Regular Paper

Automatic Generation of State Transition Diagrams from Requirement Specifications Using Chain of Thought Prompts

Takeki Ninomiya†, Maiko Onishi‡, Shinpei Ogata*, and Kozo Okano*

†Graduate School of Science and Engineering, Shinshu University, Japan
‡Graduate School of Humanities and Sciences, Ochanomizu University, Japan
* Faculty of Engineering, Shinshu University, Japan
{23w2074b}@shinshu-u.ac.jp

Abstract - In software development, development proceeds using requirement specifications that describe software requirements in natural language. However, ambiguities in the description of natural words may cause unintended behavior in the system. To solve such problems, software developers create state transition diagrams from requirement specifications and perform model checking. The purpose of this paper is to automatically convert requirement specifications written in natural language to state transition diagrams. Using a large-scale language model as the conversion method, state transition diagrams are created by extracting state transitions from requirement statements and converting them to PlantUML description format. As a prompting method, chain-of-thought prompts are used, and the prompt describes the process of converting requirement statements into PlantUML. The experiment confirmed that the state transition diagrams generated using this method accurately reflected the information about states, transitions, and events described in the requirement statements. However, model checking of the generated timer state transition diagram revealed omissions and errors in guard conditions and actions. On the other hand, combining this method with model checking enabled the identification of deficiencies in the requirement statements.

Keywords: Requirement Specification, State Transition Diagrams, LLM, PlantUML, Chain-f-Thought Prompting

1 INTRODUCTION

In software development, the process often proceeds by using requirement specifications written in natural language [1]. Those requirements are written on the assumption that the product will behave as the developer expects it to behave. However, when reflecting requirements, ambiguities in natural language and inconsistencies in requirements can cause the system to behave differently from the specification and in unintended ways [2]. Design errors due to ambiguous or inconsistent wording are often discovered in the testing process later in the development process, and these ambiguous statements force the developer to go back to the design process again [3][4]. Rework caused by the testing process errors increases a great deal of extra costs. One of the methods to prevent such rework is to create state transition diagrams from requirement specifications and perform model checking. Model checking based on state transition diagrams allows developers to check for unrecoverable and undesirable system states caused by unintended behavior during the design phase. However, creating state transition diagrams and inspecting models requires specialized knowledge. Therefore, it is difficult for beginners to handle. In addition, if the system is complex and consists of many components, manually extracting all the states of the components can be a labor-intensive task. Therefore, the goal of this research is the automatic conversion of state transition diagrams from requirement specifications containing state transition descriptions written in Japanese. This research is expected to help designers and developers share system specifications without conflicts at low cost. Previous research has proposed a method for extracting state transition descriptions from requirement statements using syntactic analysis based on the rules of natural language notation, and creating state transition diagrams based on the extracted elements [5]-[8]. A rule-based method using dependency analysis can extract the name of a state variable and its state from requirement statements [9]. However, previous research had several issues. The first issue was the inability to fully extract the necessary elements. This was particularly challenging when dealing with complex transition conditions, as it was difficult to extract all elements while clarifying their logical relationships. The second issue was the need to add new syntax rules or refine existing rules when extracting information from long and complex sentences. To address the first issue, it was necessary to improve extraction rules. However, creating rules that could handle all types of sentences, such as those with numerous modifiers or redundant phrasing, proved to be difficult. The final issue was that simply fitting the extracted elements into a state transition diagram template did not result in a correct diagram. This was due to problems with multiple names or definitions referring to the same state. In summary, while previous research automated the partial extraction of elements for state transition descriptions to some extent, it did not achieve complete automation of the diagram creation process.

