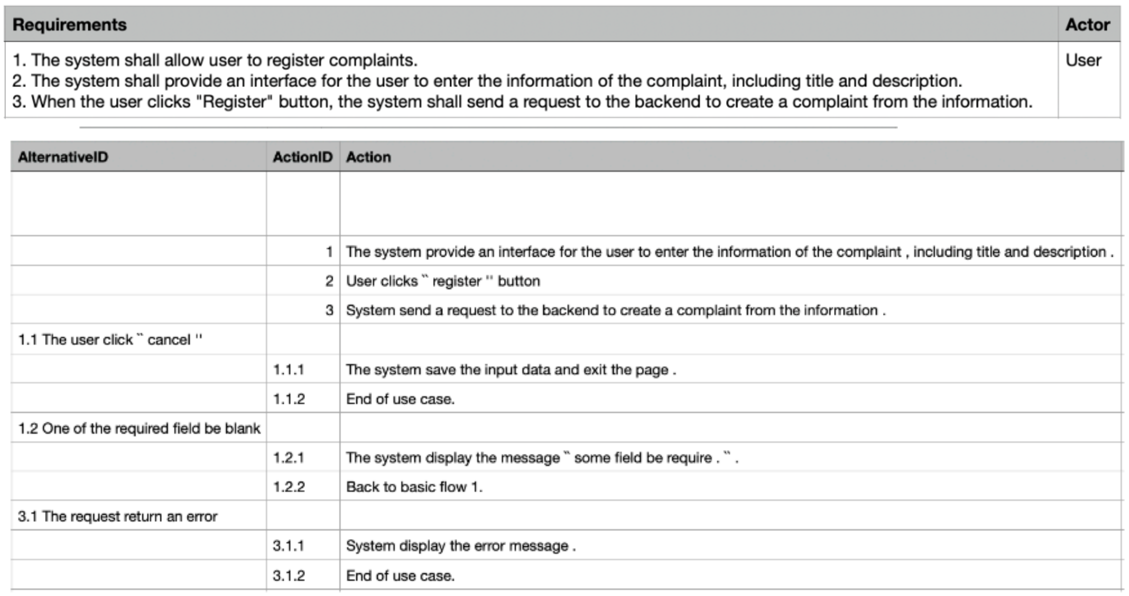


Figure 3.1: System Architecture of UI Generating System



Requirements	Actor
1. The system shall allow user to register complaints. 2. The system shall provide an interface for the user to enter the information of the complaint, including title and description. 3. When the user clicks "Register" button, the system shall send a request to the backend to create a complaint from the information.	User

AlternativeID	ActionID	Action
	1	The system provide an interface for the user to enter the information of the complaint , including title and description .
	2	User clicks ‶ register ″ button
	3	System send a request to the backend to create a complaint from the information .
1.1 The user click ‶ cancel ″		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‶ some field be require . ‶ .
	1.2.2	Back to basic flow 1.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

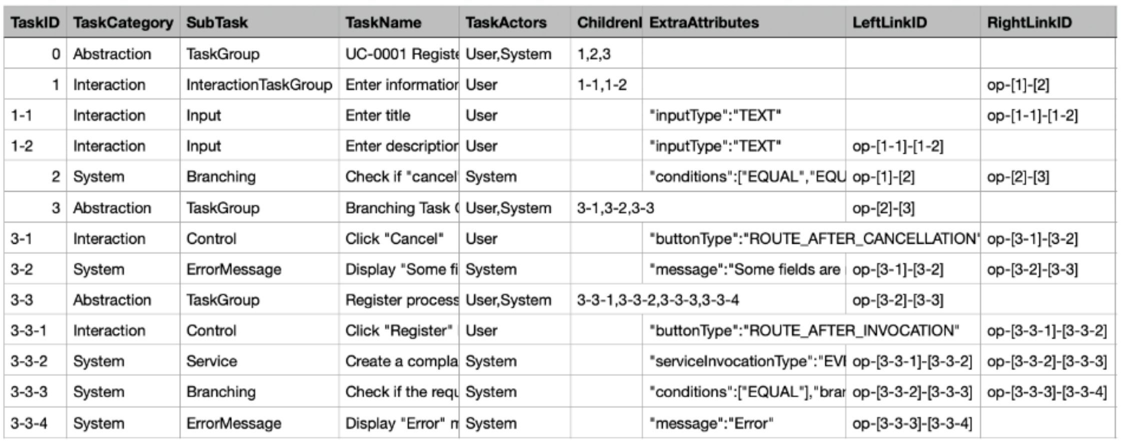
Figure 3.2: Use Case Specification

doi:10.6342/NTU202404066

The system architecture diagram is shown in (Figure 3.1), with the responsible part highlighted in the red block on the left. The main study begins with the input of the Use Case Specification, where the information is derived from EARs requirements (upper part of Figure 3.2) through rule-based transformation. The Use Case Specification (lower part of Figure 3.2) consists of three components: actions, requirements, and actors. An example of this is the ”Register Complaint” use case.

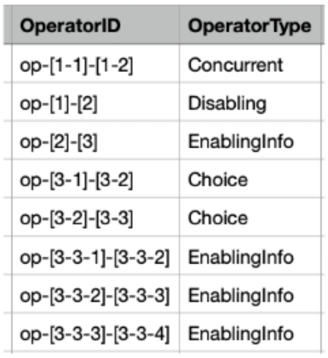
After a series of prediction steps, a Use Case CTT (Figure 3.3), which is a Task Model, is generated. These steps will be detailed in the following sections. The data generated from these steps undergoes a visualization process, as shown in (Figure 3.4). In addition to the Use Case CTT, a Main CTT must also be generated, representing the relationship between use cases. Therefore, a Use Case Diagram (Figure 3.5) is needed as input to generate this model.

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenI	ExtraAttributes	LeftLinkID	RightLinkID
0	Abstraction	TaskGroup	UC-0001 Registe	User,System	1,2,3			
1	Interaction	InteractionTaskGroup	Enter informatior	User	1-1,1-2			op-[1]-[2]
1-1	Interaction	Input	Enter title	User		"inputType":"TEXT"		op-[1-1]-[1-2]
1-2	Interaction	Input	Enter descriptior	User		"inputType":"TEXT"	op-[1-1]-[1-2]	
2	System	Branching	Check if "cancel	System		"conditions":["EQUAL","EQU	op-[1]-[2]	op-[2]-[3]
3	Abstraction	TaskGroup	Branching Task (	User,System	3-1,3-2,3-3		op-[2]-[3]	
3-1	Interaction	Control	Click "Cancel"	User		"buttonType":"ROUTE_AFTER_CANCELLATION"		op-[3-1]-[3-2]
3-2	System	ErrorMessage	Display "Some fi	System		"message":"Some fields are	op-[3-1]-[3-2]	op-[3-2]-[3-3]
3-3	Abstraction	TaskGroup	Register process	User,System	3-3-1,3-3-2,3-3-3,3-3-4		op-[3-2]-[3-3]	
3-3-1	Interaction	Control	Click "Register"	User		"buttonType":"ROUTE_AFTER_INVOCATION"		op-[3-3-1]-[3-3-2]
3-3-2	System	Service	Create a compla	System		"serviceInvocationType":"EVI	op-[3-3-1]-[3-3-2]	op-[3-3-2]-[3-3-3]
3-3-3	System	Branching	Check if the requ	System		"conditions":["EQUAL"],"brar	op-[3-3-2]-[3-3-3]	op-[3-3-3]-[3-3-4]
3-3-4	System	ErrorMessage	Display "Error" m	System		"message":"Error"	op-[3-3-3]-[3-3-4]	

(a) Task



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

(b) Temporal Operator

Figure 3.3: Use Case CTT

doi:10.6342/NTU202404066

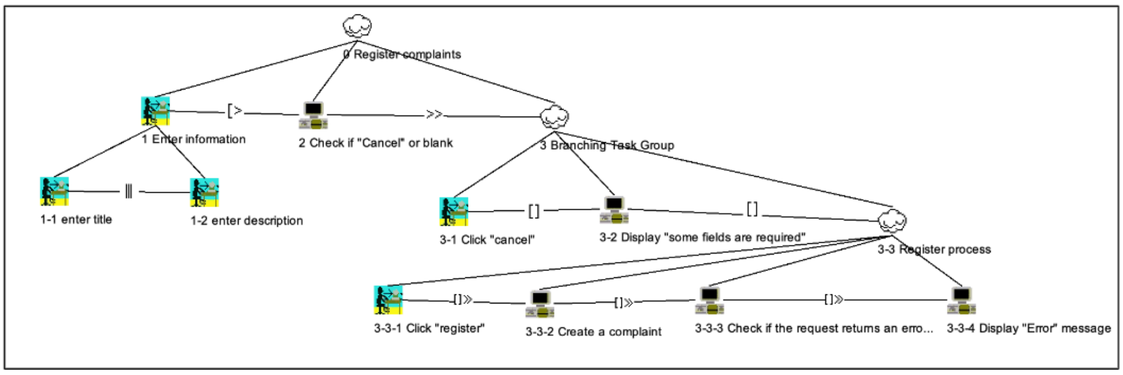
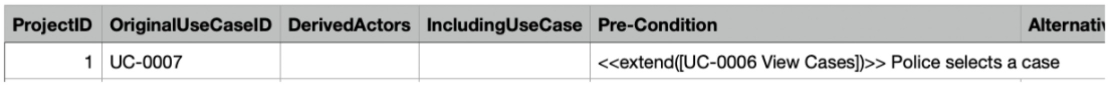


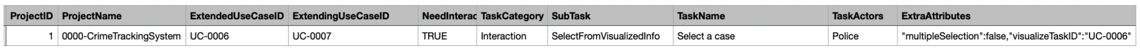
Figure 3.4: Visualized Use Case CTT

(Figure 3.5) presents an example of a Use Case Diagram. By predicting the Interaction Tasks, which represent the actions required to trigger transitions between use cases, we can obtain the necessary data to construct the Main CTT. This data is shown in (Figure 3.6. Finally, the data is transformed into a tree structure according to specific rules, forming the Main CTT. (Figure 3.7) illustrates a visual example of the Main CTT.



ProjectID	OriginalUseCaseID	DerivedActors	IncludingUseCase	Pre-Condition	Alternati
1	UC-0007			<<extend([UC-0006 View Cases])>> Police selects a case	

Figure 3.5: Use Case Diagram



ProjectID	ProjectName	ExtendedUseCaseID	ExtendingUseCaseID	NeedInterac	TaskCategory	SubTask	TaskName	TaskActors	ExtraAttributes
1	0000-CrimeTrackingSystem	UC-0006	UC-0007	TRUE	Interaction	SelectFromVisualizedInfo	Select a case	Police	"multipleSelection":false,"visualizeTaskID":"UC-0006"

Figure 3.6: Main CTT

doi:10.6342/NTU202404066

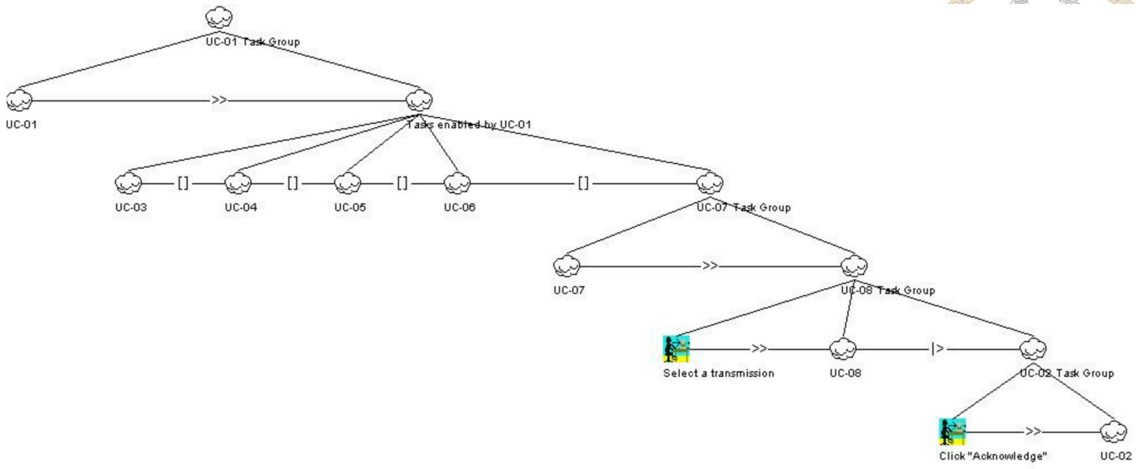


Figure 3.7: Visualized Main CTT Example

During the creation of the CTTs, we made predictions for five tasks: Task, Task Type, Extra Attribute, Temporal Operator, and Interaction Task. The Interaction Task is mainly used to create the Main CTT.

3.2 Alternative and Exception Flows in Task Model

Compared to previous work [5], alternative flow and exception flow have been added to the Task Model. For example, in the ”Register Complaints” use case, when the requirement states ”The system provides an interface for ...” (Figure 3.8a), the system automatically generates two branch flow diagrams.(3.8b)：

1. Click ”Cancel”
2. Some fields are blank.

The corresponding Task Model is shown in (Figure 3.8c):

1. Enter information

doi:10.6342/NTU202404066

2. Check if there are any empty fields or if the action was canceled, with the extra attribute containing all possible task IDs for branching

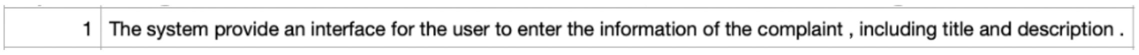
3. Branching Task Group

   (a) Alternative flow: Cancellation

   (b) Exception flow: Display a message ”some fields are blank”

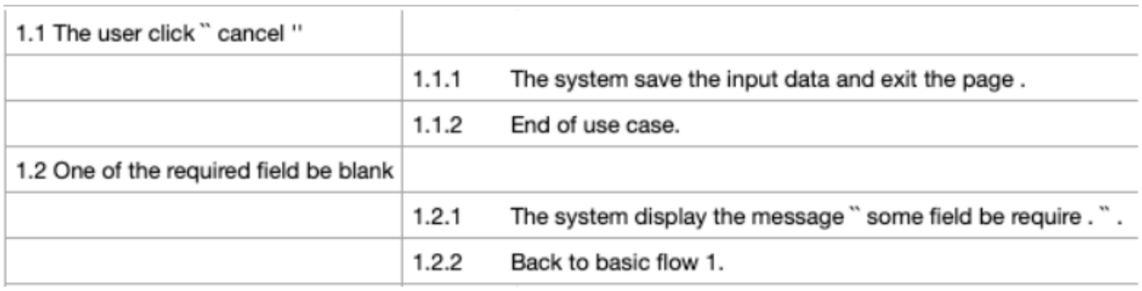
   (c) Next step in the Basic flow

The visualized task model generated from these steps is shown in (Figure 3.8d).



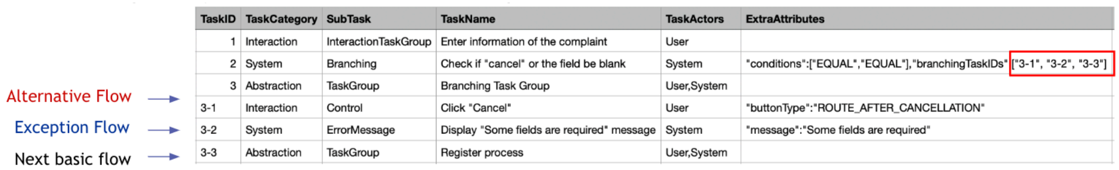
1	The system provide an interface for the user to enter the information of the complaint , including title and description .

(a) Requirement: ”The system provide...”



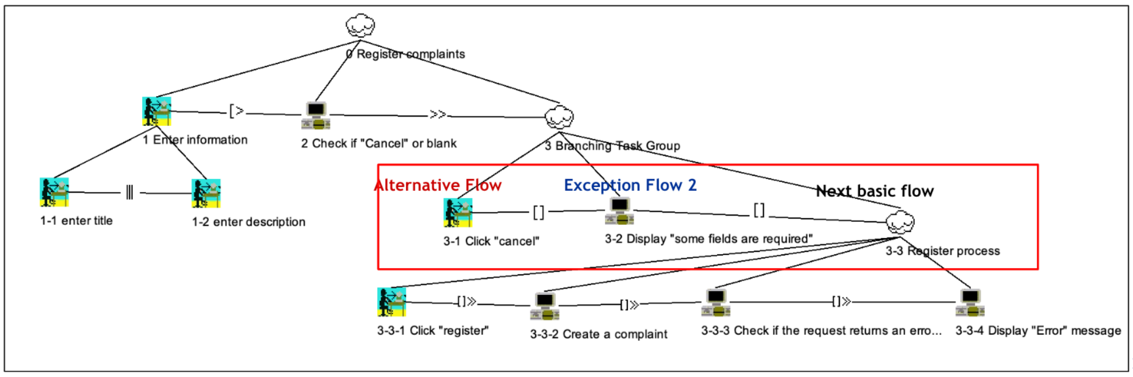
1.1 The user click ˋ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ˋ some field be require . ˋ .
	1.2.2	Back to basic flow 1.

(b) Use Case Spec: branch flow



	TaskID	TaskCategory	SubTask	TaskName	TaskActors	ExtraAttributes
	1	Interaction	InteractionTaskGroup	Enter information of the complaint	User	
	2	System	Branching	Check if "cancel" or the field be blank	System	"conditions":["EQUAL","EQUAL"],"branchingTaskIDs" ["3-1", "3-2", "3-3"]
	3	Abstraction	TaskGroup	Branching Task Group	User,System	
Alternative Flow →	3-1	Interaction	Control	Click "Cancel"	User	"buttonType":"ROUTE_AFTER_CANCELLATION"
Exception Flow →	3-2	System	ErrorMessage	Display "Some fields are required" message	System	"message":"Some fields are required"
Next basic flow →	3-3	Abstraction	TaskGroup	Register process	User,System	

(c) Task Model



(d) Visualized Task Model

Figure 3.8: Various Diagrams Illustrating the Task Models and Flow.

doi:10.6342/NTU202404066

Chapter 4 T5 Model

4.1 Introduction

The full name of T5 is Text-to-Text Transfer Transformer, text-to-text meaning that it unifies the input and output of all types of NLP tasks into a textual form. The T5 model is based on the Transformer architecture, which contains encoders and decoders. Based on the pre-trained model, the T5 model is fine-tuned for a specific task in order to adapt it to various application contexts. As a result, the same model architecture can be used to solve various types of problems, including text classification, sentiment analysis, machine translation, and summarization.

4.2 Parameter Settings

Table 4.1: Experimental Environment Setup

Environments	**Version/Description**
Docker Image	pytorch/pytorch:2.0.1-cuda11.7-cudnn8-runtime
NumPy	1.24.3
Pandas	2.1.0
PyTorch	2.0.1
Transformers	4.33.2
Accelerate	0.23.0
Scikit-learn	1.3.0

doi:10.6342/NTU202404066

Table 4.2: Model Training Setup

Hyper parameter	**Setting**
Training model	t5-base
Training to testing ratio	9 to 1
Prediction metric	Classification: Accuracy Summarization: Rouge-L
Model to be saved	Best metric value
Early stopper	Depend on Task
Optimizer	AdamW
Learning rate	2e-5

4.2.1 Environmental Parameters

To ensure the repeatability of the experiments, the hardware and software environments used are described here in detail. All experiments were performed in the following environment (Table 4.1)

4.2.2 Training Hyperparameters

(Table 4.2) shows the training hyperparameter settings. The model uses t5-base with a 9:1 ratio between training and test sets, and uses accuracy as the predictor for classification questions and ROUGE-L for summarization questions. The final model selects the one with the highest predictor and uses the early stopper mechanism to adapt to the convergence time of different tasks. The optimizer uses AdamW and the learning rate is set to 2×10^{-5}.

4.3 Training

(Fig. 4.1) shows the process of training. The machine learning includes the prediction of five tasks: 1. Task name, IDs, actors; 2. Task Types; 3. Extra Attributes; 4. Temporal

doi:10.6342/NTU202404066

Operator; 5. Interaction Tasks.