On the other hand, natural language processing technology using Large Language Models (LLMs) is rapidly developing and its usefulness is being confirmed. Many studies in the field of software modeling have also explored the use of LLMs such as ChatGPT. In some studies, LLM was used to convert requirement statements into UML descriptions such as plantUM and to create UML diagrams [10]. Prior research

confirmed that chatGPT understands most UML diagrams, including class diagrams, use cases, state transition diagrams, sequence diagrams, and activity diagrams [10]. This research uses ChaTGPT to create state transition diagrams by extracting state transitions from requirement statements described in natural language and converting them to PlantUML description format. This enables automation up to the creation of state transition diagrams, which was previously impossible. Furthermore, this study will evaluate the extent to which the state transition diagrams created by this method. The proposed method uses a prompting technique called Chain of Thought Prompting, which gave ChatGPT an example of describing a state transition diagram in PlantUML format and a procedure for creating it. The results confirmed that ChatGPT can create state transition diagrams that satisfy the elements of state, transition, and event.This paper is organized as follows. Section 2 presents the technology used in this study and related research. Section 3 presents the proposed methodology. Sections 4 and 5 present the experiments and results, respectively. Section 6 discusses the method based on the results. Finally, Section 7 concludes the paper.

2 PREPARE

2.1 State Transition Diagrams

A state transition diagram represents the behavior of a model consisting of a combination of states, transitions, and events. In model-driven development, state transition diagrams are widely used for purposes such as checking implementation specifications and analyzing scenarios [11]. In UML 2.0, state transition diagrams are called state machine diagrams. State transition diagrams have a notation that specifies the method of operation, called semantics. The semantics include conditions and transition actions related to transitions, actions and activities inside states, and structures such as composite and parallel states [11].

2.2 Modeling with ChatGPT

Large Language Models (LLMs) are models of natural language processing that have been trained on large amounts of text data. Typical examples of large-scale language models include BERT, announced by Google in 2018, and GPT-3, announced by OpenAI in 2020. In May 2024, OpenAI introduced the new GPT-4o, a model with better performance in languages other than English. This study used the GPT3.5 and GPT-4o models for its experiments. Since ChatGPT is a language model, it cannot generate graphical models, but it can generate models using text-based UML notation. ChatGPT understands several UML notations such as PlantUML, Mermaid, Markdown UML. Prior studies have shown several characteristics of software modeling using ChatGPT [10].

- ChatGPT can represent models in multiple UML notations. In general, PlantUML tends to have fewer syntax errors.
- Previous conversation history will cause fluctuations in the generated results.
- Variation in ChatGPT responses to the same prompt
- The problem domain affects the structure, content, and level of abstraction of the generated model. Also, ChatGPT performs poorly when modeling meaningless entity names such as symbols.

Considering the above, this method selected PLantUML as the UML notation. PlantUML is a tool that allows code-based description of UML and has been used in previous studies for automated creation of UML diagrams [12]. Also, prior research using chatGPT recommends starting a new chat each time a new model is generated. This is because previous conversation history influences the results generated. Therefore, this experiment also switched to a new chat in each experiment. Since April 2024, a feature called memory has been added to the paid version of ChatGPT plus, which allows information to be stored in ChatGPT beyond each chat section, but in this experiment, the memory feature was turned off.

2.3 Related Research

This section presents related research that applies natural language processing (NLP) techniques to support the creation and validation of UML diagrams. We categorize them into three groups: software modeling methods using LLM, rule-based methods, and other traditional NLP methods.

Studies that have utilized LLMs as UML generation tools and evaluated their accuracy include Cámara et al. [10] and Ferrari et al. [13]. In Ferrari et al., LLMs were used to generate sequence diagrams from requirements. Additionally, Wang et al. [14] used LLMs as a modeling support tool and evaluated the generated UML diagrams such as class diagrams and sequence diagrams. The rule-based element extraction of traditional methods is often used not only in software modeling but also for automatic test case generation in embedded systems and other applications. Muhammad et al. [8] designed rules to extract actions and conditions from English sentences, and by extracting them, they supported the design verification of embedded systems. Nakamura et al. [7] used natural language processing tools JUMAN and KNP to extract clauses representing conditions and actions through a syntactic tree analysis approach for requirements written in Japanese. Abdelkareem [12] extracted elements from scenario-based text and created sequence diagrams by describing the extracted elements in PlantUML. Rule-based methods generally perform extraction by following a sequence of steps, including preprocessing, morphological analysis, and syntactic analysis. As a different method from rule-based, Shaohong et al. [15] used natural language processing techniques such as inverse document frequency (IDF) calculation and open information extraction (IE) to extract important words and phrases, thereby generating SysML diagrams. The positioning of this research is in software modeling using LLMs, with a particular focus on state transition diagrams.