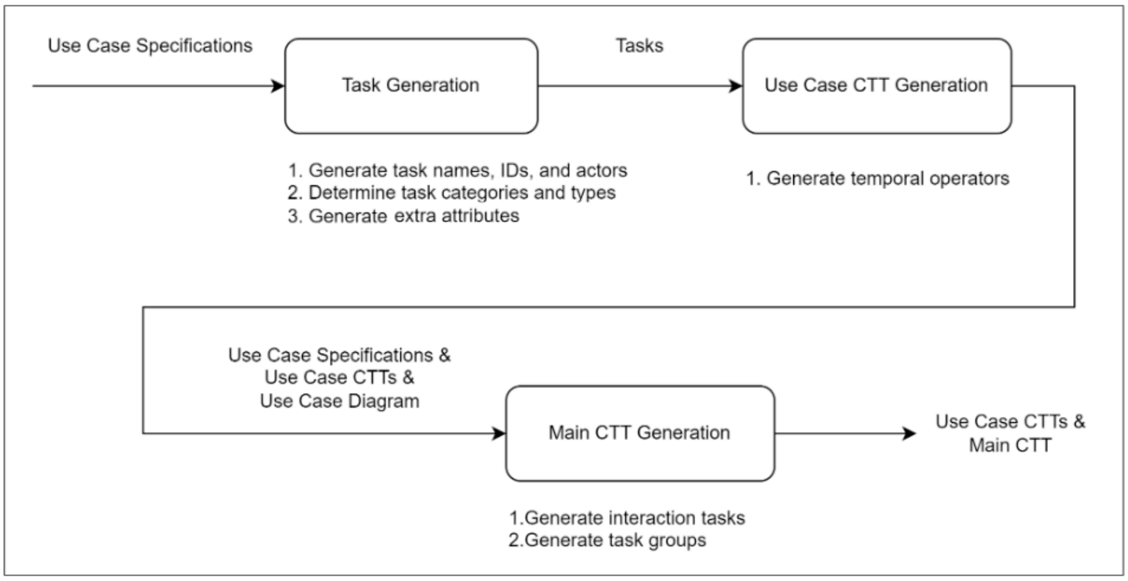

Use Case Specifications
Task Generation
Tasks
Use Case CTT Generation
1. Generate task names, IDs, and actors
2. Determine task categories and types
3. Generate extra attributes
1. Generate temporal operators
Use Case Specifications &
Use Case CTTs &
Use Case Diagram
Main CTT Generation
Use Case CTTs &
Main CTT
1.Generate interaction tasks
2.Generate task groups

Figure 4.1: Training Process

4.3.1 Task Training

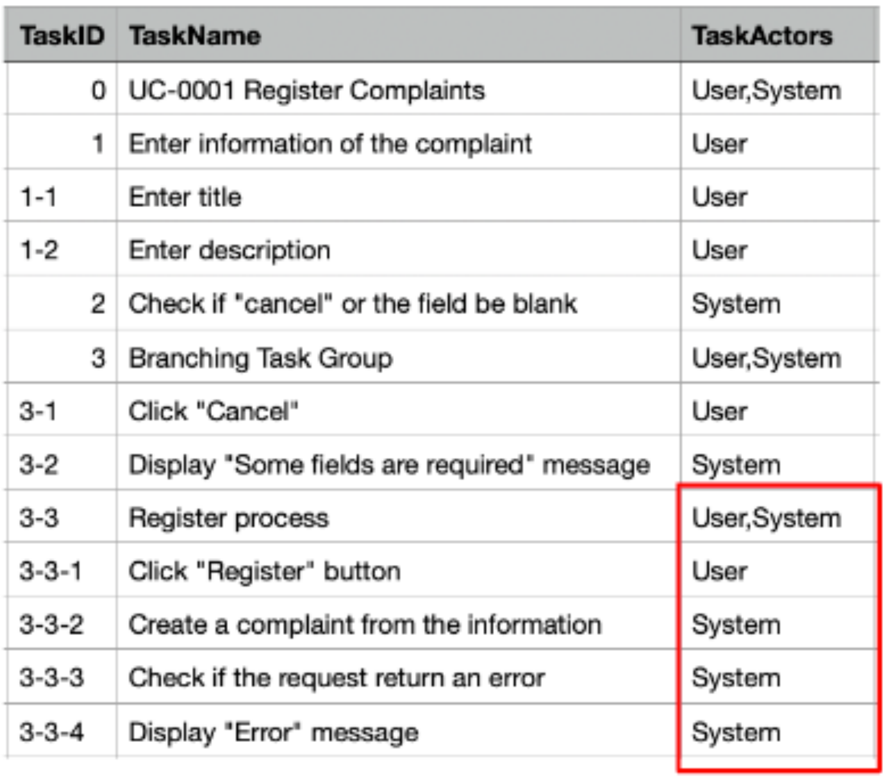
Table 4.3: Task Training Information

Item	**Description**
Model input	Use Case Information, Requirement, Actions in basic, alternative, and exception flows
Model output	Task names and corresponding IDs and actors
Summarization task	Abstractive summarization
Number of samples	464

First, the first step is to train the model to generate Tasks (Table 4.3). The input information includes Use Case Information, Requirement, and Actions in basic, alternative, and exception flows. the desired output is all the text in (Fig 4.2) containing the task ID, actor and name. The part highlighted in red shows that the parent node includes the roles of all its leaf nodes. For instance, if the leaf nodes include a user and a system, the parent node will also contain those roles. This is an abstractive summarization problem with 464 samples.

doi:10.6342/NTU202404066

Figure 4.2: Task Example



TaskID	TaskName	TaskActors
0	UC-0001 Register Complaints	User,System
1	Enter information of the complaint	User
1-1	Enter title	User
1-2	Enter description	User
2	Check if "cancel" or the field be blank	System
3	Branching Task Group	User,System
3-1	Click "Cancel"	User
3-2	Display "Some fields are required" message	System
3-3	Register process	User,System
3-3-1	Click "Register" button	User
3-3-2	Create a complaint from the information	System
3-3-3	Check if the request return an error	System
3-3-4	Display "Error" message	System

4.3.2 Task Type Training

Table 4.4: Task Type Training Information

Item	Description
Model input	Names, IDs, and actors of all tasks
Model output	Category and type of the leaf tasks
Classification task	Multi-class classification (17 types)
Number of samples	3965

The second step is to train the model to generate the Task Type (Table 4.4). The input information includes the Task actor, name and ID generated in the previous step. The desired output is the category and type of the leaf tasks. This is a classification problem with 17 types assigned to the leaf nodes and 3 categories of task groups, totaling 3965 training samples. There are four task categories (Figure 2.1), but these four categories do not correspond to all UI components. Therefore, it is necessary to provide more detailed task types for them in order to map specific tasks to corresponding UI elements, and all

doi:10.6342/NTU202404066

Table 4.5: Task Type Summary

Abstract	Interaction	System
Task Group	Interaction Task Group	System Task Group
Include Task	Select From Fixed List	Checking
	Select From Dynamically Acquired List	Error Message
	Input	Feedback
	Select From Visualized Information	Visualize Fixed Value
	Control	Visualize Dynamically Acquired Value
	Respond Alert	Service
		Input Validation
		Filtering Information
		Branching
		Downloading

task types are written in (table 4.5), and a brief description of each type is as follows.

- **Abstraction**
  - **TaskGroup**: Tasks that contain sub-tasks from different categories.
  - **IncludeTask**: Contains another CTT task.
- **Interaction**
  - **Interaction Task Group**: A task that contains subtasks from the "Interaction" category.
  - **Select From Fixed List**: User selects data from a pre-defined data list.
  - **Select From Dynamically Acquired List**: User selects the data from the Dynamically acquired list.
  - **Input**: User input data.
  - **Select From Visualized Information**: User selects data from the information displayed on the screen.
  - **Control**: User triggers an activity in the system, such as clicking a button.
  - **Respond Alert**: User responds to a system alert.

doi:10.6342/NTU202404066

- **System**
  - **System Task Group**: Tasks that include subtasks from the ”System” category.
  - **Checking**: The system displays messages for user confirmation.
  - **Error Message**: The system displays error or warning messages.
  - **Feedback**: The system shows task progress.
  - **Visualize Fixed Value**: The system displays fixed data.
  - **Visualize Dynamically Acquired Value**: The system displays dynamically acquired data.
  - **Service**: The system triggers backend services.
  - **Input Validation**: The system validates user inputs.
  - **Filter Information**: The system filters data.
  - **Branching**: The system handles alternative flows and exception flows.
  - **Downloading**: The system downloads files.

The category of a parent task can be inferred from its leaf tasks. For example, in (Figure 4.3), the red section ”Click Register” belongs to the category: Interaction, type: Control. The blue section ”Create User” belongs to the category: System, type: Service. The node containing these two leaf tasks is categorized as Abstract because it includes two different categories, and the type of the node itself is Task Group.

4.3.3 Extra Attribute Training

The third step is to train the model to generate the Extra Attribute (Table 4.6). The input information includes the inputs from the previous step, as well as the generated Task

doi:10.6342/NTU202404066

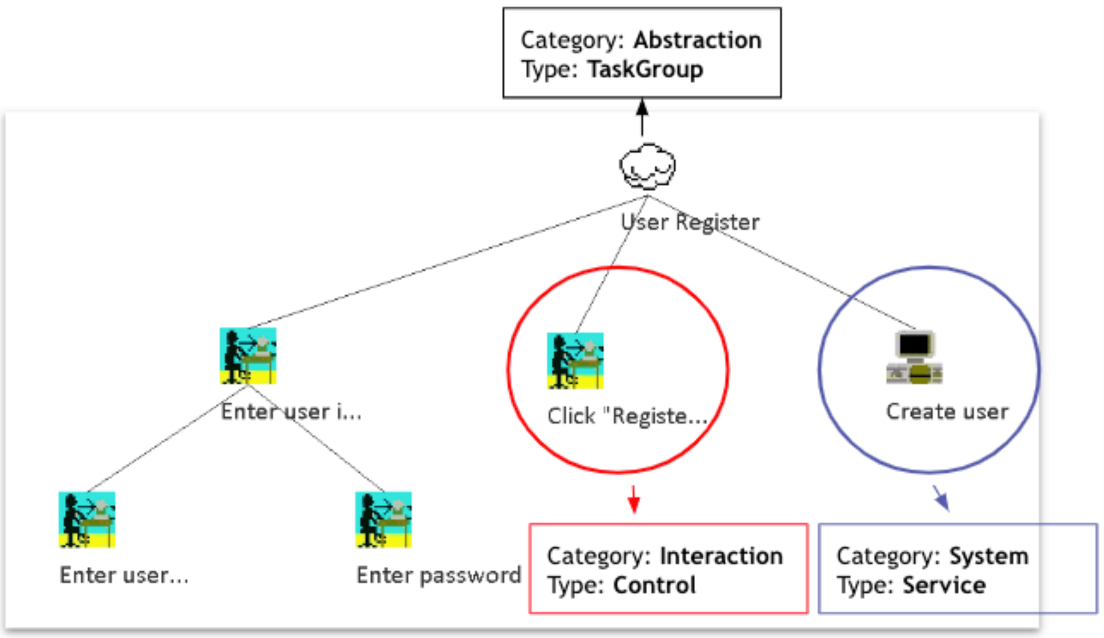


Figure 4.3: Task Type Example

Table 4.6: Extra Attribute Training Information

Item	**Description**
Model input	Names, IDs, and actors of all tasks Task Category and Task Type
Model output	The extra attributes corresponding to task types
Summarization task	Abstractive summarization
Number of samples	3965

Types and Categories. The expected output is the Extra Attributes for all tasks. This is a summarization task with 3965 samples. The following table shows the corresponding Extra Attributes for each Task Type (Table 4.7):

An example of an Extra Attribute is shown in (Figure 4.4). The red-circled ”Click Register” is of the Control type, so it has an Extra Attribute ”Button Type” with the value `"buttonType":"ROUTE_AFTER_INVOCATION"`, indicating that after clicking this button, a request will be sent to the backend. Extra Attributes are needed because Task Type alone can’t fully describe the action. We use these details to give more specific and accurate descriptions.

doi:10.6342/NTU202404066

Table 4.7: Task Types and Extra Attributes

Category	**Task Type**	**Extra Attributes**
Abstract	Task Group	
	Include Task	TreeID
Interaction	Interaction Task Group	
	Input	InputType
	Select From Fixed List	VisualizedTaskID, MultipleSelection
	Select From Dynamically Acquired List	VisualizedTaskID, MultipleSelection
	Control	Button Type
	Select From Visualized Information	VisualizedTaskID, MultipleSelection
	Resp Alert	AlertTaskID
System	System Task Group	
	Checking	Message
	Error Message	Message
	Feedback	
	Filter Info	ServiceTaskID
	Input Validation	ValidatorType
	Visualize Fixed Value	InformationType, Value
	Visualize Dynamically AcquiredValue	Information Type
	Service	ServiceInvocationType
	Branching	BranchingType
	Downloading	TargetTaskID

4.3.4 Temporal Operator Training

Table 4.8: Temporal Operator Training Information

Item	**Description**
Model input	Information of tasks
Model output	Temporal operators
Classification task	Multi-class classification (8 types)
Number of samples	3538

The fourth step is to train the model to generate Temporal Operators (as shown in Table 4.8). This represents the temporal relationships between tasks. The input information includes all the generated task information. The expected output is all the Temporal Operators. This is a classification task with eight categories (as shown in Figure 2.2), with a total of 3538 samples.

doi:10.6342/NTU202404066

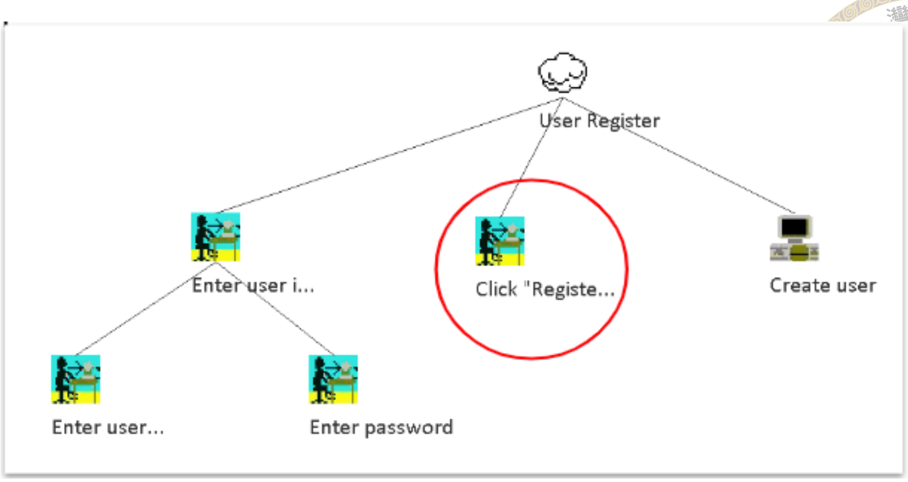
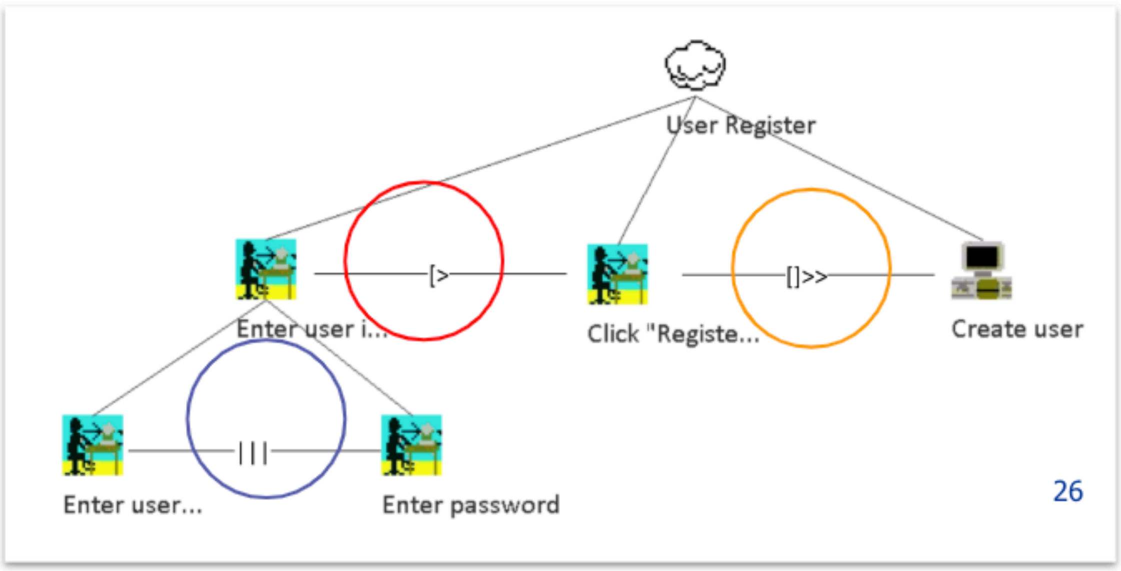


Figure 4.4: Extra Attribute Example

Figure 4.5: Temporal Operator Example



As an example (Figure 4.5), the red circle is ”Disabling”, which means that after the action ”Click Register”, the previous action ”Enter user information” will be disabled. The blue circle is ”Concurrent”, which means ”Enter user name”' and ”Enter password” can be done simultaneously. The orange circle is ”Enabling with Information Passing”, which means that the information entered previously will be sent to the system for user creation after registration is clicked.

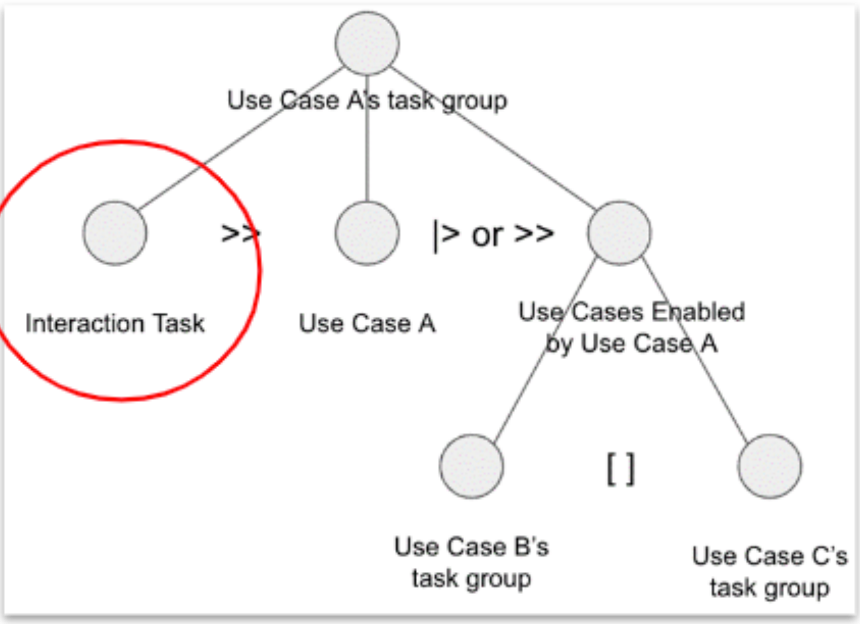
doi:10.6342/NTU202404066

4.3.5 Interaction Task Training

Table 4.9: Interaction Task Training Information

Item	Description
Model input	Use Case Specification of extended use case and extending use case Use Case Diagram
Model output	Interaction task of the task group for the extended use case
Summarization task	Abstractive summarization
Number of samples	355

Figure 4.6: Interaction Task Example



The fifth step is to train the model to generate an Interaction Task (Table 4.9). Sometimes an Interaction Task is needed to trigger between use cases that have an extended relationship. Therefore, this is the part of the task model that must be predicted. Inputs include the contents of the two Use Case Specifications to which the Interaction Task is connected, and the preconditions in the Use Case Diagram. The desired output is information about the Interaction Task, including name, task type, actor, and extra attribute. This is a summarization task with 355 samples. (Figure 4.6) As an example, the red-circled part means there will be an Interaction Task before triggering Use Case A.

doi:10.6342/NTU202404066

4.3.6 Imbalanced Data

Initially, the loss function for the classification problem was Cross Entropy Loss (Equation 4.1). However, due to the imbalance in the dataset, the model tended to label all test data as the most frequent label. This caused the model's predictions to lose accuracy.

$$CE(p, q) = -\sum_{i=1}^{C} w_i p_i \log(q_i) \tag{4.1}$$

To solve this problem, the training data for each category was weighted. Classes with smaller sample sizes are given higher loss penalties when the prediction fails. The weight of the label c is calculated as follows (Equation 4.2):

$$w_{\text{label}} = \frac{\max_{c \in \mathcal{C}} N(c)}{N(\text{label})} \tag{4.2}$$

Here, $N(c)$ represents the number of samples in class c. This weighting method gives more attention to classes with fewer samples, but it may still perform poorly in cases of extreme imbalance.

Therefore, Focal Loss [4], an improved version of Cross Entropy Loss, is introduced here. The Facebook AI Research team developed this loss function by adding a modulation factor to the Cross Entropy Loss, as shown in the following formula (Equation 4.3):

$$FL(p, q) = -\sum_{i=1}^{C} w_i p_i (1 - q_i)^{\gamma} \log(q_i) \tag{4.3}$$

doi:10.6342/NTU202404066

Here, γ is the modulation factor, q_i is the predicted probability, and w_i is the weight. This way, the loss function further reduces the contribution of large sample labels, encouraging the model to focus more on the hard-to-predict samples.

4.3.7 Training Results

Table 4.10: Multi-Class Task

Tasks	Accuracy (Cross Entropy)	Accuracy (Focal Loss gamma=2)
Task Type	0.983	**0.992**
Temporal Operator	0.949	**0.960**

Table 4.11: Summarization Task

Tasks	Accuracy (Rouge-L)
Task	0.970
Extra Attribute	0.985
Interaction Task	0.993

(Tables 4.10 and 4.11) respectively show the accuracy of the five model predictions. For the classification problem, a comparison was made between Cross Entropy and Focal Loss. The results indicate that using Focal Loss significantly improved the model's accuracy.

4.4 Prediction Process

The prediction process can be divided into a total of nine steps (Figure 4.7), where the gray part requires the use of machine learning for prediction, while the white part can be used to generate insufficient information through rules. The first step is to enter the Use Case Specification to generate the task. Next, Task Types are predicted using the trained model. The corresponding Task Category is then generated for each task according to the rules.

doi:10.6342/NTU202404066

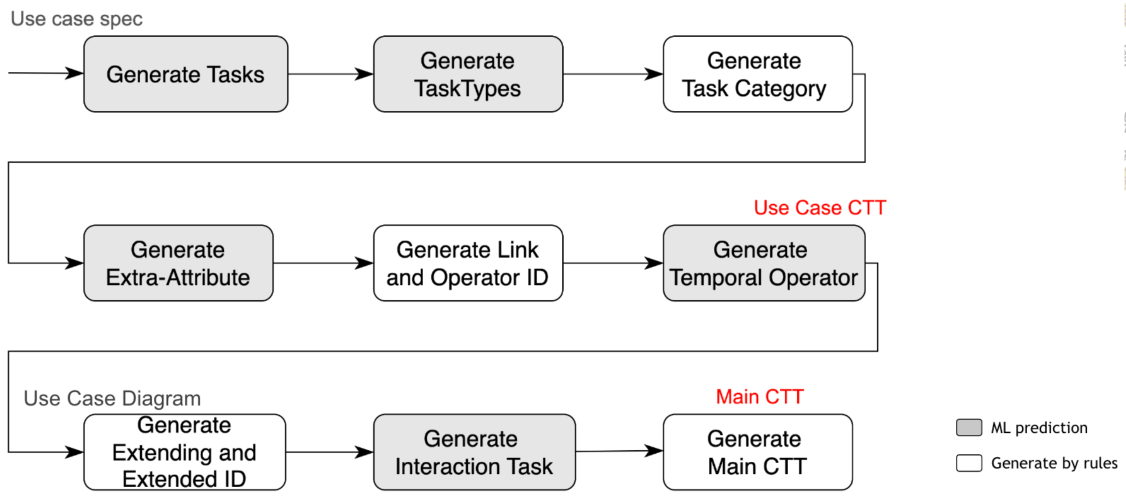


Figure 4.7: Prediction Pipeline

The Extra Attribute is predicted based on the previous information. Next, based on the tasks generated earlier, a rule is used to generate child IDs for each group of tasks, and left and right links for subsequent prediction of Temporal Operators. After that, the trained model is used to predict the Temporal Operators for all the links.

Next, use cases with an extend relationship are extracted from the Use Case Diagram to predict the Interaction Tasks. The trained model is then used to predict each Interaction Task. Finally, a program is used to convert this information into the CTT format, generating the complete Task Models (Use Case CTT and Main CTT) for subsequent steps. Below are examples of the output results for each step:

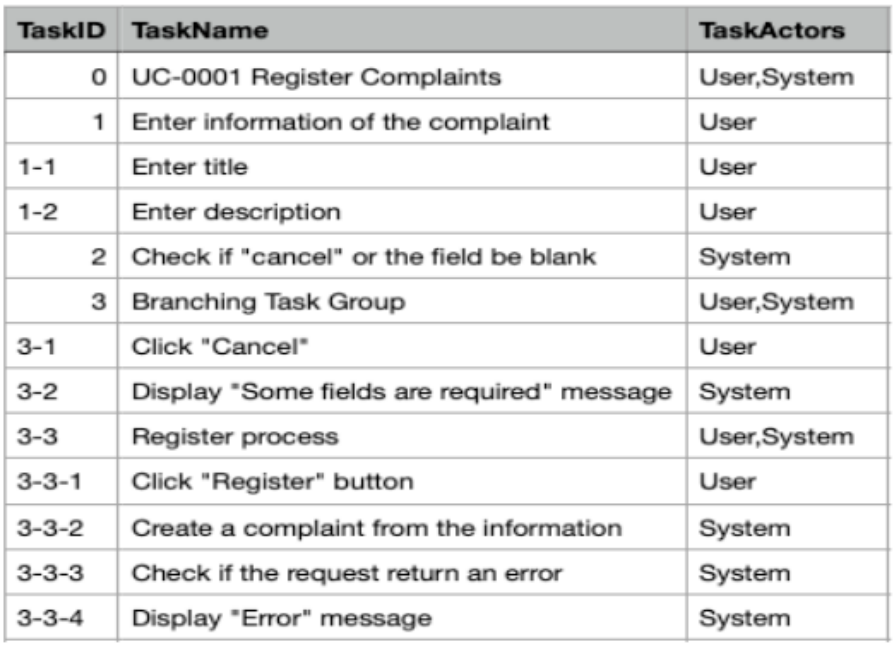
4.4.1 Step1：Task Prediction

Input: Information about the Use Case Specification.

Output: Task with task ID, name and actors.

Example：

doi:10.6342/NTU202404066



TaskID	TaskName	TaskActors
0	UC-0001 Register Complaints	User,System
1	Enter information of the complaint	User
1-1	Enter title	User
1-2	Enter description	User
2	Check if "cancel" or the field be blank	System
3	Branching Task Group	User,System
3-1	Click "Cancel"	User
3-2	Display "Some fields are required" message	System
3-3	Register process	User,System
3-3-1	Click "Register" button	User
3-3-2	Create a complaint from the information	System
3-3-3	Check if the request return an error	System
3-3-4	Display "Error" message	System

Figure 4.8: Task Prediction Example

4.4.2 Step2：Task Types Prediction

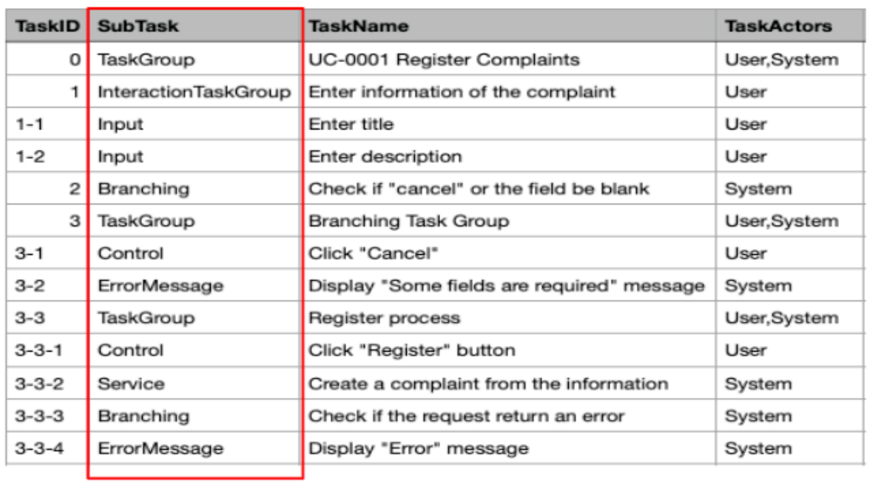
Input: Generated task name, IDs and actor.

Output: Task Types.

Description: Predict the Task Types using the trained model. The model input is the task information, and the output is the corresponding Task Types.

doi:10.6342/NTU202404066

Example：



TaskID	SubTask	TaskName	TaskActors
0	TaskGroup	UC-0001 Register Complaints	User,System
1	InteractionTaskGroup	Enter information of the complaint	User
1-1	Input	Enter title	User
1-2	Input	Enter description	User
2	Branching	Check if "cancel" or the field be blank	System
3	TaskGroup	Branching Task Group	User,System
3-1	Control	Click "Cancel"	User
3-2	ErrorMessage	Display "Some fields are required" message	System
3-3	TaskGroup	Register process	User,System
3-3-1	Control	Click "Register" button	User
3-3-2	Service	Create a complaint from the information	System
3-3-3	Branching	Check if the request return an error	System
3-3-4	ErrorMessage	Display "Error" message	System

Figure 4.9: Task Type Prediction Example

4.4.3 Step3：Task Category Generation

Input: Task Type.

Output: Task Category.

Description: Generate a Task Category based on the Task Type. The rules are as follows:

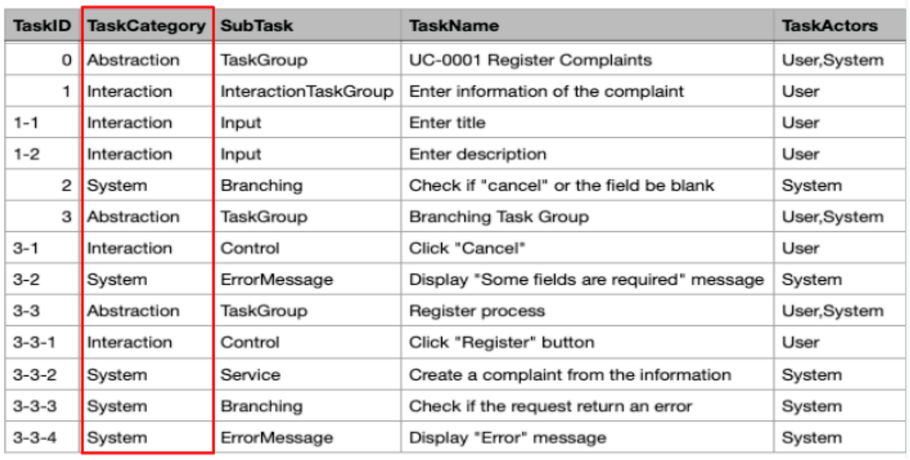
- Based on the Task Type of the leaf node, first assign the corresponding Task Category.
- For non-leaf nodes, assign the category based on the children's categories. If all children have the same category, the non-leaf node's category will be the same as the children's category. If the children belong to two or more different categories, the non-leaf node's category will be Abstraction.

For example, as shown in the diagram, the SubTasks for TaskID 1-1 and 1-2 are ”Input”, so their Task Category is Interaction. The parent node's category is also Interaction since

doi:10.6342/NTU202404066

both its children's categories are Interaction. On the other hand, the SubTasks for TaskID 3-1 and 3-2 are "Control" and "ErrorMessage", so their Task Categories are Interaction and System, respectively. Therefore, the Task Category for TaskID 3 is determined as Abstraction.

Example：



TaskID	TaskCategory	SubTask	TaskName	TaskActors
0	Abstraction	TaskGroup	UC-0001 Register Complaints	User,System
1	Interaction	InteractionTaskGroup	Enter information of the complaint	User
1-1	Interaction	Input	Enter title	User
1-2	Interaction	Input	Enter description	User
2	System	Branching	Check if "cancel" or the field be blank	System
3	Abstraction	TaskGroup	Branching Task Group	User,System
3-1	Interaction	Control	Click "Cancel"	User
3-2	System	ErrorMessage	Display "Some fields are required" message	System
3-3	Abstraction	TaskGroup	Register process	User,System
3-3-1	Interaction	Control	Click "Register" button	User
3-3-2	System	Service	Create a complaint from the information	System
3-3-3	System	Branching	Check if the request return an error	System
3-3-4	System	ErrorMessage	Display "Error" message	System

Figure 4.10: Task Category Generation Example

4.4.4 Step4：Extra Attributes Prediction

Input: The previous information, including task name, IDs, actors and task types.

Output: Extra Attributes.

Description: Use a trained model to predict extra attributes. The model input is task name, IDs, actors and task types. The output is the corresponding extra attributes. In (Figure 4.11), the extra attributes for the branching task (Task ID 2) can be seen: `"conditions": ["EQUAL", "EQUAL"], "branchingTaskIDs":["3-1" , "3-2", "3-3"]`, which contains a condition and corresponding branching task IDs. these branching task IDs point to different tasks (TaskID 3-1, 3-2, 3-3), which are executed according to the different conditions.

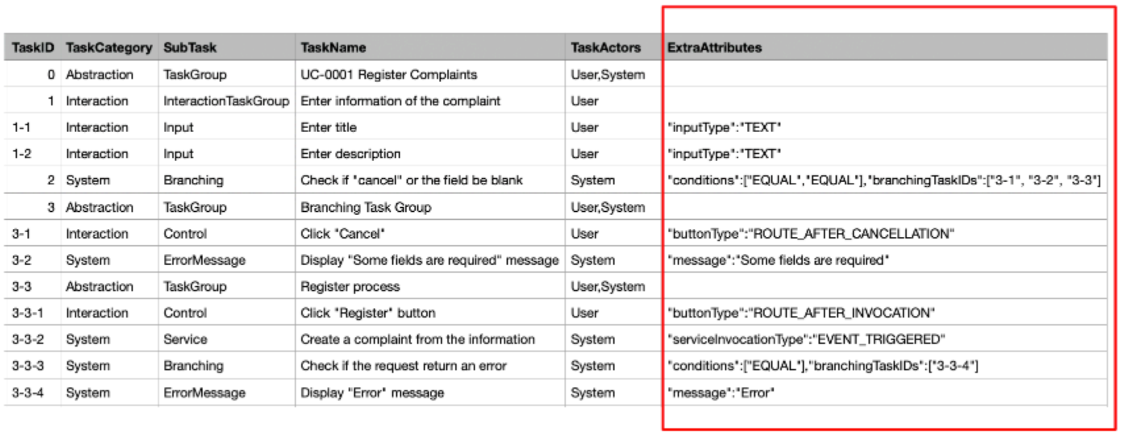
doi:10.6342/NTU202404066

- Task ID 3-1: Indicates that when the condition is ”Cancel,” the ”Click Cancel” task is executed. The extra attribute corresponding to this task is `"buttonType":"ROUTE_AFTER_CANCELL`
which means ”Cancel after clicking.”

- Task ID 3-2: Indicates that when the condition is ”Some fields are required,” the extra attribute for this task is `"message":"some fields are required"`, displaying an error message.

- Task ID 3-3: Indicates that when the condition is ”All fields are correct,” the registration operation is performed.

Example：



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ExtraAttributes
0	Abstraction	TaskGroup	UC-0001 Register Complaints	User,System	
1	Interaction	InteractionTaskGroup	Enter information of the complaint	User	
1-1	Interaction	Input	Enter title	User	"inputType":"TEXT"
1-2	Interaction	Input	Enter description	User	"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the field be blank	System	"conditions":["EQUAL","EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	User,System	
3-1	Interaction	Control	Click "Cancel"	User	"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "Some fields are required" message	System	"message":"Some fields are required"
3-3	Abstraction	TaskGroup	Register process	User,System	
3-3-1	Interaction	Control	Click "Register" button	User	"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Create a complaint from the information	System	"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System	"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System	"message":"Error"

Figure 4.11: Extra Attribute Prediction Example

4.4.5 Step5：Children IDs and Links Generation

Input: The task generated earlier.

Output: Child IDs, left and right links.

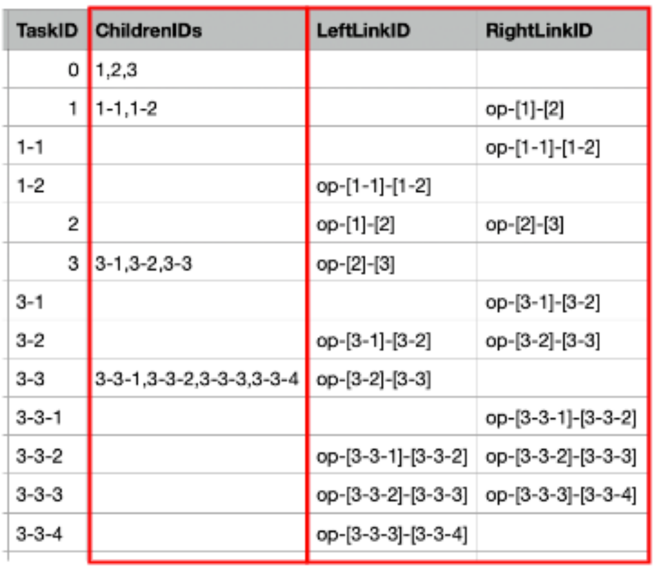
Description：

doi:10.6342/NTU202404066

1. Recursively list all sub-tasks and update ChildrenIDs.

2. Generate LeftLinkID and RightLinkID for each task.

Example：



TaskID	ChildrenIDs	LeftLinkID	RightLinkID
0	1,2,3		
1	1-1,1-2		op-[1]-[2]
1-1			op-[1-1]-[1-2]
1-2		op-[1-1]-[1-2]	
2		op-[1]-[2]	op-[2]-[3]
3	3-1,3-2,3-3	op-[2]-[3]	
3-1			op-[3-1]-[3-2]
3-2		op-[3-1]-[3-2]	op-[3-2]-[3-3]
3-3	3-3-1,3-3-2,3-3-3,3-3-4	op-[3-2]-[3-3]	
3-3-1			op-[3-3-1]-[3-3-2]
3-3-2		op-[3-3-1]-[3-3-2]	op-[3-3-2]-[3-3-3]
3-3-3		op-[3-3-2]-[3-3-3]	op-[3-3-3]-[3-3-4]
3-3-4		op-[3-3-3]-[3-3-4]	

Figure 4.12: Children ID and LinkID Generation Example

4.4.6 Step6：Temporal Operators Prediction

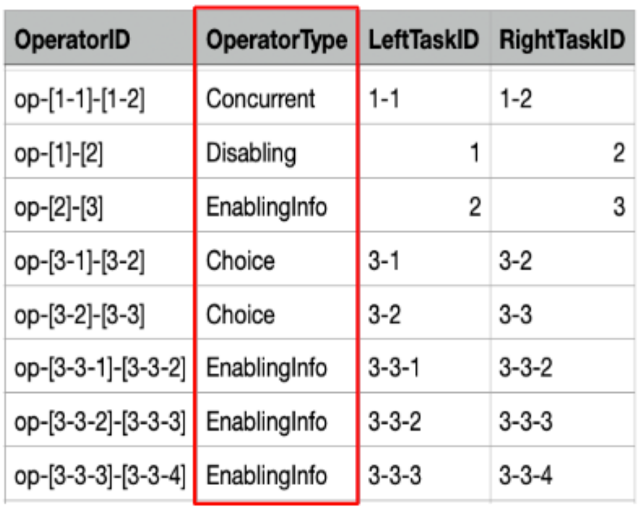
Input: Generated links and all the task information.

Output: Temporal Operators.

Description: Use the trained model to predict all Temporal Operators. See (Figure 4.11), which can correspond to Task IDs 1-1 and 1-2 is Concurrent (Figure 4.13), indicating that the two input operations can be performed in parallel.

doi:10.6342/NTU202404066

Example：



OperatorID	OperatorType	LeftTaskID	RightTaskID
op-[1-1]-[1-2]	Concurrent	1-1	1-2
op-[1]-[2]	Disabling	1	2
op-[2]-[3]	EnablingInfo	2	3
op-[3-1]-[3-2]	Choice	3-1	3-2
op-[3-2]-[3-3]	Choice	3-2	3-3
op-[3-3-1]-[3-3-2]	EnablingInfo	3-3-1	3-3-2
op-[3-3-2]-[3-3-3]	EnablingInfo	3-3-2	3-3-3
op-[3-3-3]-[3-3-4]	EnablingInfo	3-3-3	3-3-4

Figure 4.13: Temporal Operator Prediction Example

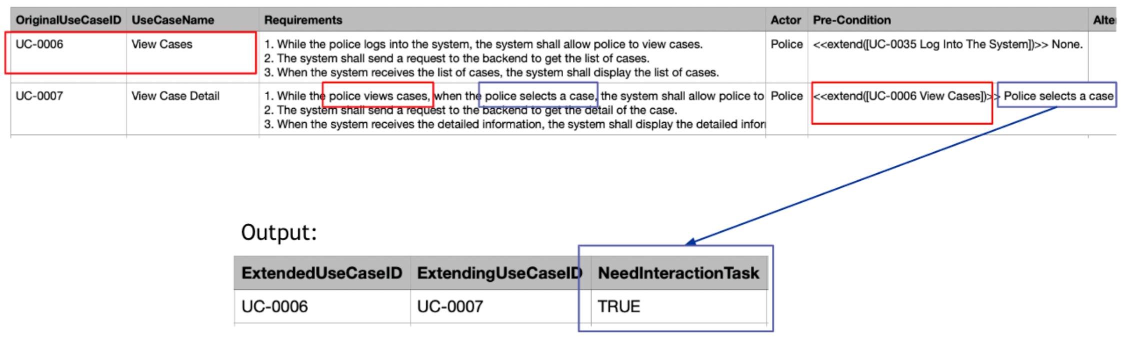
4.4.7 Step7：Extend Relationship Use Cases Generation

Input: Use Case Diagram.

Output: Use cases for extended relationships.

Description: Capture all use cases that include a precondition and match them. For use cases that have actions in the preconditions, set the NeedInteractionTask attribute to true. See the blue box in (Figure 4.14).

Example：



OriginalUseCaseID	UseCaseName	Requirements	Actor	Pre-Condition	Alte
UC-0006	View Cases	1. While the police logs into the system, the system shall allow police to view cases. 2. The system shall send a request to the backend to get the list of cases. 3. When the system receives the list of cases, the system shall display the list of cases.	Police	<<extend([UC-0035 Log Into The System])>> None.	
UC-0007	View Case Detail	1. While the police views cases, when the police selects a case, the system shall allow police to 2. The system shall send a request to the backend to get the detail of the case. 3. When the system receives the detailed information, the system shall display the detailed infor	Police	<<extend([UC-0006 View Cases])>> Police selects a case	

Output:

ExtendedUseCaseID	ExtendingUseCaseID	NeedInteractionTask
UC-0006	UC-0007	TRUE

Figure 4.14: Extend Relationship Use Cases Generation Example

doi:10.6342/NTU202404066

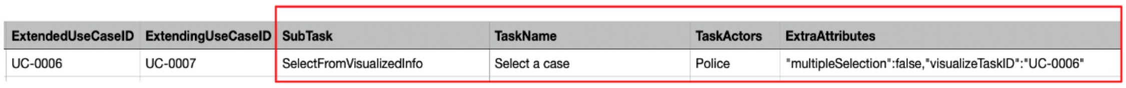
4.4.8 Step8：Interaction Tasks Prediction

Input: Extended Use Case ID, Extending Use Case ID, Use Case Specification, Use Case Diagram.

Output: Interaction Task information, including task name, type, actor, and extra attributes.

Description: Use the trained model to predict all Interaction Tasks in the project. As shown in (Figure 4.15), it primarily describes the task type, actor, and extra attributes of the ”Select a case” task (Selecting a case from the UC-0006 use case).