3 PROPOSED METHOD

This section provides an overview of the methodology. An overview of this method is shown in Fig. 1. Figure 1 rep-

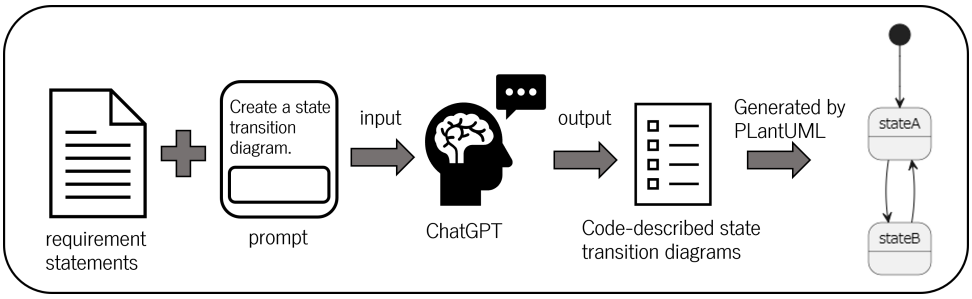

Figure 1: Overview of the proposed method

```
    pre-state --> post-state : event
    post-state: (entry or do or exit)action
It represents how the PlantUML format is structured when the requirement statements
   includes the elements pre-state, post-state, event, and action.

For example, the statement "When the power is off, turning on the power switches to
   standby mode and lights up the standby lamp." is converted as follows.
    Power Off --> Standby : Turn on the power
    Standby : entry / Light up the standby lamp
```

Listing 1: Part of the process of converting requirement statements into PlantUML

resents giving ChatGPT requirement statements and prompts explaining how to create a state transition diagram, and receiving a state transition diagram described in PlantUML format as a response from ChatGPT. This is executed in PlantUML and converted into a state transition diagram image. Requirement statements are generally classified into functional and non-functional requirements. Functional requirements include elements such as UI, database, processes, and context. In this method, the primary input statements are those that describe the processes in the functional requirements. Statements describing processes consist of elements such as activities, parameters, and rules, and they represent the processing of transitions and actions. A transition statement is a statement that describe a change of state by a pair of states and events [16]. An active statement is a statement that, based on a combination of state and event, instructs the execution of operations [16]. Or it instructs the interactions with the external environment through actuators or other means.