Example：



ExtendedUseCaseID	ExtendingUseCaseID	SubTask	TaskName	TaskActors	ExtraAttributes
UC-0006	UC-0007	SelectFromVisualizedInfo	Select a case	Police	"multipleSelection":false,"visualizeTaskID":"UC-0006"

Figure 4.15: Interaction Task Prediction Example

4.4.9 Step9：Complete Use Case CTT and Main CTT Generation

Input: All the previous output information.

Output: Complete Task Models (Use Case CTT and Main CTT).

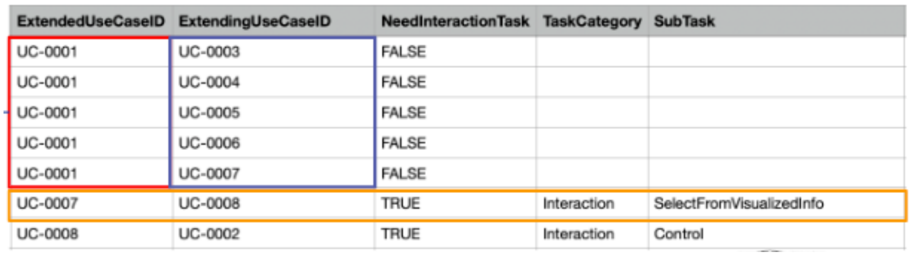
Description: Since the Main CTT only contains information and has not yet been converted into a tree structure, and it does not include the temporal operators between use cases, a program needs to be used to convert all the information (as shown in Figure 4.16) into the complete CTT format (as shown in Figure 4.17).

For example, it can be observed that all use cases extending from UC-0001 are placed

doi:10.6342/NTU202404066

under the ”Task enabling by UC-01” node in (Figure 4.17). Additionally, the Interaction Task between UC-0007 and UC-0008 is displayed on the left of the UC-08 in (Figure 4.17), indicating that the action ”Select a Transmission” must be performed to trigger the UC-08 use case.

Example：



ExtendedUseCaseID	ExtendingUseCaseID	NeedInteractionTask	TaskCategory	SubTask
UC-0001	UC-0003	FALSE		
UC-0001	UC-0004	FALSE		
UC-0001	UC-0005	FALSE		
UC-0001	UC-0006	FALSE		
UC-0001	UC-0007	FALSE		
UC-0007	UC-0008	TRUE	Interaction	SelectFromVisualizedInfo
UC-0008	UC-0002	TRUE	Interaction	Control

Figure 4.16: Interaction Tasks Example

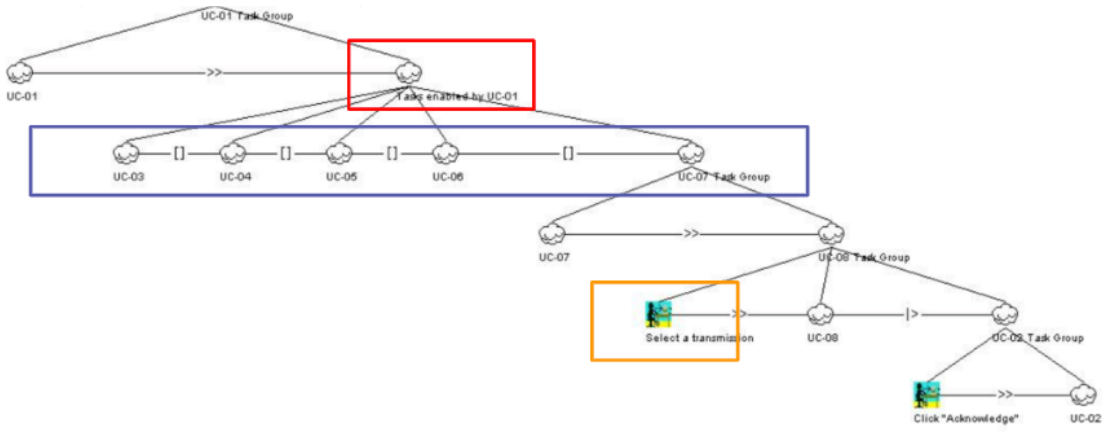


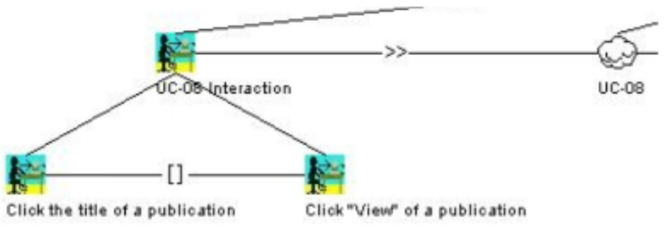
Figure 4.17: Main CTT Generation Example

The information from the Interaction Tasks needs to be converted into the Task Model (Main CTT) structure. To achieve this, the Chain of Responsibility (COR) pattern is used to write a program that generates the Task Model, with four handlers: actor, description, task, and temporal operator. The complete class diagram is shown in (Figure A.1)。

- **Actor**：This part handles all the actors in each use case, used for identity verification.

doi:10.6342/NTU202404066

- **Description**：This part handles descriptive information such as project ID, use case name, etc.
- **Task**：This part manages the entire tree structure, including the extend relationships between use cases and whether they include Interaction Tasks.
- **Temporal Operator**：This part handles the temporal relationships between nodes. Rules are used here to generate temporal operators, with the rules as follows:
  - The right side of Interaction is "Enabling".
  - Underneath "Enabling" are all choices.
  - The link between "Include Task" and ("Enable task" or "Task Group" or " Extending Task"), check if these tasks have interaction.
    - If yes: Disabling.
    - If no: Enabling.
  - Special Case(Figure 4.18): The link under the Interaction Task Group falls under Case 1, but it should be "Choice"



UC-08 Interaction
>>
UC-08
[]
Click the title of a publication
Click "View" of a publication

Figure 4.18: Special Case of Main CTT Temporal Operator

doi:10.6342/NTU202404066

Chapter 5 Influence Function

5.1 The Introduction of Influence Function

The influence function identifies data points that have a significant effect on the estimation of model parameters, which helps to understand the sensitivity of the model to different data points. For a particular test data point, the influence function calculates the influence values for all training data points and identifies both harmful and helpful data points. Applications of the influence function include understanding model behavior, correcting mislabeled samples, and improving the quality of training data.

The formula for the Influence Function is as follows:

$$I_{up,loss}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}; \hat{\theta})^T H_\theta^{-1} \nabla_\theta L(z; \hat{\theta})$$

where,

- z_{test} ：Test data point
- L ：The gradient of the loss function with respect to model parameters
- z ：Training data point
- H ：Hessian Matrix

doi:10.6342/NTU202404066

This formula represents the inner product value, indicating the influence of the training data point z on the test data point z_{test}. The original code ([1]])calculates the influence values for image classification; in this study, it has been modified to **calculate the influence scores for text classification**, thus identifying the helpful and harmful training data points for specific test data points.

5.2 Sequence Diagram

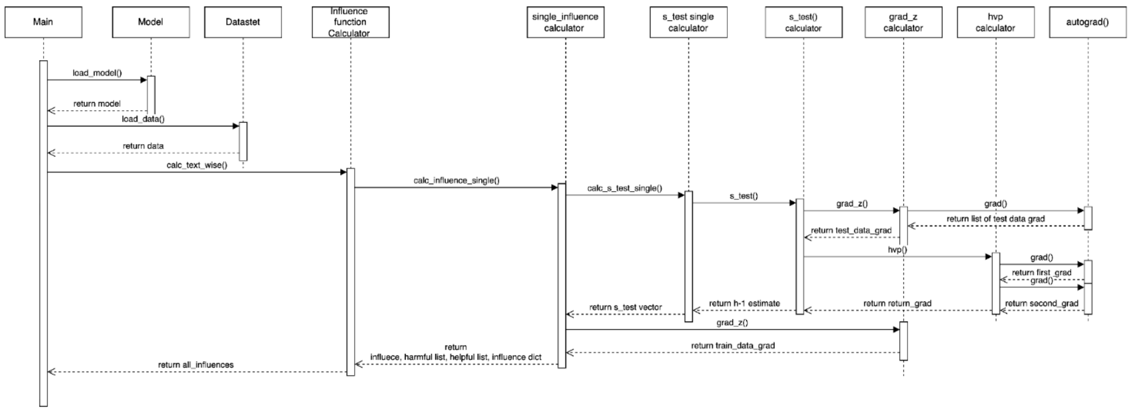


Figure 5.1: Sequence Diagram of Influence function

（Figure5.1) shows the sequence diagram of the influence function. To calculate the influence scores, you first need to input the trained model, as well as the training and test datasets, and then call the `calc_text_wise()` function.The `calc_influence_single()` function is responsible for calculating the influence of a single text, `calc_s_test_single()` calculates the s_{test} vector for a single test sample. The calculation of s_{test} corresponds to the $H_{\theta}^{-1}\nabla_{\theta}L(z_{\text{test}};\hat{\theta})$ part in the formula. Finally, the s_{test} vector is dotted with the gradients of the training data ($\nabla_{\theta}L(z;\hat{\theta})$).

doi:10.6342/NTU202404066

5.3 Cuda Out of Memory Issue

When writing a calculation of the influence function of a text, I encountered the problem ”Cuda Out of Memory”. This is because when calculating the Hessian matrix, the gradient of the loss function is calculated twice and the result of the first calculation is retained in the second calculation. Compared to the source code of the CNN model, the T5 model has larger and more parameters, resulting in excessive memory usage. In addition, the `grad_z()` function saves the experimental results, which increases memory usage.

In order to solve this problem, this study follows the DeepSpeed [7] method, which offloads GPU tasks to the CPU during the training process. However, since the tensors must be located on the same device during the computation, and there are two GPUs in the lab, these tensors need to be allocated properly.

In (Figure 5.2), the CUDA out of memory problem is solved by offloading the red section to the CPU and using another GPU for the blue section.

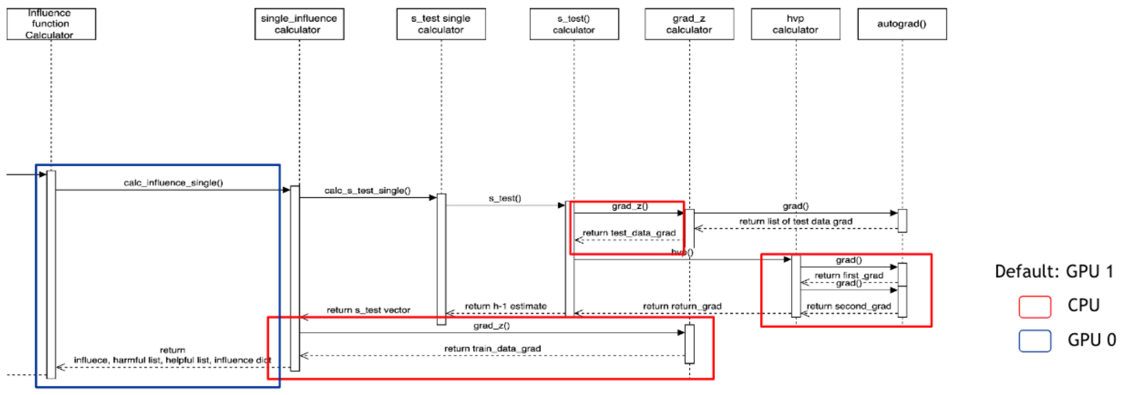


Figure 5.2: Cuda out of Memory Solution

doi:10.6342/NTU202404066

5.4 Restraint

The goal of this study is to identify harmful training data points using influence functions. By removing these harmful data points, it aims to improve the accuracy of the model. However, the study [2] pointed out that the gap between the actual results and the expectations after removing the harmful data could be caused by several reasons:

- Differences in Model Parameters: Different model parameters can lead to varying results, even when the same harmful data points are removed. This is because each model's initialization and training process affect the final parameter values.
- Damping Error: To avoid NaN issues, damping added when calculating the inverse of the Hessian matrix may cause inaccuracies.
- Difference between Unconverged and Converged Models: Assuming that the model is converged, but it may not be fully converged actually, which affects the results.
- Error in Non-Linear Models: The influence function is based on local linearization, which can lead to errors when applied to non-linear models.
- Errors in Iterative Methods: Iterative methods can cause errors if there are not enough iterations.

5.5 Experiment

For the Task Type prediction task, we use the influence function to calculate the scores for each training data point. The influence function code provides two methods for selecting test data:

doi:10.6342/NTU202404066

1. Randomly select n data.

2. Select one data for each label.

First, the influence scores for each training data point are calculated for every test data point. These scores are then averaged across all test data points to obtain a final influence score for each training data point. The final score represents the data quality ranking. This study then uses these rankings to select subsets of data. Specifically, it compares the training accuracy using the top 20%, 50%, and 80% of the data points (based on the influence scores) against the accuracy using the bottom 20%, 50%, and 80% of the data points. This comparison helps to evaluate the impact of the highest and lowest quality data points on training performance.

Additionally, this study tested the influence function's ability to correct mislabeled samples by comparing the training results using the top 95% of the best data points with those using all the data. It's important to note that the remaining 5% of data points are not necessarily incorrect, but they have less impact on the model. Thus, removing them may potentially improve the overall accuracy of the model.

5.5.1 Accuracy Comparison Between Good and Bad Data Across Different Intervals

The parameter settings for this experiment are as follows: `scale` is set to 25 (default value), `damping` is set to 0.01 (default value), `recursion_depth` is set to 100, and `r` is set to 5 (number of repetitions for calculating `s_test`). For `k`, it is set to 20%, 50%, and 80%, and the training results of the top 20% are compared with the bottom 20%, the top 50% with the bottom 50%, and the top 80% with the bottom 80%, respectively.

doi:10.6342/NTU202404066

Dataset \k	20%	50%	80%
best	**0.923**	**0.978**	**0.986**
worst	0.909	0.955	0.982

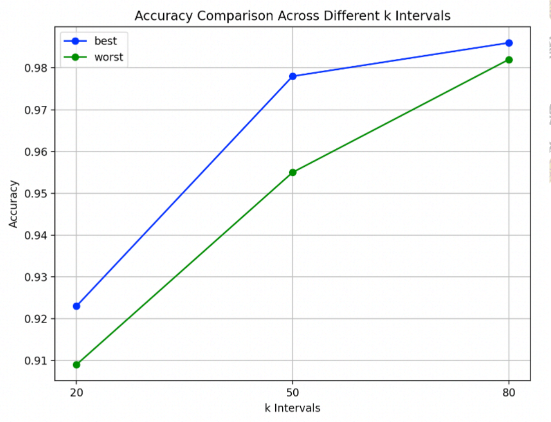


Figure 5.3: Influence Function Results with Randomly Selected Test Data

First, five test samples were randomly selected, and the influence scores for all corresponding training data were calculated. The average of these scores was taken as the final score for each training sample. The experimental results (Figure 5.3) show that, across all intervals, the training accuracy of the best k% data is higher than that of the worst k% data.

Next, the second method for selecting test data was used. One sample from each category was chosen as the test data, and their average was taken as the final score.

doi:10.6342/NTU202404066

Dataset \k	20%	50%	80%
best	**0.867**	0.989	**0.996**
worst	0.839	**0.992**	0.988

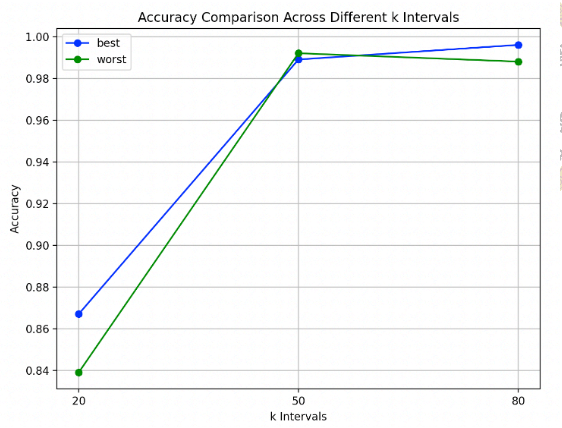


Figure 5.4: Accuracy Comparison Across Different k Intervals Using One Sample per Class as Test Data

In (Figure5.4), for k=50%, it can be observed that the accuracy for the worst data is higher than that for the best data. This may be due to the errors mentioned earlier. According to previous studies, using the right weight decay can fix these errors. So, this experiment used the recommended `weight_decay=0.001` to retrain and recalculate the influence scores. The results after recalculating are as follows:

Dataset \k	20%	50%	80%
best	**0.902**	**0.989**	**0.993**
worst	0.874	0.978	0.989

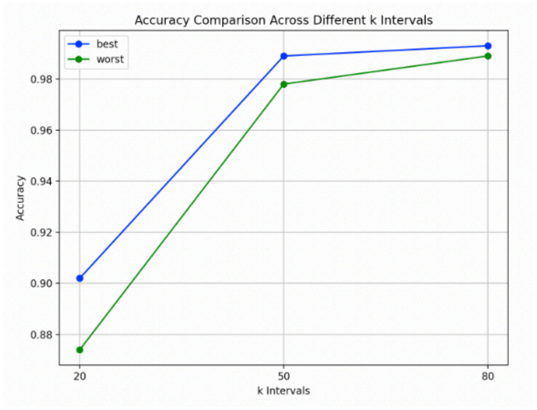


Figure 5.5: Effect of Weight Decay (0.001) on Accuracy Across Different k Intervals

(Figure 5.5) shows that, in all three intervals, the accuracy of the best k% data is better than that of the worst k% data.

doi:10.6342/NTU202404066

5.5.2 Retraining the Model after Removing Low-Scoring Data

Next, the study used two methods to select test data for calculating influence scores. The best 95% of the data was used to train the model to check if its performance is better than training with the original dataset. The experimental results are as follows: (as shown in Table 5.1), removing the least helpful 5% of the data based on influence scores can improve model performance.

Table 5.1: Comparison of Training Results Using the Best 95% and Full Dataset

Methods for Getting Test Data \k	**95%**	**100%**
Randomly select n samples	**0.996**	0.992
Select one sample for each label	**0.997**	0.992

Through experimentation, it was found that in 8 out of 9 intervals under different conditions, the results were consistent with the original expectations. This indicates that the influence function plays a significant role in selecting training datasets for text. According to the second experiment, removing a small portion of the data that negatively impacts training can improve training accuracy.

doi:10.6342/NTU202404066

Chapter 6 Conclusion and Future Work

6.1 Conclusion

To summarize this study, the training dataset for the task model was reconstructed, with a new format designed for tasks in alternative and exception flows. Additionally, five models were trained using the T5-base model to generate task model information, and an automated process was designed to generate complete Task Models (Use Case CTT and Main CTT) from use case specifications. Finally, the accuracy of the model was improved by identifying and removing harmful datasets using influence functions.

Despite the progress made, several issues still need to be resolved:

1. The accuracy of task generation needs to be improved. The model predicts incorrect text content when dealing with uncommon alternative flows.
2. Task ID predictions are sometimes incorrect. This is important because subsequent tasks use these IDs as references (e.g., for temporal operators link ids).
3. The validation of the influence function is not thorough enough. Due to the time-consuming nature of retraining, cross-validation has not yet been used.

doi:10.6342/NTU202404066

6.2 Future Work

1. Find methods to improve the accuracy of task generation, especially for the rare alternative flows and task ID predictions.

2. Explore more effective methods to validate the influence function's effectiveness, potentially finding time-saving alternatives to cross-validation.

3. Use influence functions to select datasets for other classification tasks, thereby improving the performance and accuracy of the model.

doi:10.6342/NTU202404066

References

[1] N. Arora. pytorch_influence_functions, 2019.

[2] J. Bae, N. Ng, A. Lo, M. Ghassemi, and R. Grosse. If influence functions are the answer, then what is the question? arXiv preprint arXiv:2209.05364, 2022.

[3] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In Proceedings of the 34th International Conference on Machine Learning, volume 70, pages 1885–1894, 2017.

[4] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. arXiv preprint arXiv:1708.02002, 2017.

[5] R.-S. Liou. Auto build user interface from task model. Master's thesis, National Taiwan University, Taipei, Taiwan, 2023.

[6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683, 2019.

[7] D. Team. Deepspeed configuration json documentation. https://www.deepspeed.ai/docs/config-json/, 2024.

doi:10.6342/NTU202404066

Appendix A — Class Diagram

A.1 Class Diagram

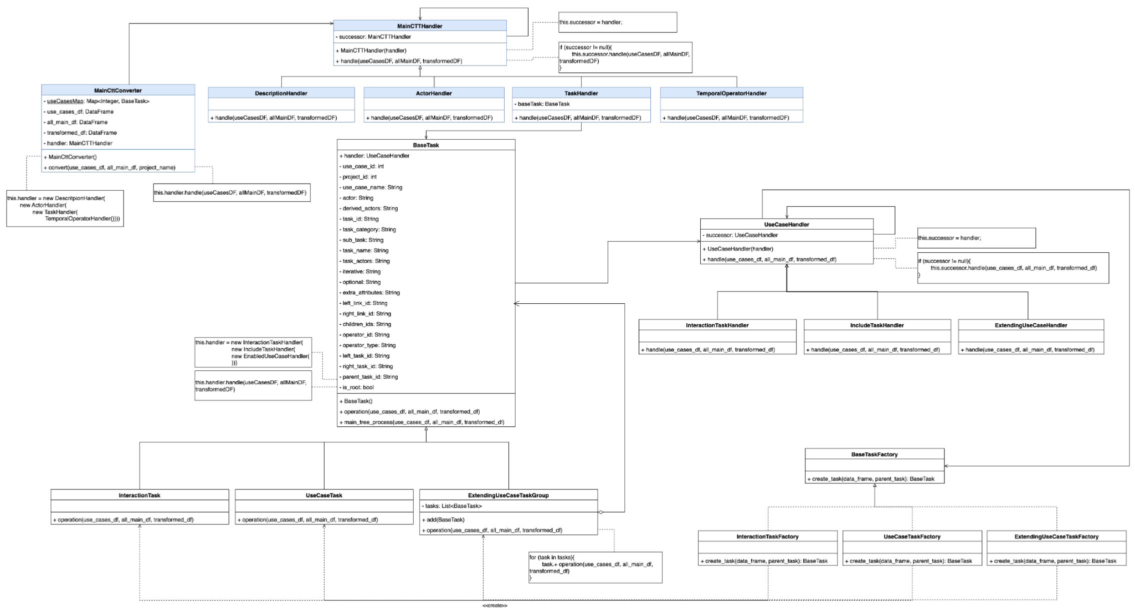

MainCTTHandler
- successor: MainCTTHandler
+ MainCTTHandler(handler)
+ handle(useCasesDF, allMainDF, transformedDF)
this.successor = handler;
if (successor != null){
this.successor.handle(useCasesDF, allMainDF, transformedDF)
}
MainCttConverter
- useCasesMap: Map<Integer, BaseTask>
- use_cases_df: DataFrame
- all_main_df: DataFrame
- transformed_df: DataFrame
- handler: MainCTTHandler
+ MainCttConverter()
+ convert(use_cases_df, all_main_df, project_name)
this.handler = new DescritpionHandler(
new ActorHandler(
new TaskHandler(
TemporalOperatorHandler())))
this.handler.handle(useCasesDF, allMainDF, transformedDF)
DescriptionHandler
+ handle(useCasesDF, allMainDF, transformedDF)
ActorHandler
+ handle(useCasesDF, allMainDF, transformedDF)
TaskHandler
- baseTask: BaseTask
+ handle(useCasesDF, allMainDF, transformedDF)
TemporalOperatorHandler
+ handle(useCasesDF, allMainDF, transformedDF)
BaseTask
+ handler: UseCaseHandler
- use_case_id: int
- project_id: int
- use_case_name: String
- actor: String
- derived_actors: String
- task_id: String
- task_category: String
- sub_task: String
- task_name: String
- task_actors: String
- iterative: String
- optional: String
- extra_attributes: String
- left_link_id: String
- right_link_id: String
- children_ids: String
- operator_id: String
- operator_type: String
- left_task_id: String
- right_task_id: String
- parent_task_id: String
- is_root: bool
+ BaseTask()
+ operation(use_cases_df, all_main_df, transformed_df)
+ main_tree_process(use_cases_df, all_main_df, transformed_df)
this.handler = new InteractionTaskHandler(
new IncludeTaskHandler(
new EnabledUseCaseHandler(
)))
this.handler.handle(useCasesDF, allMainDF, transformedDF)
UseCaseHandler
- successor: UseCaseHandler
+ UseCaseHandler(handler)
+ handle(use_cases_df, all_main_df, transformed_df)
this.successor = handler;
if (successor != null){
this.successor.handle(use_cases_df, all_main_df, transformed_df)
}
InteractionTaskHandler
+ handle(use_cases_df, all_main_df, transformed_df)
IncludeTaskHandler
+ handle(use_cases_df, all_main_df, transformed_df)
ExtendingUseCaseHandler
+ handle(use_cases_df, all_main_df, transformed_df)
BaseTaskFactory
+ create_task(data_frame, parent_task): BaseTask
InteractionTask
+ operation(use_cases_df, all_main_df, transformed_df)
UseCaseTask
+ operation(use_cases_df, all_main_df, transformed_df)
ExtendingUseCaseTaskGroup
- tasks: List<BaseTask>
+ add(BaseTask)
+ operation(use_cases_df, all_main_df, transformed_df)
for (task in tasks){
task.+ operation(use_cases_df, all_main_df, transformed_df)
}
InteractionTaskFactory
+ create_task(data_frame, parent_task): BaseTask
UseCaseTaskFactory
+ create_task(data_frame, parent_task): BaseTask
ExtendingUseCaseTaskFactory
+ create_task(data_frame, parent_task): BaseTask
<<create>>

Figure A.1: Class Diagram of the Main CTT Converter

doi:10.6342/NTU202404066

Appendix B — Case Study

B.1 Inventory System

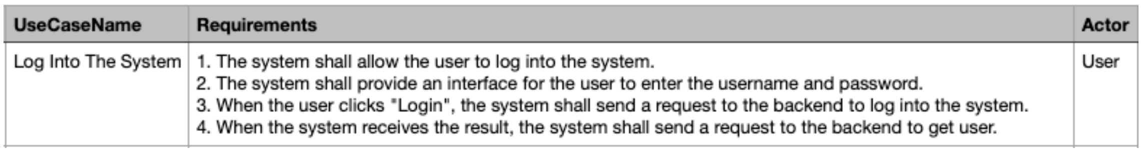
This case study explores how to automatically generate Task Models from Use Case Specifications, using the Inventory System as an example. It highlights 11 important use cases, each including the following:

Input：Requirements、Use Case Specification

Output：Task Models and the corresponding Use Case CTT figures

Some use cases also include descriptions of preconditions, which are used to generate the final Main CTT.

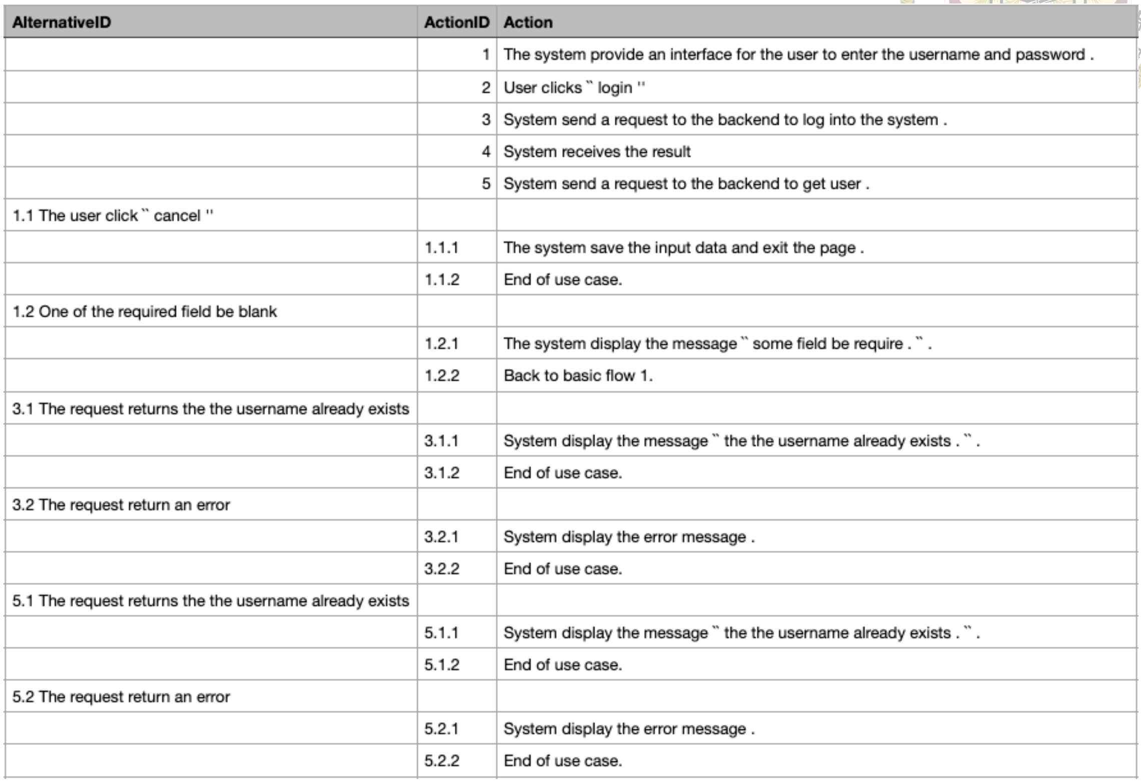
B.1.1 Use Case：[Log into the system]



UseCaseName	Requirements	Actor
Log Into The System	1. The system shall allow the user to log into the system. 2. The system shall provide an interface for the user to enter the username and password. 3. When the user clicks "Login", the system shall send a request to the backend to log into the system. 4. When the system receives the result, the system shall send a request to the backend to get user.	User

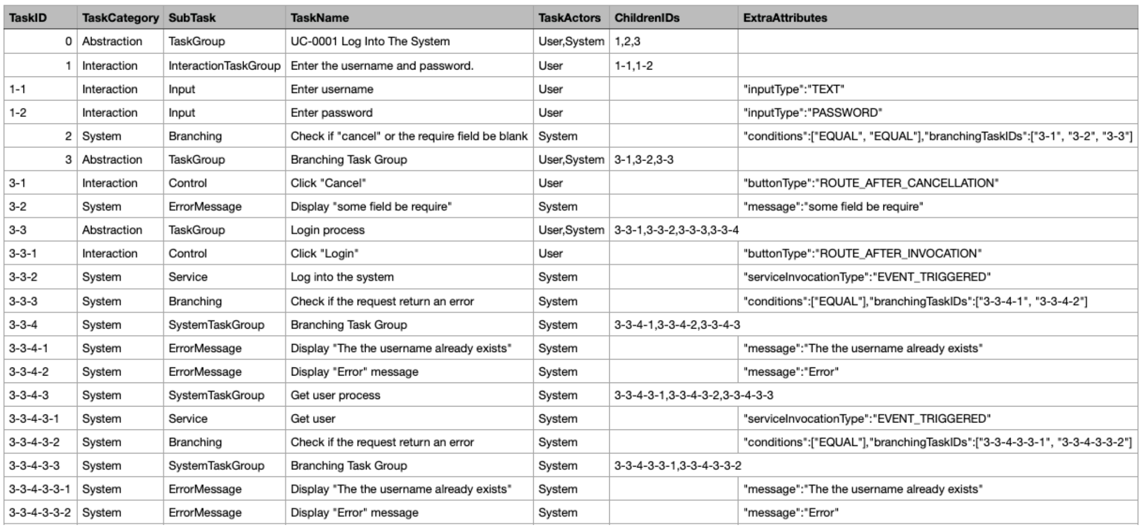
Figure B.2: Requirement

doi:10.6342/NTU202404066



AlternativeID	ActionID	Action
	1	The system provide an interface for the user to enter the username and password .
	2	User clicks ‵ login ''
	3	System send a request to the backend to log into the system .
	4	System receives the result
	5	System send a request to the backend to get user .
1.1 The user click ‵ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‵ some field be require . ‵ .
	1.2.2	Back to basic flow 1.
3.1 The request returns the the username already exists		
	3.1.1	System display the message ‵ the the username already exists . ‵ .
	3.1.2	End of use case.
3.2 The request return an error		
	3.2.1	System display the error message .
	3.2.2	End of use case.
5.1 The request returns the the username already exists		
	5.1.1	System display the message ‵ the the username already exists . ‵ .
	5.1.2	End of use case.
5.2 The request return an error		
	5.2.1	System display the error message .
	5.2.2	End of use case.

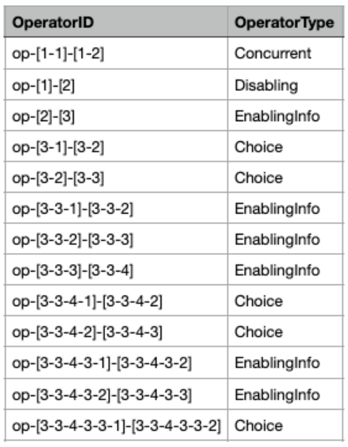
Figure B.3: Use Case Spec



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0001 Log Into The System	User,System	1,2,3	
1	Interaction	InteractionTaskGroup	Enter the username and password.	User	1-1,1-2	
1-1	Interaction	Input	Enter username	User		"inputType":"TEXT"
1-2	Interaction	Input	Enter password	User		"inputType":"PASSWORD"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	User,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	User		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Login process	User,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Login"	User		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Log into the system	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4-1", "3-3-4-2"]
3-3-4	System	SystemTaskGroup	Branching Task Group	System	3-3-4-1,3-3-4-2,3-3-4-3	
3-3-4-1	System	ErrorMessage	Display "The the username already exists"	System		"message":"The the username already exists"
3-3-4-2	System	ErrorMessage	Display "Error" message	System		"message":"Error"
3-3-4-3	System	SystemTaskGroup	Get user process	System	3-3-4-3-1,3-3-4-3-2,3-3-4-3-3	
3-3-4-3-1	System	Service	Get user	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-4-3-2	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4-3-3-1", "3-3-4-3-3-2"]
3-3-4-3-3	System	SystemTaskGroup	Branching Task Group	System	3-3-4-3-3-1,3-3-4-3-3-2	
3-3-4-3-3-1	System	ErrorMessage	Display "The the username already exists"	System		"message":"The the username already exists"
3-3-4-3-3-2	System	ErrorMessage	Display "Error" message	System		"message":"Error"

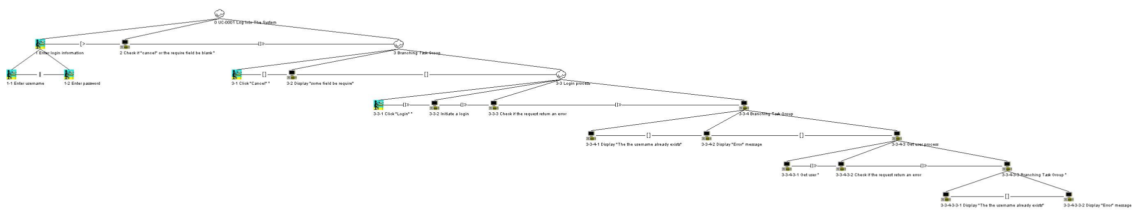
Figure B.4: Task Model - Task

doi:10.6342/NTU202404066



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo
op-[3-3-4-1]-[3-3-4-2]	Choice
op-[3-3-4-2]-[3-3-4-3]	Choice
op-[3-3-4-3-1]-[3-3-4-3-2]	EnablingInfo
op-[3-3-4-3-2]-[3-3-4-3-3]	EnablingInfo
op-[3-3-4-3-3-1]-[3-3-4-3-3-2]	Choice

Figure B.5: Task Model - Temporal Operator

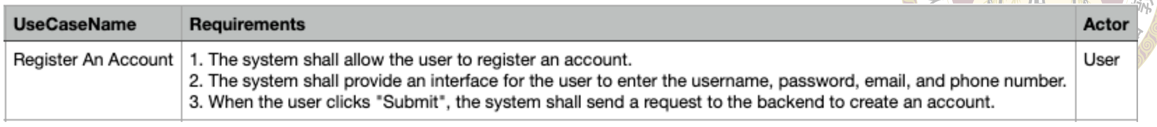


0 UC-0001 Log Into The System
1 Enter login information
[>
2 Check if "cancel" or the require field be blank "
[]»
3 Branching Task Group
1-1 Enter username
|||
1-2 Enter password
3-1 Click "Cancel" "
[]
3-2 Display "some field be require"
[]
3-3 Login process
3-3-1 Click "Login" "
[]»
3-3-2 Initiate a login
[]»
3-3-3 Check if the request return an error
[]»
3-3-4 Branching Task Group
3-3-4-1 Display "The the username already exists"
[]
3-3-4-2 Display "Error" message
[]
3-3-4-3 Get user process
3-3-4-3-1 Get user "
[]»
3-3-4-3-2 Check if the request return an error
[]»
3-3-4-3-3 Branching Task Group "
3-3-4-3-3-1 Display "The the username already exists"
[]
3-3-4-3-3-2 Display "Error" message

Figure B.6: CTT

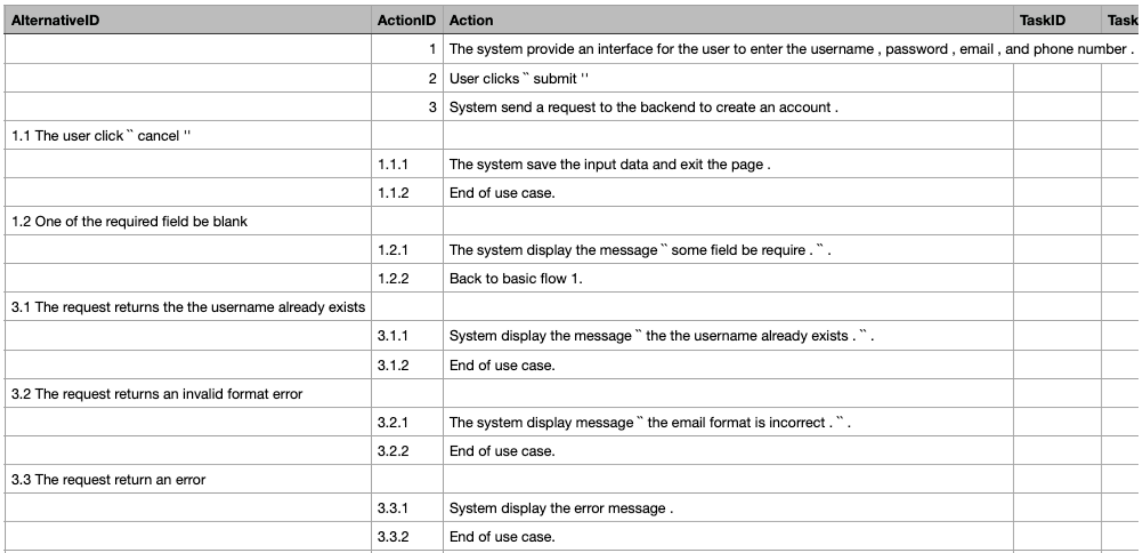
doi:10.6342/NTU202404066

B.1.2 Use Case：[Register an Account]



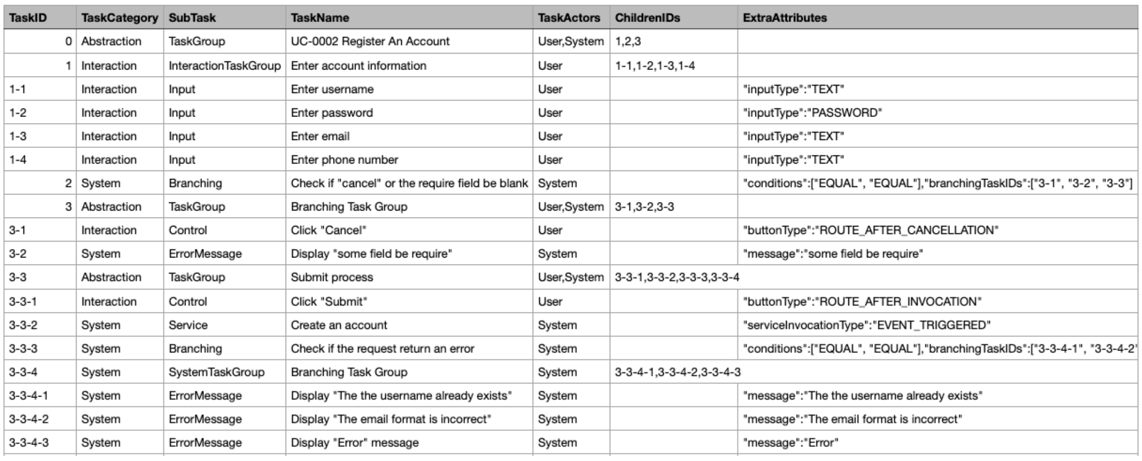
UseCaseName	Requirements	Actor
Register An Account	1. The system shall allow the user to register an account. 2. The system shall provide an interface for the user to enter the username, password, email, and phone number. 3. When the user clicks "Submit", the system shall send a request to the backend to create an account.	User

Figure B.7: Requirement



AlternativeID	ActionID	Action	TaskID	Task
	1	The system provide an interface for the user to enter the username , password , email , and phone number .		
	2	User clicks ˋ submit ''		
	3	System send a request to the backend to create an account .		
1.1 The user click ˋ cancel ''				
	1.1.1	The system save the input data and exit the page .		
	1.1.2	End of use case.		
1.2 One of the required field be blank				
	1.2.1	The system display the message ˋ some field be require . '' .		
	1.2.2	Back to basic flow 1.		
3.1 The request returns the the username already exists				
	3.1.1	System display the message ˋ the the username already exists . '' .		
	3.1.2	End of use case.		
3.2 The request returns an invalid format error				
	3.2.1	The system display message ˋ the email format is incorrect . '' .		
	3.2.2	End of use case.		
3.3 The request return an error				
	3.3.1	System display the error message .		
	3.3.2	End of use case.		