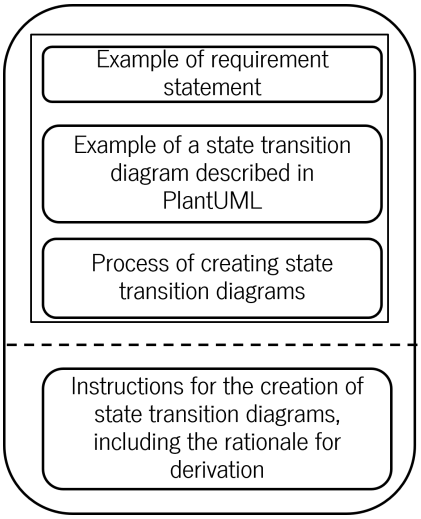

Figure 2: Composition of prompts

To enable ChatGPT to appropriately convert these requirements into state transition diagrams, careful crafting of prompts is essential. This method used Chain of Thought Prompting as the prompt description method. It has been found that ChatGPT responses are more concrete when specific examples of inputs and outputs are given. Chain of Thought Prompting is a method that improves LLM capabilities by including intermediate reasoning steps before solving the problem in the prompt [17]. Chain of thought prompts have been shown to improve the performance of LLMs in tasks involving arithmetic, common sense reasoning, and symbolic reasoning [17]. As an intermediate inference step, this method describes the process of extracting the state transition description from the requirement statement and converting it to PlantUML format. Figure 2 shows the prompt used in this study. As an example, ChatGPT receives requirement statements, a PlantUML description created from the requirement statements, and the procedure for converting the requirement statements to PlantUML. Lstlisting 1 shows part of a prompt that illustrates the process of converting requirement statements to PlantUML. Usually, a requirement statement describing the behavior of the system indicates that when an event occurs, the system will go from state A to state B and take action. For example, requirement statements regarding a timer is described as follows: “タイマ起動中に、タイマボタンを 3sec 以上続けて長押ししたら、ブザーを 100msec 鳴らした後、0min0sec にリセットされ、タイマが停止する。” (“When

the timer is running, if the timer button is pressed and held for more than 3 seconds, the buzzer will sound for 100 milliseconds, then the timer will reset to 0 minutes and 0 seconds, and the timer will stop.") Therefore, this study defines the elements that constitute state transitions as the pre-state, post-state, event, guard condition, and action. The prompt in Lstlisting 1 explains how to convert each of those elements to PlantUML format when they are included in requirement statements. This allows ChatGPT to understand conversion patterns according to the rules.

There are two ways to describe actions in a state transition diagram:

1. Actions executed upon entering or exiting a state, or while staying within that state.
2. Conditional actions executed when conditions are satisfied, and the transition is successfully completed.

The former type of action is documented within the state using entry, exit, or do, depending on the execution timing. Entry is an action executed once upon transitioning into a state, exit is an action executed once upon leaving a state, and do is an activity continuously executed while remaining in the state. An action refers to a process that is executed without interruption or suspension, whereas an activity refers to a process that is expected to allow interruptions or suspensions during execution. The prompt explains that, depending on the execution timing, one of entry, exit, or do should be specified for the action. On the other hand, the latter type of conditional action is documented within the transition. In a state transition diagram, conditional actions are represented as "event [guard condition]/conditional action". When the requirement statement contains a conditional action, the prompt instructs ChatGPT to convert it as follows:

- pre-state –> post-state : event [guard condition]/ conditional action

For example, a timer requirement statement is converted by ChatGPT into a PlantUML description as follows.

- 「タイマ起動中に、タイマボタンを 3sec 以上続けて長押ししたら、ブザーを 100msec 鳴らした後、0min0sec にリセットされ、タイマが停止する。」
- 起動中 –> 停止中 : タイマボタンを 3sec 以上長押し/100msec ブザー鳴動，$t = 0min0sec$
- "When the timer is running, if the timer button is pressed and held for more than 3 seconds, the buzzer will sound for 100 milliseconds, then the timer will reset to 0 minutes and 0 seconds, and the timer will stop."
- Running –> Stopped : Press and hold the timer button for more than 3 seconds/buzzer sound for 100msec, $t = 0min0sec$

The execution order of processes associated with a state transition is as follows: checking the guard condition, executing the transition if the condition is met, performing the conditional action, and executing the entry action of the post-transition state. Since conditional actions and entry actions within a state are executed within the same cycle, it is recommended to consolidate operations involving the same target or variable into either one [11]. It is also recommended that conditional actions include processes that cannot be interrupted and can be completed in a short amount of time. Whether an action is documented as a state action or a conditional action is determined by the factors mentioned above, as well as the common design rules established by the development team. In this method, the prompt does not specify whether an action should be documented as a state action or a conditional action; the final decision is left to ChatGPT.

As shown in Fig. 2, the prompt for creating a state transition diagram includes, in addition to the input statements, the instruction to indicate from which statements you have created which transition, together with the rationale for its derivation. This instruction causes the ChatGPT response to output the process of creating a transition from requirement statements with the result. If an error is found in the state transition diagram through model checking, the developer may be able to identify the cause of the error by examining the derivation process. Therefore, it is recommended to output the derivation process along with the generated result during the creation.