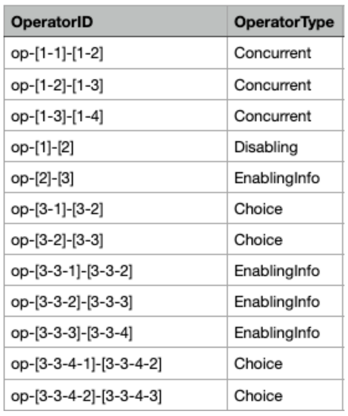
Figure B.8: Use Case Spec



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0002 Register An Account	User,System	1,2,3	
1	Interaction	InteractionTaskGroup	Enter account information	User	1-1,1-2,1-3,1-4	
1-1	Interaction	Input	Enter username	User		"inputType":"TEXT"
1-2	Interaction	Input	Enter password	User		"inputType":"PASSWORD"
1-3	Interaction	Input	Enter email	User		"inputType":"TEXT"
1-4	Interaction	Input	Enter phone number	User		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	User,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	User		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	User,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	User		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Create an account	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-3-4-1", "3-3-4-2"
3-3-4	System	SystemTaskGroup	Branching Task Group	System	3-3-4-1,3-3-4-2,3-3-4-3	
3-3-4-1	System	ErrorMessage	Display "The the username already exists"	System		"message":"The the username already exists"
3-3-4-2	System	ErrorMessage	Display "The email format is incorrect"	System		"message":"The email format is incorrect"
3-3-4-3	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.9: Task Model - Task

doi:10.6342/NTU202404066



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1-2]-[1-3]	Concurrent
op-[1-3]-[1-4]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo
op-[3-3-4-1]-[3-3-4-2]	Choice
op-[3-3-4-2]-[3-3-4-3]	Choice

Figure B.10: Task Model - Temporal Operator

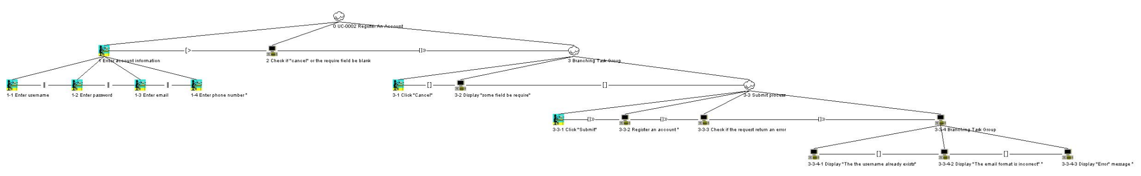
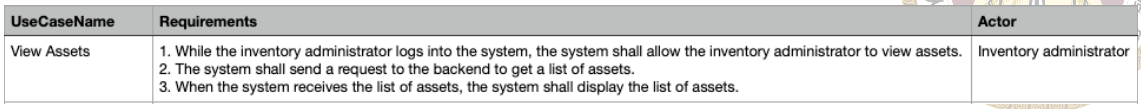

0 UC-0002 Register An Account
1 Enter account information
[>
2 Check if "cancel" or the require field be blank
[]≫
3 Branching Task Group
1-1 Enter username
||
1-2 Enter password
||
1-3 Enter email
||
1-4 Enter phone number *
3-1 Click "Cancel"
[]
3-2 Display "some field be require"
[]
3-3 Submit process
3-3-1 Click "Submit"
[]≫
3-3-2 Register an account *
[]≫
3-3-3 Check if the request return an error
[]≫
3-3-4 Branching Task Group
3-3-4-1 Display "The the username already exists"
[]
3-3-4-2 Display "The email format is incorrect" *
[]
3-3-4-3 Display "Error" message *

Figure B.11: CTT

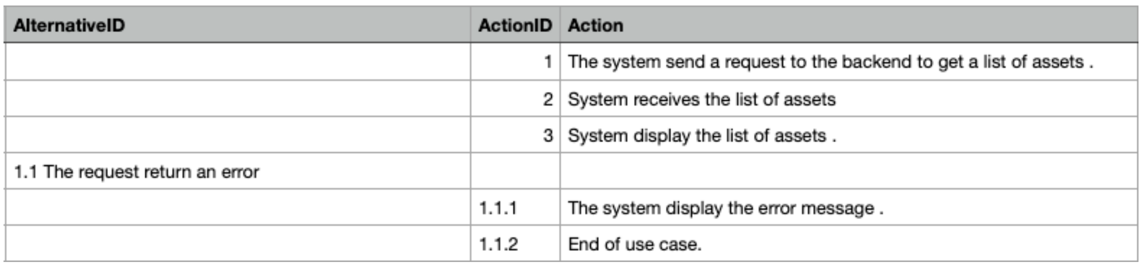
doi:10.6342/NTU202404066

B.1.3 Use Case：[View Assets]



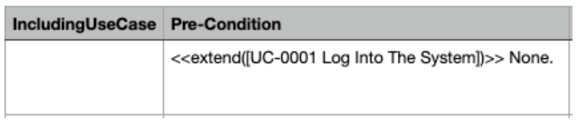
UseCaseName	Requirements	Actor
View Assets	1. While the inventory administrator logs into the system, the system shall allow the inventory administrator to view assets. 2. The system shall send a request to the backend to get a list of assets. 3. When the system receives the list of assets, the system shall display the list of assets.	Inventory administrator

Figure B.12: Requirement



AlternativeID	ActionID	Action
	1	The system send a request to the backend to get a list of assets .
	2	System receives the list of assets
	3	System display the list of assets .
1.1 The request return an error		
	1.1.1	The system display the error message .
	1.1.2	End of use case.

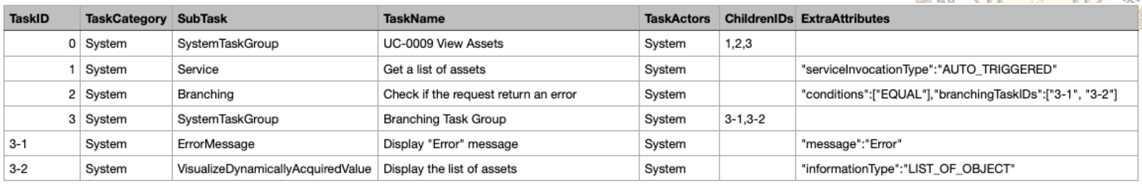
Figure B.13: Use Case Spec



IncludingUseCase	Pre-Condition
	<<extend([UC-0001 Log Into The System])>> None.

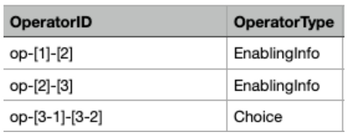
Figure B.14: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	System	SystemTaskGroup	UC-0009 View Assets	System	1,2,3	
1	System	Service	Get a list of assets	System		"serviceInvocationType":"AUTO_TRIGGERED"
2	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-1", "3-2"]
3	System	SystemTaskGroup	Branching Task Group	System	3-1,3-2	
3-1	System	ErrorMessage	Display "Error" message	System		"message":"Error"
3-2	System	VisualizeDynamicallyAcquiredValue	Display the list of assets	System		"informationType":"LIST_OF_OBJECT"

Figure B.15: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	EnablingInfo
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice

Figure B.16: Task Model - Temporal Operator

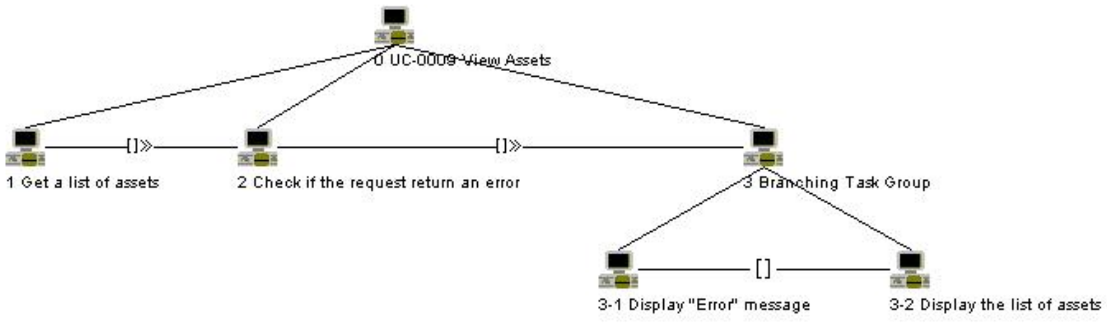
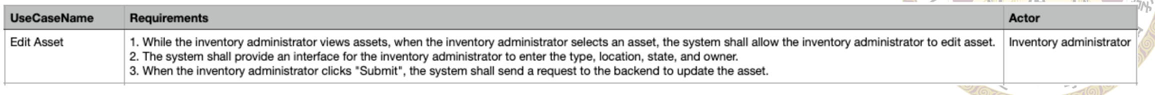

0 UC-0009 View Assets
1 Get a list of assets
[]≫
2 Check if the request return an error
[]≫
3 Branching Task Group
3-1 Display "Error" message
[]
3-2 Display the list of assets

Figure B.17: CTT

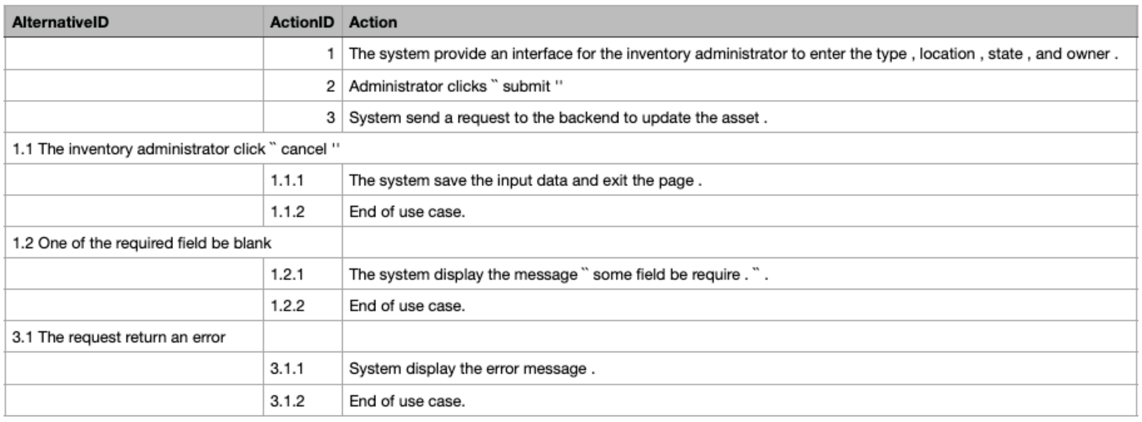
doi:10.6342/NTU202404066

B.1.4 Use Case：[Edit Asset]



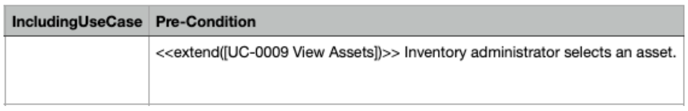
UseCaseName	Requirements	Actor
Edit Asset	1. While the inventory administrator views assets, when the inventory administrator selects an asset, the system shall allow the inventory administrator to edit asset. 2. The system shall provide an interface for the inventory administrator to enter the type, location, state, and owner. 3. When the inventory administrator clicks "Submit", the system shall send a request to the backend to update the asset.	Inventory administrator

Figure B.18: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the inventory administrator to enter the type , location , state , and owner .
	2	Administrator clicks ‶ submit ''
	3	System send a request to the backend to update the asset .
1.1 The inventory administrator click ‶ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‶ some field be require . ‶ .
	1.2.2	End of use case.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

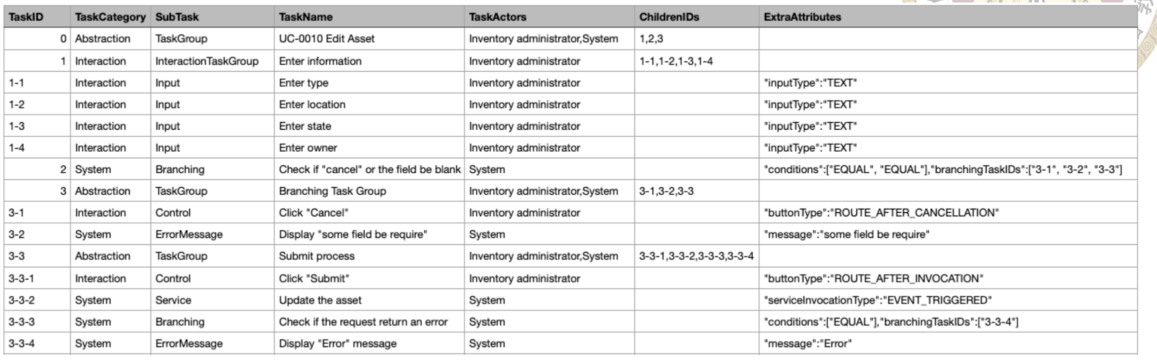
Figure B.19: Use Case Spec



IncludingUseCase	Pre-Condition
	<<extend([UC-0009 View Assets])>> Inventory administrator selects an asset.

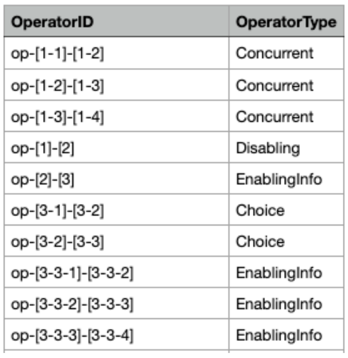
Figure B.20: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0010 Edit Asset	Inventory administrator,System	1,2,3	
1	Interaction	InteractionTaskGroup	Enter information	Inventory administrator	1-1,1-2,1-3,1-4	
1-1	Interaction	Input	Enter type	Inventory administrator		"inputType":"TEXT"
1-2	Interaction	Input	Enter location	Inventory administrator		"inputType":"TEXT"
1-3	Interaction	Input	Enter state	Inventory administrator		"inputType":"TEXT"
1-4	Interaction	Input	Enter owner	Inventory administrator		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	Inventory administrator,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	Inventory administrator		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	Inventory administrator,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	Inventory administrator		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Update the asset	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.21: Task Model - Task



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1-2]-[1-3]	Concurrent
op-[1-3]-[1-4]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.22: Task Model - Temporal Operator

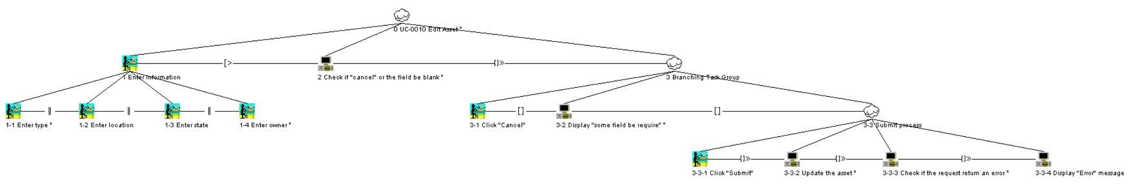
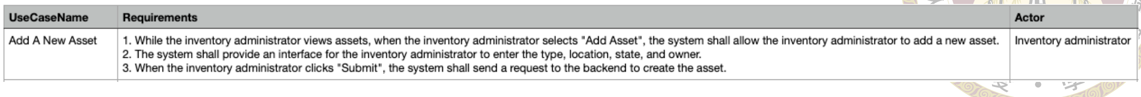


Figure B.23: CTT

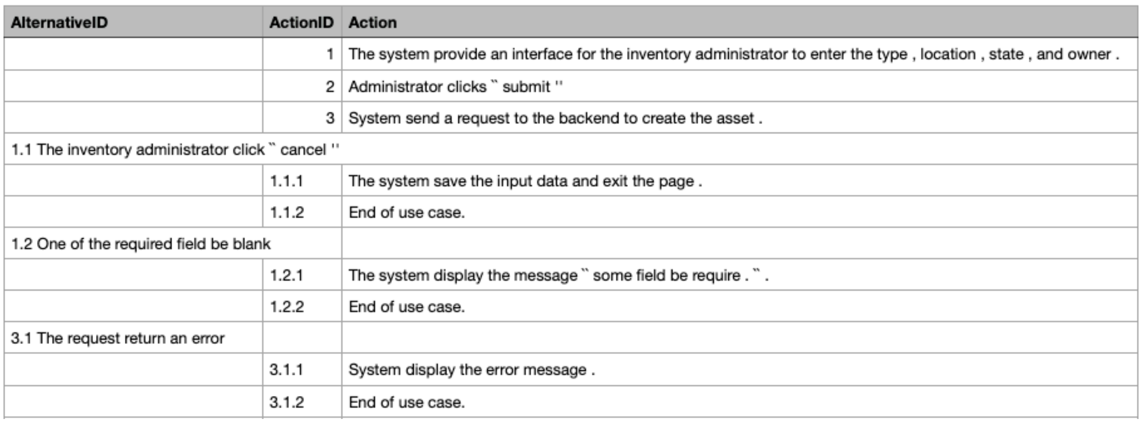
doi:10.6342/NTU202404066

B.1.5 Use Case：[Add a new asset]



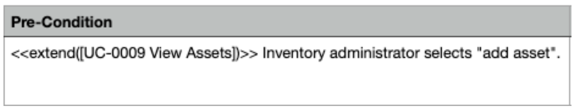
UseCaseName	Requirements	Actor
Add A New Asset	1. While the inventory administrator views assets, when the inventory administrator selects "Add Asset", the system shall allow the inventory administrator to add a new asset. 2. The system shall provide an interface for the inventory administrator to enter the type, location, state, and owner. 3. When the inventory administrator clicks "Submit", the system shall send a request to the backend to create the asset.	Inventory administrator

Figure B.24: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the inventory administrator to enter the type , location , state , and owner .
	2	Administrator clicks ‵‵ submit ''
	3	System send a request to the backend to create the asset .
1.1 The inventory administrator click ‵‵ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‵‵ some field be require . ‵‵ .
	1.2.2	End of use case.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

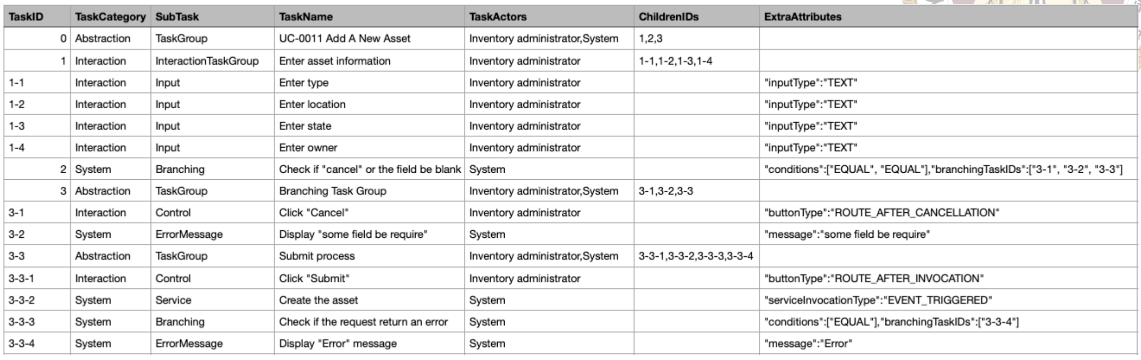
Figure B.25: Use Case Spec



Pre-Condition
<<extend([UC-0009 View Assets])>> Inventory administrator selects "add asset".

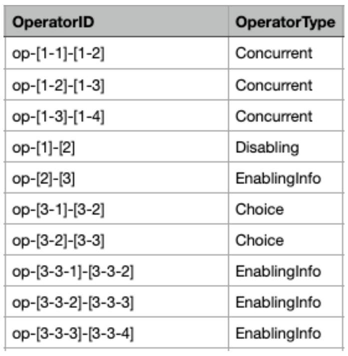
Figure B.26: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0011 Add A New Asset	Inventory administrator,System	1,2,3	
1	Interaction	InteractionTaskGroup	Enter asset information	Inventory administrator	1-1,1-2,1-3,1-4	
1-1	Interaction	Input	Enter type	Inventory administrator		"inputType":"TEXT"
1-2	Interaction	Input	Enter location	Inventory administrator		"inputType":"TEXT"
1-3	Interaction	Input	Enter state	Inventory administrator		"inputType":"TEXT"
1-4	Interaction	Input	Enter owner	Inventory administrator		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	Inventory administrator,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	Inventory administrator		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	Inventory administrator,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	Inventory administrator		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Create the asset	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.27: Task Model - Task



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1-2]-[1-3]	Concurrent
op-[1-3]-[1-4]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.28: Task Model - Temporal Operator

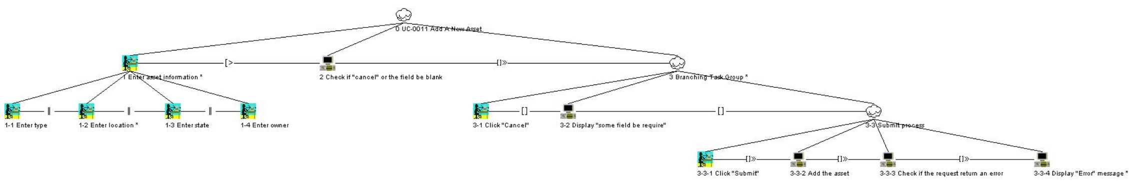
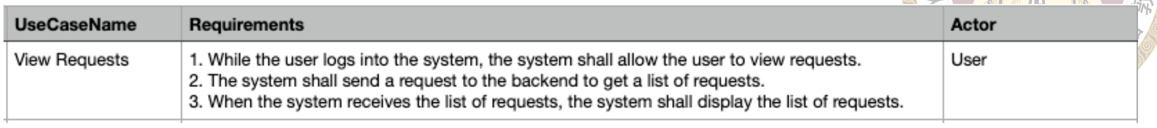

0 UC-0011 Add A New Asset
[>
[]»
1 Enter asset information *
2 Check if "cancel" or the field be blank
3 Branching Task Group *
|||
1-1 Enter type
1-2 Enter location *
1-3 Enter state
1-4 Enter owner
[]
3-1 Click "Cancel"
3-2 Display "some field be require"
3-3 Submit process
3-3-1 Click "Submit"
3-3-2 Add the asset
3-3-3 Check if the request return an error
3-3-4 Display "Error" message *

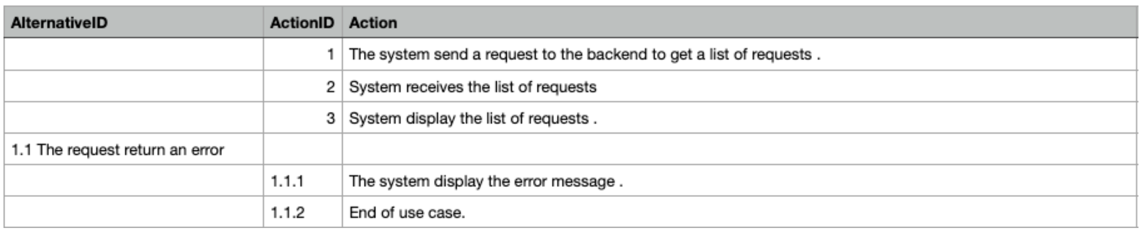
Figure B.29: CTT

B.1.6 Use Case：[View Requests]



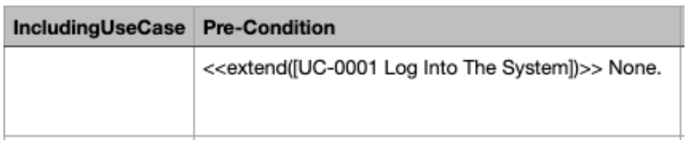
UseCaseName	Requirements	Actor
View Requests	1. While the user logs into the system, the system shall allow the user to view requests. 2. The system shall send a request to the backend to get a list of requests. 3. When the system receives the list of requests, the system shall display the list of requests.	User

Figure B.30: Requirement



AlternativeID	ActionID	Action
	1	The system send a request to the backend to get a list of requests .
	2	System receives the list of requests
	3	System display the list of requests .
1.1 The request return an error		
	1.1.1	The system display the error message .
	1.1.2	End of use case.

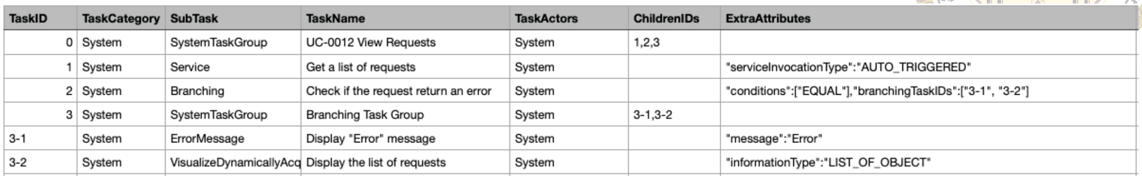
Figure B.31: Use Case Spec



IncludingUseCase	Pre-Condition
	<<extend([UC-0001 Log Into The System])>> None.

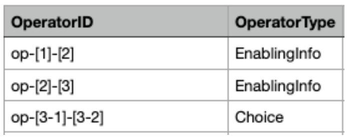
Figure B.32: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	System	SystemTaskGroup	UC-0012 View Requests	System	1,2,3	
1	System	Service	Get a list of requests	System		"serviceInvocationType":"AUTO_TRIGGERED"
2	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-1", "3-2"]
3	System	SystemTaskGroup	Branching Task Group	System	3-1,3-2	
3-1	System	ErrorMessage	Display "Error" message	System		"message":"Error"
3-2	System	VisualizeDynamicallyAcq	Display the list of requests	System		"informationType":"LIST_OF_OBJECT"

Figure B.33: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	EnablingInfo
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice

Figure B.34: Task Model - Temporal Operator

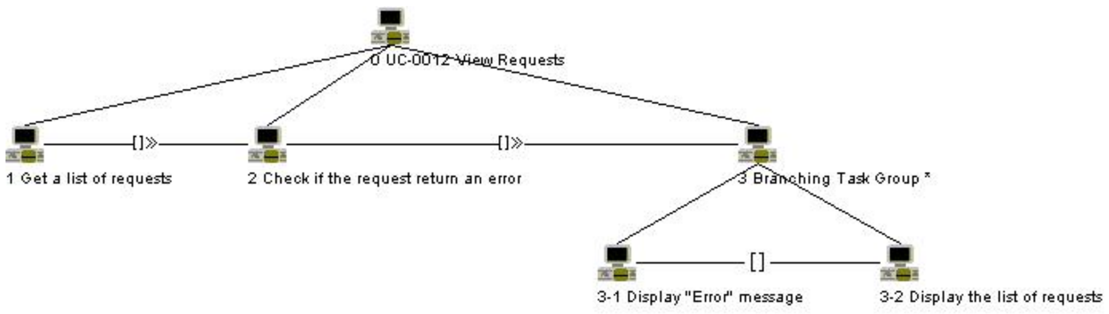
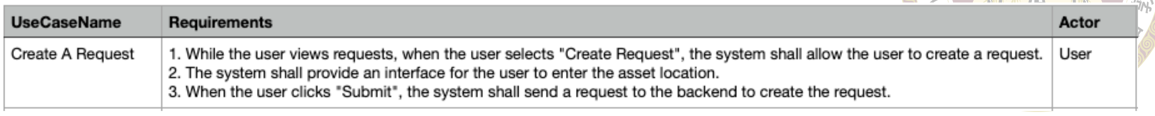

0 UC-0012 View Requests
1 Get a list of requests
[]»
2 Check if the request return an error
[]»
3 Branching Task Group *
3-1 Display "Error" message
[]
3-2 Display the list of requests

Figure B.35: CTT

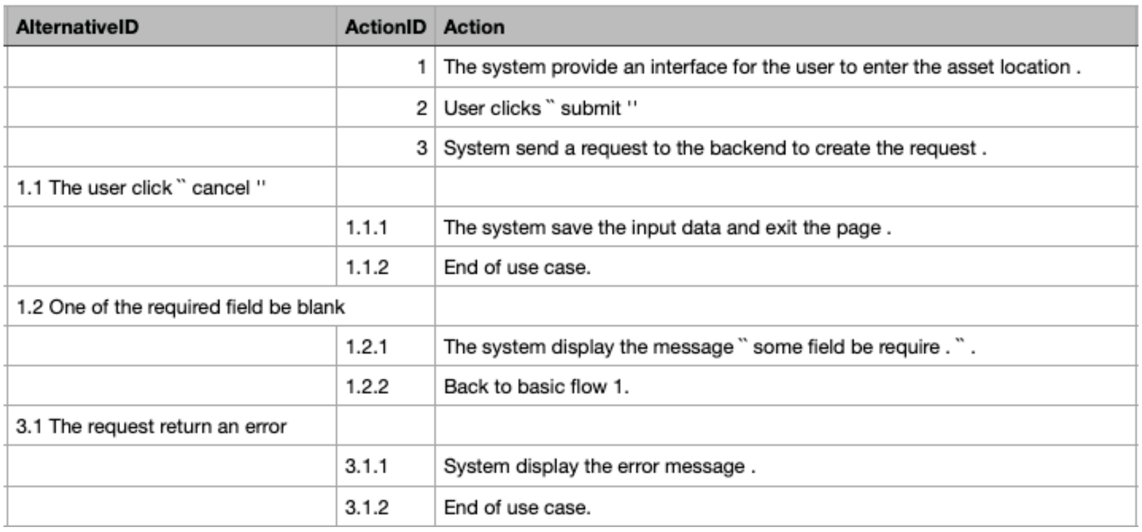
doi:10.6342/NTU202404066

B.1.7 Use Case：[Create a Request]



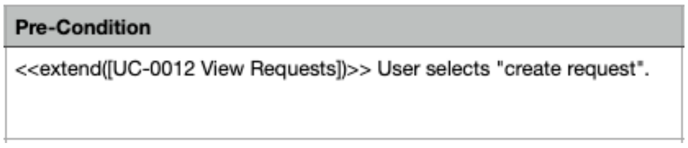
UseCaseName	Requirements	Actor
Create A Request	1. While the user views requests, when the user selects "Create Request", the system shall allow the user to create a request. 2. The system shall provide an interface for the user to enter the asset location. 3. When the user clicks "Submit", the system shall send a request to the backend to create the request.	User

Figure B.36: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the user to enter the asset location .
	2	User clicks ‵ submit ''
	3	System send a request to the backend to create the request .
1.1 The user click ‵ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‵ some field be require . ‶ .
	1.2.2	Back to basic flow 1.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

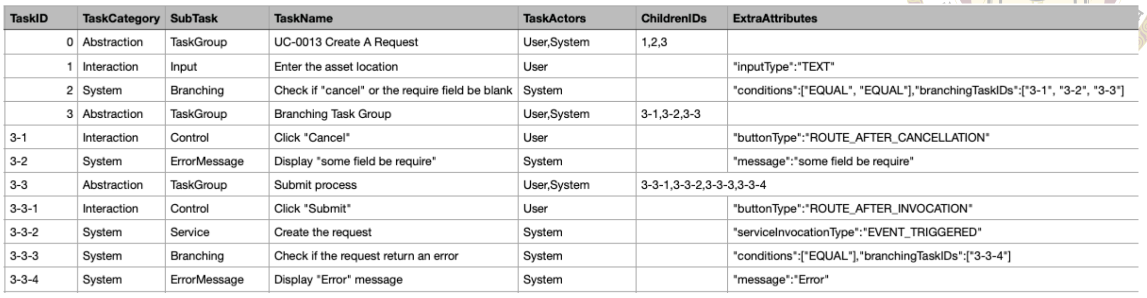
Figure B.37: Use Case Spec



Pre-Condition
<<extend([UC-0012 View Requests])>> User selects "create request".

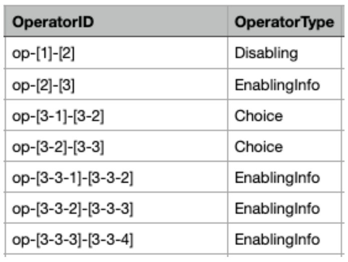
Figure B.38: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0013 Create A Request	User,System	1,2,3	
1	Interaction	Input	Enter the asset location	User		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	User,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	User		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	User,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	User		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Create the request	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.39: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.40: Task Model - Temporal Operator

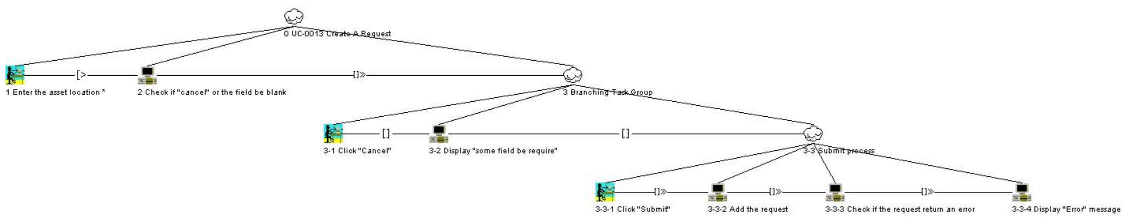
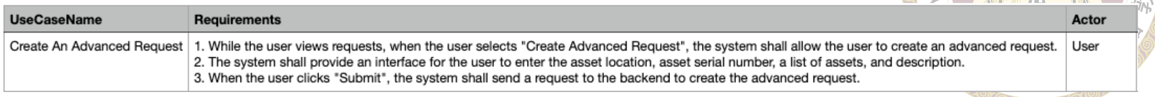

0 UC-0013 Create A Request
1 Enter the asset location *
[>
2 Check if "cancel" or the field be blank
[]»
3 Branching Task Group
3-1 Click "Cancel"
[]
3-2 Display "some field be require"
[]
3-3 Submit process
3-3-1 Click "Submit"
[]»
3-3-2 Add the request
[]»
3-3-3 Check if the request return an error
[]»
3-3-4 Display "Error" message

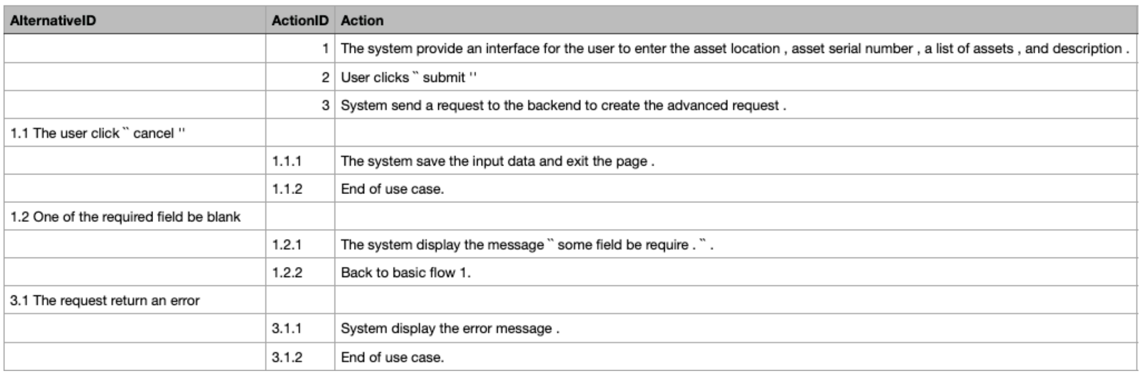
Figure B.41: CTT

B.1.8 Use Case：[Create an Advanced Request]



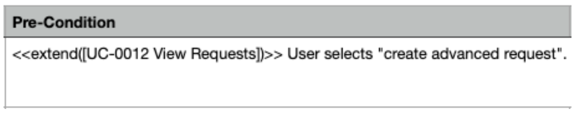
UseCaseName	Requirements	Actor
Create An Advanced Request	1. While the user views requests, when the user selects "Create Advanced Request", the system shall allow the user to create an advanced request. 2. The system shall provide an interface for the user to enter the asset location, asset serial number, a list of assets, and description. 3. When the user clicks "Submit", the system shall send a request to the backend to create the advanced request.	User

Figure B.42: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the user to enter the asset location , asset serial number , a list of assets , and description .
	2	User clicks ` submit ''
	3	System send a request to the backend to create the advanced request .
1.1 The user click ` cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ` some field be require . ` .
	1.2.2	Back to basic flow 1.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

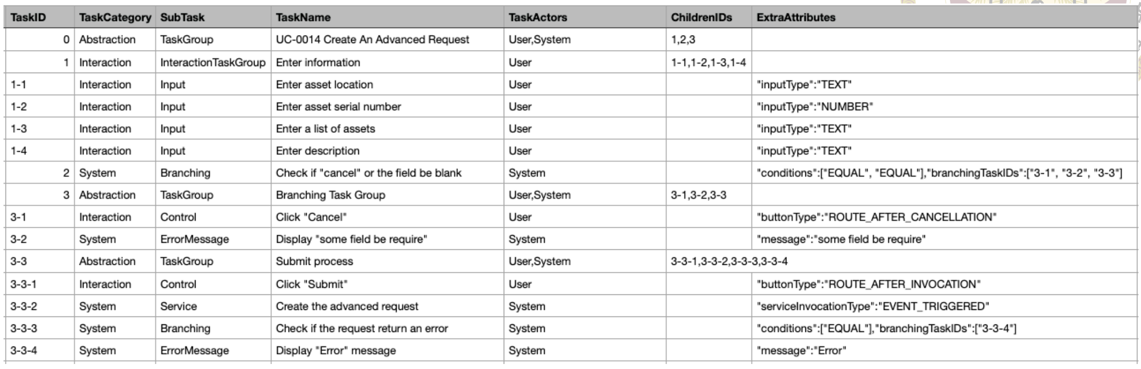
Figure B.43: Use Case Spec



Pre-Condition
<<extend([UC-0012 View Requests])>> User selects "create advanced request".

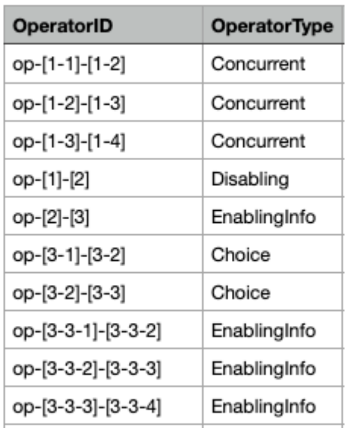
Figure B.44: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0014 Create An Advanced Request	User,System	1,2,3	
1	Interaction	InteractionTaskGroup	Enter information	User	1-1,1-2,1-3,1-4	
1-1	Interaction	Input	Enter asset location	User		"inputType":"TEXT"
1-2	Interaction	Input	Enter asset serial number	User		"inputType":"NUMBER"
1-3	Interaction	Input	Enter a list of assets	User		"inputType":"TEXT"
1-4	Interaction	Input	Enter description	User		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	User,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	User		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	User,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	User		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Create the advanced request	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.45: Task Model - Task



OperatorID	OperatorType
op-[1-1]-[1-2]	Concurrent
op-[1-2]-[1-3]	Concurrent
op-[1-3]-[1-4]	Concurrent
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.46: Task Model - Temporal Operator

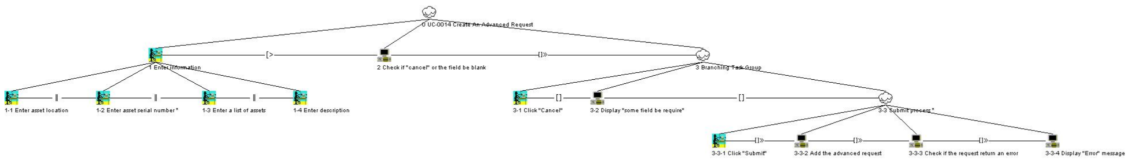
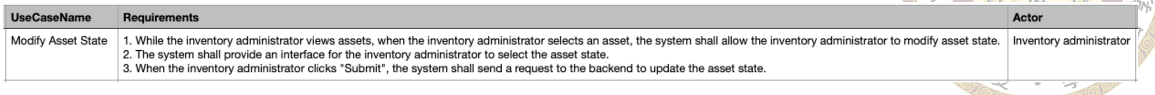

0 UC-0014 Create An Advanced Request
1 Enter information
[>
2 Check if "cancel" or the field be blank
[]>>
3 Branching Task Group
1-1 Enter asset location
|||
1-2 Enter asset serial number "
|||
1-3 Enter a list of assets
|||
1-4 Enter description
3-1 Click "Cancel"
[]
3-2 Display "some field be require"
[]
3-3 Submit process "
3-3-1 Click "Submit"
[]>>
3-3-2 Add the advanced request
[]>>
3-3-3 Check if the request return an error
[]>>
3-3-4 Display "Error" message

Figure B.47: CTT

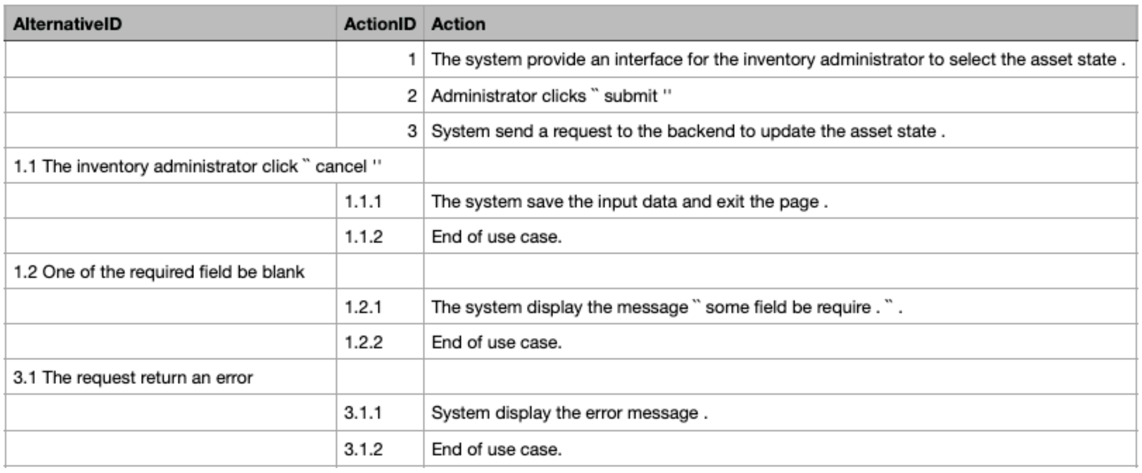
doi:10.6342/NTU202404066

B.1.9 Use Case：[Modify Asset State]



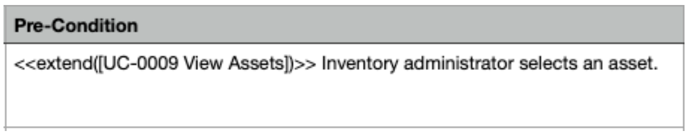
UseCaseName	Requirements	Actor
Modify Asset State	1. While the inventory administrator views assets, when the inventory administrator selects an asset, the system shall allow the inventory administrator to modify asset state. 2. The system shall provide an interface for the inventory administrator to select the asset state. 3. When the inventory administrator clicks "Submit", the system shall send a request to the backend to update the asset state.	Inventory administrator

Figure B.48: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the inventory administrator to select the asset state .
	2	Administrator clicks〝submit〞
	3	System send a request to the backend to update the asset state .
1.1 The inventory administrator click〝cancel〞		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message〝some field be require .〞.
	1.2.2	End of use case.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

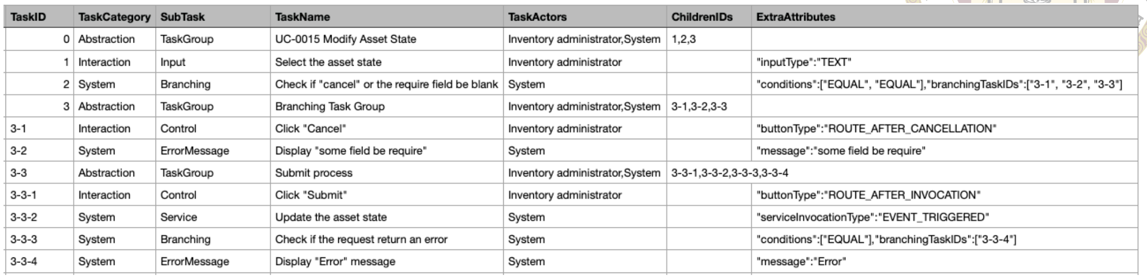
Figure B.49: Use Case Spec



Pre-Condition
<<extend([UC-0009 View Assets])>> Inventory administrator selects an asset.