4 EXPERIMENTS

The evaluation experiments investigated the quality of state transition diagrams generated using the proposed method. Additionally, they compared the transformation accuracy with that of conventional methods to assess the differences. As a preliminary experiment, we utilized GPT-3.5 and GPT-4o and compared the results using two prompting methods: the proposed Chain-of-Thought approach and the Zero-Shot approach (i.e., without prompt optimization). The best results were obtained when using GPT-4o with the Chain-of-Thought method. Therefore, in the main experiment, we use GPT-4o as the LLM and apply the Chain-of-Thought prompting method for the transformation process.

4.1 Requirements Specification Used in the Experiment

In the experiment, two requirement specifications were prepared, and three state transition diagrams were created. The first is a requirement statement for a CD player control panel in Astah's "UML State Machine Diagram and State Transition Table Tutorial" [18]. The requirements for the CD player operation panel are described as transition statements in a basic syntactic format, such as "When in the previous state, if an event occurs, it transitions to the subsequent state and performs an action." There are eight requirement sentences in total: six sentences describing state transitions, one sentence describing state actions, and one sentence describing the initial state. The experiment will verify whether ChatGPT can create a simple state transition diagram with a small number of states through this requirement statement.

The second is the requirement statement from the "Hotpot, Version 7" created and published by SESSAME [19]. Compared to the requirements for the CD player, the requirement

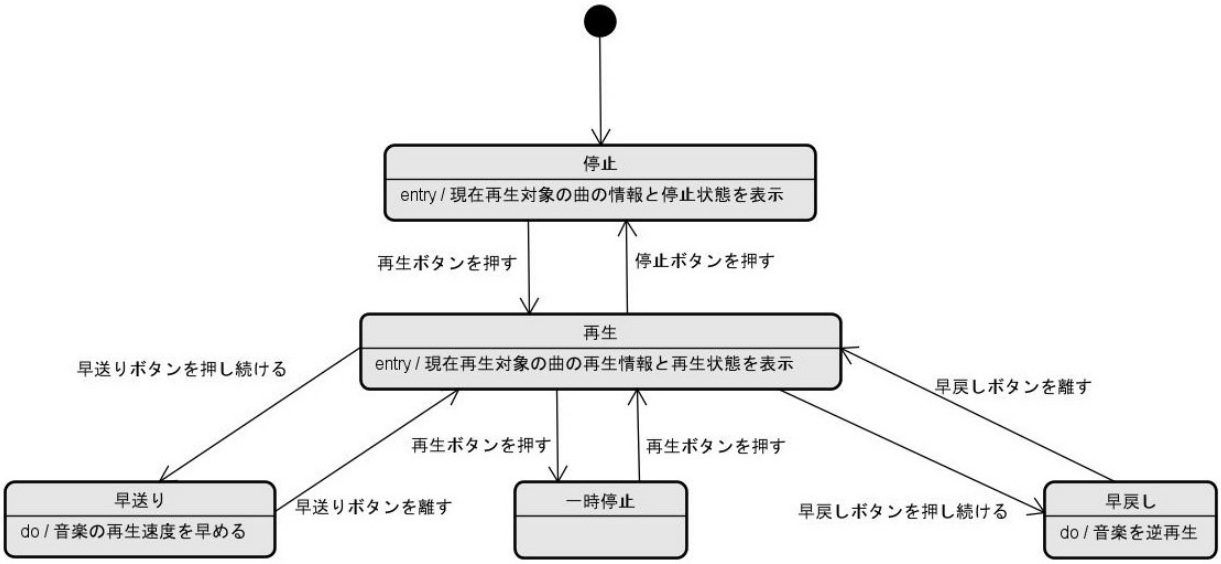

Figure 3:
The state transition diagram of the CD player control panel created using chain of thought prompting in GPT4o.
-Stop (停止), Playback (再生), Pause (一時停止), Fast forward (早送り), Fast Rewind (早戻し)
-Press the Play button (再生ボタンを押す), Press the Stop button (停止ボタンを押す)
-Hold down the Fast Forward button (早送りボタンを押し続ける), Release the Fast Forward button (早送りボタンを離す)
-Hold down the Fast Rewind button (早戻しボタンを押し続ける), Releasethe Fast Rewind button (早戻しボタンを離す)
-Display the information of the currently playing track and the stopped state (現在再生対象の曲の情報と停止状態を表示)
-Display the playback information and playback state of the currently playing track
(現在再生対象の曲の再生時情報と再生状態を表示)
-Increase the music playback speed (音楽の再生速度を早める), Play the music in reverse (音楽を逆再生)

statements for the Hotpot are more complex, with more detailed descriptions of transition conditions and actions. This experiment will verify whether appropriate state transition diagrams can be created based on requirement statements written in a style similar to those used in the field. Additionally, the Hotpot has been used as an example in several related studies [4][9]. Therefore, it was deemed beneficial for evaluating the transformation accuracy of this method and was selected as the requirement specification for the experiment. In this experiment, two items were targeted: the lock button item and the timer item from the "Hot Pot." The requirement statements for the lock button total nine. Of these, three are described as bullet points outlining conditions. The remaining six consist of five sentences describing states and transitions, and one sentence explaining the initial state. The lock button is characterized by its transition conditions being listed as bullet points. In creating the state transition diagram, the process involved first inputting six requirements into GPT and then subsequently inputting the bullet-pointed conditions. On the other hand, the timer's requirement statements consist of a total of nine sentences. Among them, four sentences describe state transitions, three sentences explain actions related to states or transitions, one sentence pertains to constraints, and one sentence describes the initial state. The timer must process increment and decrement operations on the remaining time. When creating the state transition diagram, instructions were added the input sentences, stating: "Define the remaining time of the timer as a variable t, consider the range of values t can take, and incorporate this into the state transition diagram."

4.2 Evaluation Methods

In the experiment, we conducted evaluations using three different methods:

(1) Comparison between the state transition diagram generated by the proposed method and the correct diagram.

(2) Comparison of accuracy between the conventional method and the proposed method.

(3) Evaluation of the generated state transition diagram through model checking.

In the first method, the generated state transition diagram was compared with a predefined correct state transition diagram to determine correctness. The Astah tutorial includes a state transition diagram created from the request statement, and this experiment treats it as the correct diagram. The state transition diagram to be used as the correct answer for the topic boiling pot was created by ourself in advance. The evaluation defined states, transitions, events, and semantics as the evaluation criteria for the state transition diagrams created. The evaluation checks that the state, transitions, and events accurately reflect what should be described in the state transition diagram without deficiencies. Regarding transitions, it is