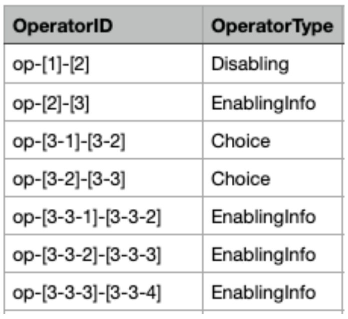
Figure B.50: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0015 Modify Asset State	Inventory administrator,System	1,2,3	
1	Interaction	Input	Select the asset state	Inventory administrator		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	Inventory administrator,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	Inventory administrator		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Submit process	Inventory administrator,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Submit"	Inventory administrator		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Update the asset state	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.51: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.52: Task Model - Temporal Operator

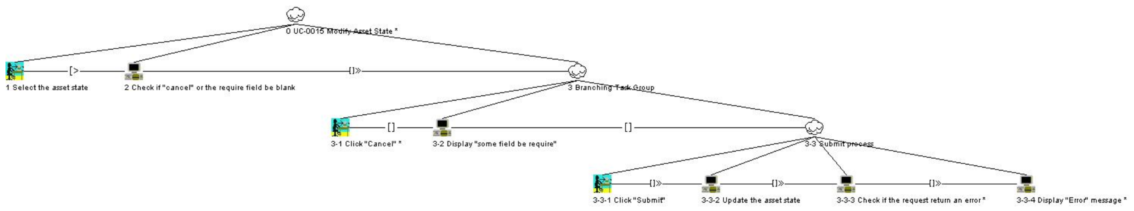
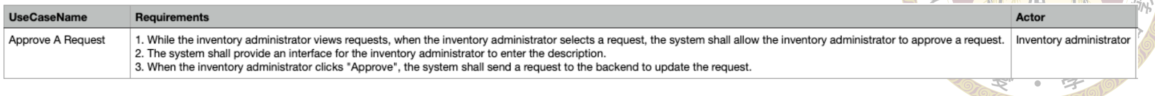

0 UC-0015 Modify Asset State *
1 Select the asset state
[>
2 Check if "cancel" or the require field be blank
[]»
3 Branching Task Group
3-1 Click "Cancel" *
[]
3-2 Display "some field be require"
[]
3-3 Submit process
3-3-1 Click "Submit"
[]»
3-3-2 Update the asset state
[]»
3-3-3 Check if the request return an error *
[]»
3-3-4 Display "Error" message *

Figure B.53: CTT

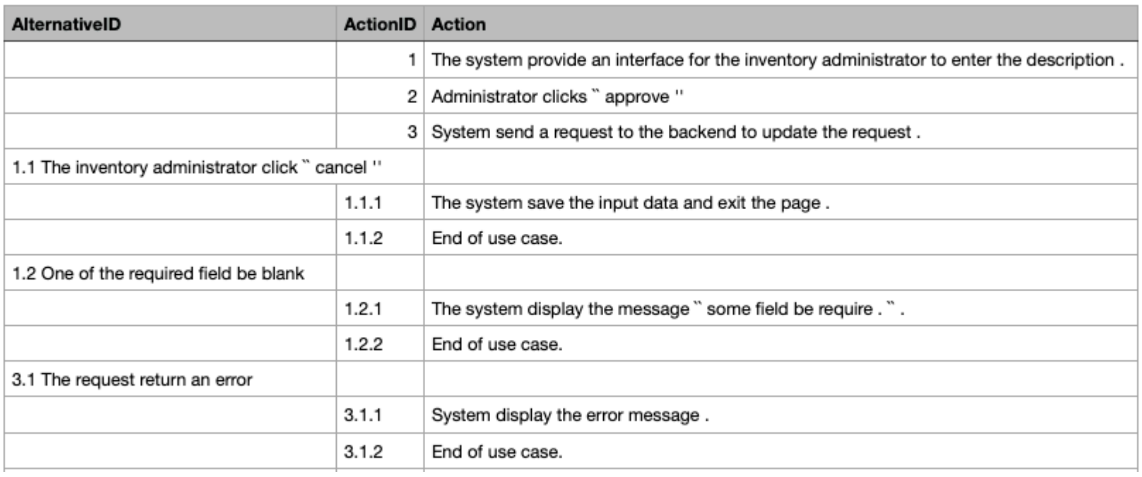
doi:10.6342/NTU202404066

B.1.10 Use Case：[Approve a Request]



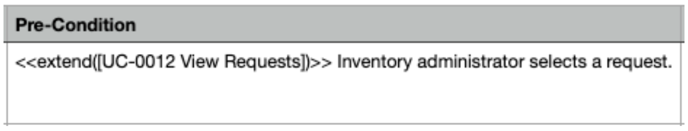
UseCaseName	Requirements	Actor
Approve A Request	1. While the inventory administrator views requests, when the inventory administrator selects a request, the system shall allow the inventory administrator to approve a request. 2. The system shall provide an interface for the inventory administrator to enter the description. 3. When the inventory administrator clicks "Approve", the system shall send a request to the backend to update the request.	Inventory administrator

Figure B.54: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the inventory administrator to enter the description .
	2	Administrator clicks〝approve〞
	3	System send a request to the backend to update the request .
1.1 The inventory administrator click〝cancel〞		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message〝some field be require .〝 .
	1.2.2	End of use case.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

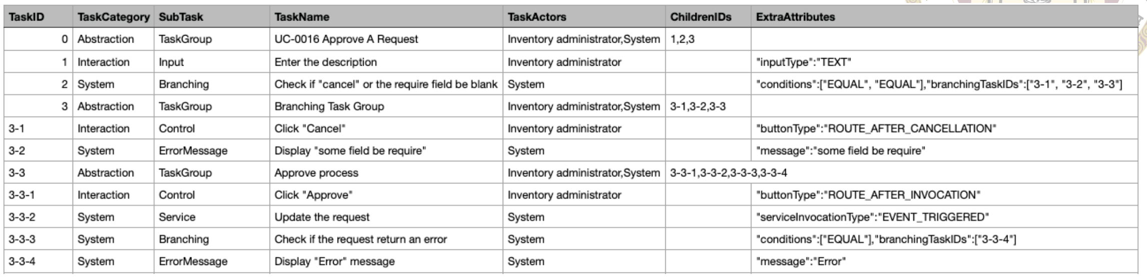
Figure B.55: Use Case Spec



Pre-Condition
<<extend([UC-0012 View Requests])>> Inventory administrator selects a request.

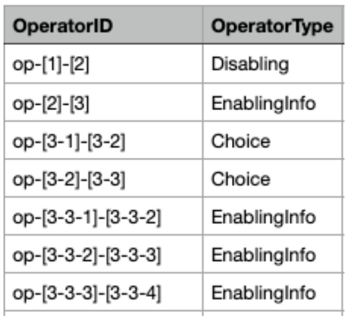
Figure B.56: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0016 Approve A Request	Inventory administrator,System	1,2,3	
1	Interaction	Input	Enter the description	Inventory administrator		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	Inventory administrator,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	Inventory administrator		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Approve process	Inventory administrator,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Approve"	Inventory administrator		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Update the request	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.57: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.58: Task Model - Temporal Operator

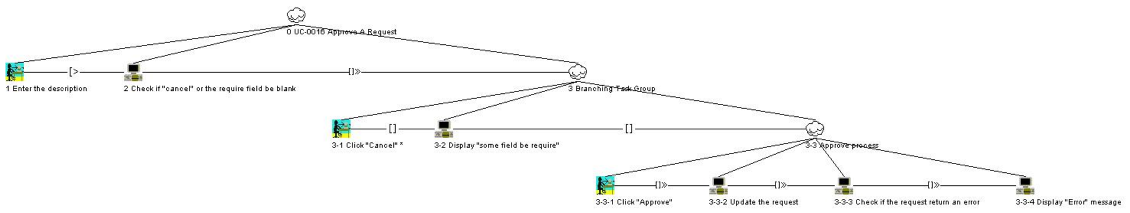
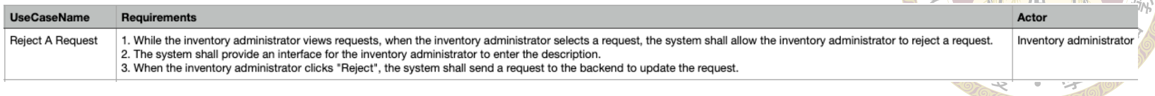

0 UC-0016 Approve A Request
1 Enter the description
[>
2 Check if "cancel" or the require field be blank
[]>>
3 Branching Task Group
3-1 Click "Cancel" *
[]
3-2 Display "some field be require"
[]
3-3 Approve process
3-3-1 Click "Approve"
[]>>
3-3-2 Update the request
[]>>
3-3-3 Check if the request return an error
[]>>
3-3-4 Display "Error" message

Figure B.59: CTT

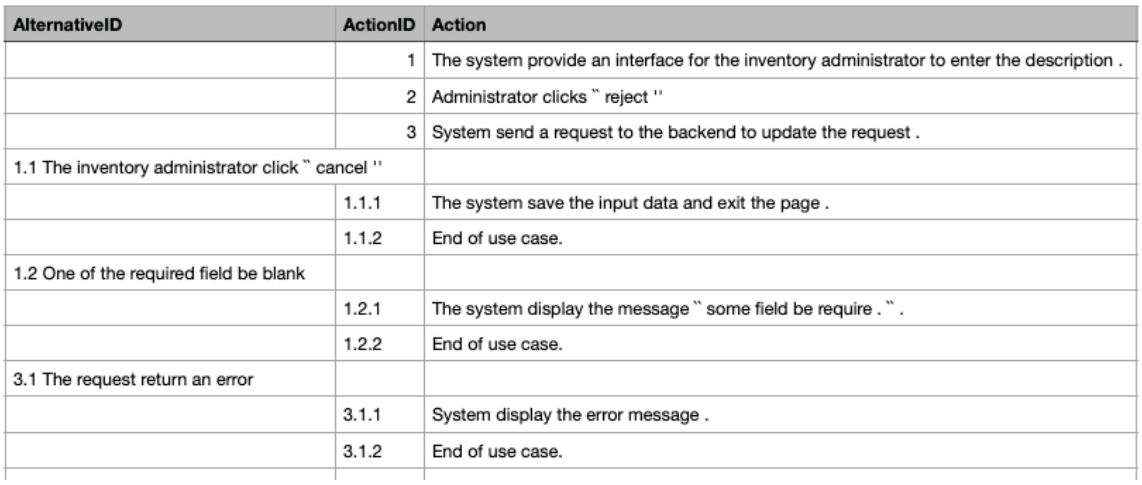
doi:10.6342/NTU202404066

B.1.11 Use Case：[Reject a Request]



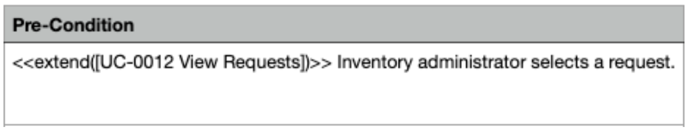
UseCaseName	Requirements	Actor
Reject A Request	1. While the inventory administrator views requests, when the inventory administrator selects a request, the system shall allow the inventory administrator to reject a request. 2. The system shall provide an interface for the inventory administrator to enter the description. 3. When the inventory administrator clicks "Reject", the system shall send a request to the backend to update the request.	Inventory administrator

Figure B.60: Requirement



AlternativeID	ActionID	Action
	1	The system provide an interface for the inventory administrator to enter the description .
	2	Administrator clicks ‟ reject ''
	3	System send a request to the backend to update the request .
1.1 The inventory administrator click ‟ cancel ''		
	1.1.1	The system save the input data and exit the page .
	1.1.2	End of use case.
1.2 One of the required field be blank		
	1.2.1	The system display the message ‟ some field be require . ‟ .
	1.2.2	End of use case.
3.1 The request return an error		
	3.1.1	System display the error message .
	3.1.2	End of use case.

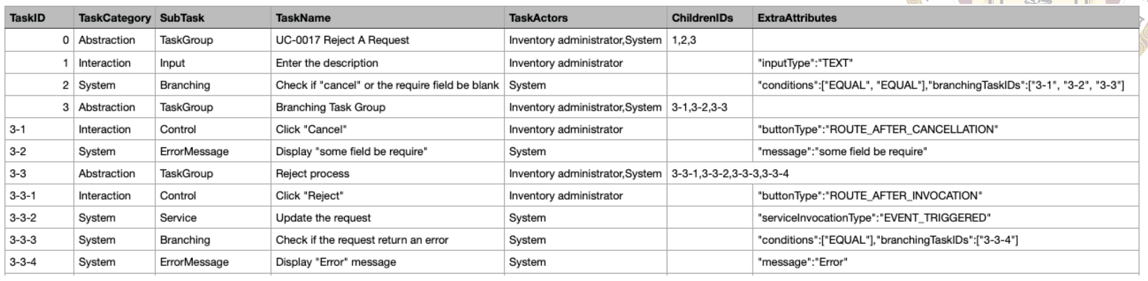
Figure B.61: Use Case Spec



Pre-Condition
<<extend([UC-0012 View Requests])>> Inventory administrator selects a request.

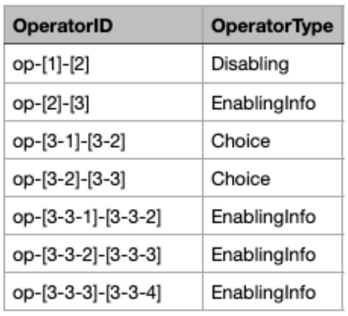
Figure B.62: Use Case Diagram

doi:10.6342/NTU202404066



TaskID	TaskCategory	SubTask	TaskName	TaskActors	ChildrenIDs	ExtraAttributes
0	Abstraction	TaskGroup	UC-0017 Reject A Request	Inventory administrator,System	1,2,3	
1	Interaction	Input	Enter the description	Inventory administrator		"inputType":"TEXT"
2	System	Branching	Check if "cancel" or the require field be blank	System		"conditions":["EQUAL", "EQUAL"],"branchingTaskIDs":["3-1", "3-2", "3-3"]
3	Abstraction	TaskGroup	Branching Task Group	Inventory administrator,System	3-1,3-2,3-3	
3-1	Interaction	Control	Click "Cancel"	Inventory administrator		"buttonType":"ROUTE_AFTER_CANCELLATION"
3-2	System	ErrorMessage	Display "some field be require"	System		"message":"some field be require"
3-3	Abstraction	TaskGroup	Reject process	Inventory administrator,System	3-3-1,3-3-2,3-3-3,3-3-4	
3-3-1	Interaction	Control	Click "Reject"	Inventory administrator		"buttonType":"ROUTE_AFTER_INVOCATION"
3-3-2	System	Service	Update the request	System		"serviceInvocationType":"EVENT_TRIGGERED"
3-3-3	System	Branching	Check if the request return an error	System		"conditions":["EQUAL"],"branchingTaskIDs":["3-3-4"]
3-3-4	System	ErrorMessage	Display "Error" message	System		"message":"Error"

Figure B.63: Task Model - Task



OperatorID	OperatorType
op-[1]-[2]	Disabling
op-[2]-[3]	EnablingInfo
op-[3-1]-[3-2]	Choice
op-[3-2]-[3-3]	Choice
op-[3-3-1]-[3-3-2]	EnablingInfo
op-[3-3-2]-[3-3-3]	EnablingInfo
op-[3-3-3]-[3-3-4]	EnablingInfo

Figure B.64: Task Model - Temporal Operator

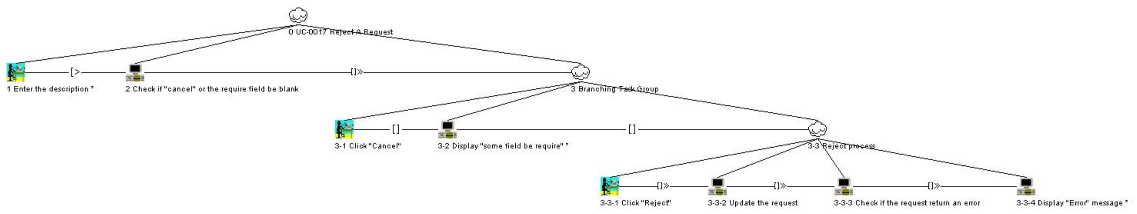
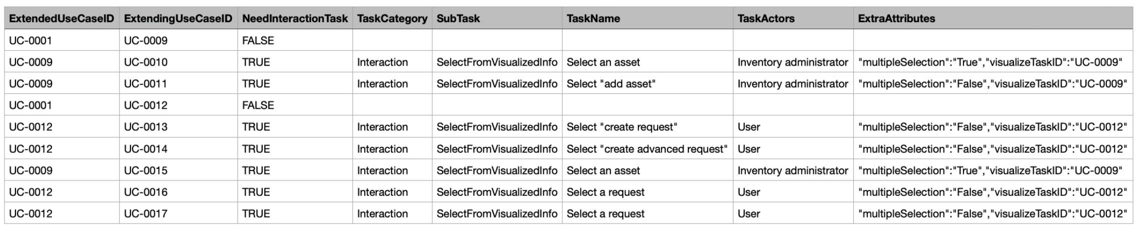

0 UC-0017 Reject A Request
1 Enter the description *
[>
2 Check if "cancel" or the require field be blank
[]»
3 Branching Task Group
3-1 Click "Cancel"
[]
3-2 Display "some field be require" *
[]
3-3 Reject process
3-3-1 Click "Reject"
[]»
3-3-2 Update the request
[]»
3-3-3 Check if the request return an error
[]»
3-3-4 Display "Error" message *

Figure B.65: CTT

doi:10.6342/NTU202404066

B.1.12 Main CTT

The use case information mentioned above, along with the predicted Interaction Task information, generates the Main CTT information as shown in (Figure B.66). The Main CTT is shown in (Figure B.67).



ExtendedUseCaseID	ExtendingUseCaseID	NeedInteractionTask	TaskCategory	SubTask	TaskName	TaskActors	ExtraAttributes
UC-0001	UC-0009	FALSE					
UC-0009	UC-0010	TRUE	Interaction	SelectFromVisualizedInfo	Select an asset	Inventory administrator	"multipleSelection":"True","visualizeTaskID":"UC-0009"
UC-0009	UC-0011	TRUE	Interaction	SelectFromVisualizedInfo	Select "add asset"	Inventory administrator	"multipleSelection":"False","visualizeTaskID":"UC-0009"
UC-0001	UC-0012	FALSE					
UC-0012	UC-0013	TRUE	Interaction	SelectFromVisualizedInfo	Select "create request"	User	"multipleSelection":"False","visualizeTaskID":"UC-0012"
UC-0012	UC-0014	TRUE	Interaction	SelectFromVisualizedInfo	Select "create advanced request"	User	"multipleSelection":"False","visualizeTaskID":"UC-0012"
UC-0009	UC-0015	TRUE	Interaction	SelectFromVisualizedInfo	Select an asset	Inventory administrator	"multipleSelection":"True","visualizeTaskID":"UC-0009"
UC-0012	UC-0016	TRUE	Interaction	SelectFromVisualizedInfo	Select a request	User	"multipleSelection":"False","visualizeTaskID":"UC-0012"
UC-0012	UC-0017	TRUE	Interaction	SelectFromVisualizedInfo	Select a request	User	"multipleSelection":"False","visualizeTaskID":"UC-0012"

Figure B.66: Main CTT information

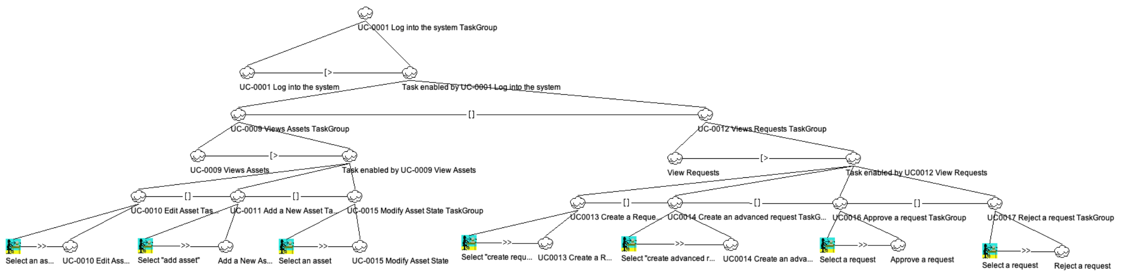

UC-0001 Log into the system TaskGroup
UC-0001 Log into the system
[>
Task enabled by UC-0001 Log into the system
UC-0009 Views Assets TaskGroup
[]
UC-0012 Views Requests TaskGroup
UC-0009 Views Assets
[>
Task enabled by UC-0009 View Assets
View Requests
[>
Task enabled by UC0012 View Requests
UC-0010 Edit Asset Tas...
[]
UC-0011 Add a New Asset Ta...
[]
UC-0015 Modify Asset State TaskGroup
UC0013 Create a Reque...
[]
UC0014 Create an advanced request TaskG...
[]
UC0016 Approve a request TaskGroup
[]
UC0017 Reject a request TaskGroup
>>
Select an as...
UC-0010 Edit Ass...
Select "add asset"
Add a New As...
Select an asset
UC-0015 Modify Asset State
Select "create requ...
UC0013 Create a R...
Select "create advanced r...
UC0014 Create an adva...
Select a request
Approve a request
Select a request
Reject a request

Figure B.67: Main CTT

The above is an example of how the Inventory System automatically generates Use Case CTT and Main CTT from the Use Case Specification using the trained model.

doi:10.6342/NTU202404066