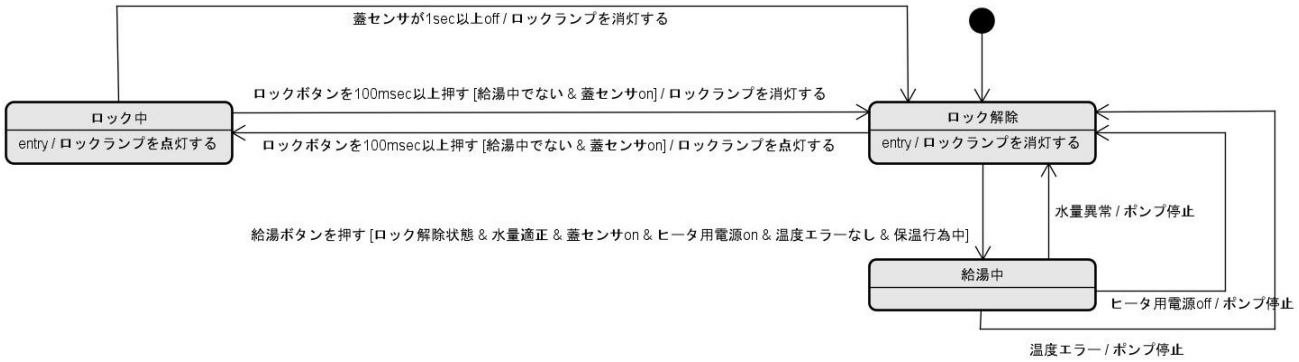

Figure 4:
The state transition diagram of the lock button created using chain of thought prompting in GPT4o.
-Locked (ロック中), Unlock (ロック解除), Boiling (給湯中), Stop the pump (ポンプ停止)
-Press the lock button for more than 100msec (ロックボタンを 100msec 以上押す)
-[Not in boiling water & Lid sensor on] ([給湯中でない & 蓋センサ on])
-Lid sensor off for more than 1sec (蓋センサが 1sec 以上 off), Press the boiling button (給湯ボタンを押す)
-[Unlocked state & Proper water level & Lid sensor on & Heater power on & No temperature error & Insulating]
([ロック解除状態 & 水量適正 & 蓋センサ on & ヒータ用電源 on & 温度エラーなし & 保温行為中])
-Abnormal water level (水量異常), Heater power off (ヒータ用電源 off), Temperature error (温度エラー)
-Turn off the lock lamp (ロックランプを消灯する), Turn on the lock lamp (ロックランプを点灯する)

common practice not to specify transition conditions for transitions from the initial pseudo-state to the initial state. However, since requirement statements defining the initial state may include events or conditions, such cases were excluded from the evaluation of transition correctness in the experiment.

For the second method, which compares accuracy with the conventional approach, we use Precision and Recall as evaluation metrics to measure correctness. The conventional approach refers to the rule-based method that utilizes dependency parsing. The main process of the conventional method consists of morphological analysis, dependency parsing, rule application, and element extraction. For morphological analysis and dependency parsing, we use GiNZA, an open-source natural language processing library [20]. By using GiNZA, it is possible to perform dependency parsing and named entity recognition based on Universal Dependencies (UD) dependency labels [21]. In previous research, extraction rules for states and transition conditions were designed using part-of-speech tagging and UD labels. A total of 19 rules were prepared for this purpose. The dependency parsing rule-based method extracts transition conditions, pre-transition states, and post-transition states from a single sentence. In contrast, the transformation using LLM converts the entire requirement text into PlantUML at once. To compare with conventional methods, values are calculated based on the extraction items used in conventional methods: transition conditions, pre-transition states, and post-transition states. In the LLM-based transformation, the pre-state corresponds to the pre-transition state, the post-state and action correspond to the post-transition state, and the event and guard condition correspond to the transition condition. Since the rule-based method is designed with rules targeting transition statements, the comparison was conducted only on transition statements. For the conventional method, some requirement statements were preprocessed by supplementing the subject and object and modifying the expressions at the end of clauses.

Finally, to determine the correctness of the behavior of the generated state transition diagrams, model checking, a formal method widely used in software development, was utilized. If there are errors or redundancies in the generated state transition diagrams, the causes can be broadly classified into two categories: “errors during the LLM transformation process” and “ambiguities or contradictions in the requirement specifications.” LLM transformation errors are defined as cases where the transformation deviates from the requirement descriptions or when elements included in the requirements are missing. On the other hand, if the requirements are met but there are errors in the state transition diagram, it can be attributed to issues in the requirement specification itself. However, in general, it is difficult to determine the errors and their causes just by inspecting the generated results or the derivation process. Therefore, we first verify whether there are any errors through model checking. Afterward, by carefully examining the results of the model checking, we identify the causes of the errors. In the experiment, the most complex timer state transition diagram among the three was selected for verification.

5 RESULT

Figure 3 shows the state transition diagram of the CD player control panel created using chain of thought prompting in GPT4o. Figure 4 shows the state transition diagram of the lock button created using chain of thought prompting in GPT4o. Figure 5 shows the state transition diagram of the timer created using chain of thought prompting in GPT4o.

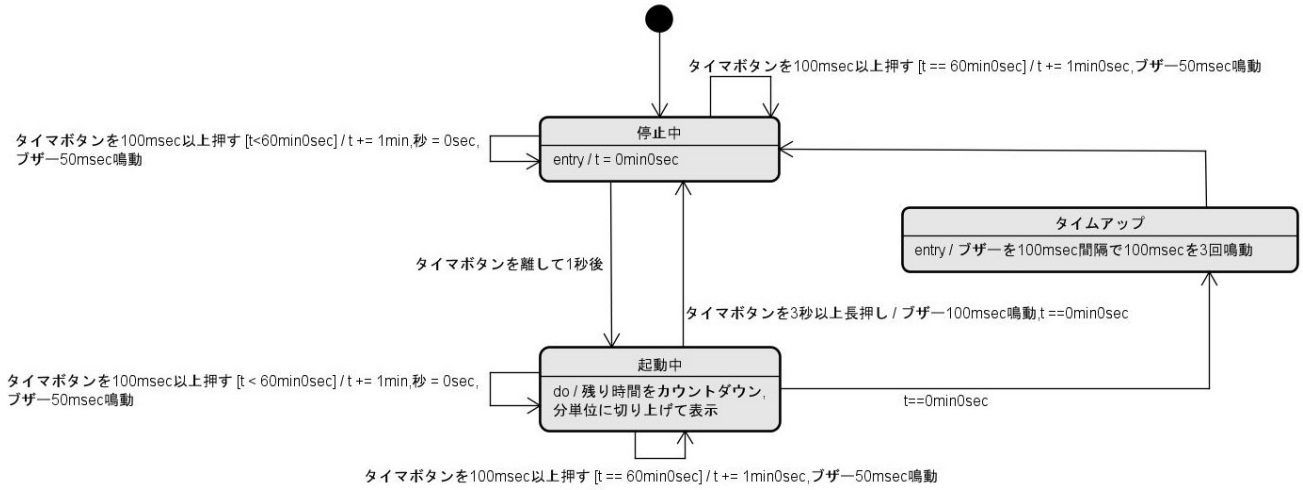

Figure 5:
The state transition diagram of the timer created using chain of thought prompting in GPT4o.
-Stopped (停止中), Running (起動中), Time's up (タイムアップ), Buzzer sounds (ブザー鳴動)
-Press the timer button for more than 100msec (タイマーボタンを 100msec 以上押す)
-Release the timer button and wait for 1 second (タイマーボタンを離して 1 秒後)
-Press and hold the timer button for more than 3 seconds (タイマーボタンを 3 秒以上長押し)
-Countdown the remaining time (残り時間をカウントダウン), display in minutes (分単位に切り上げて表示)
-Buzzer sounds three times for 100msec intervals (ブザーを 100msec 間隔で 100msec を 3 回鳴動)

5.1 CD Player Control Panel

The state transition diagram of a CD player control panel has five states, nine transitions including the transition from the start state to the initial state, and eight events for the transitions. In the experiments, when GPT-4 or chain-of-thought prompting was used, these states, transitions, and events were accurately reflected in the state transition diagram without any omissions or excesses. Although the notation of actions differs from that in the Astah diagram, the content is consistent. These good results can be attributed to the fact that the requirements are clear and have a simple sentence structure.

5.2 Lock Button

When using GPT-4 with chain-of-thought prompting, the states, transitions, and events of the lock button were accurately reflected in the state transition diagram, except for one transition and its associated event. The only transition that could not be reflected was that the pump stops when the hot water button is released during hot water heating. This transition could not be reflected because it was not described in the requirement statements. This functionality is presumed to have been omitted from the requirement statements because it was considered self-evident. It was confirmed that multiple transition conditions described in bullet points could be appropriately reflected by separating and inputting each process individually. It can be confirmed from Fig. 4 that all elements of the transition conditions are captured without omission, and their logical relationships are correctly represented as a conjunction (AND relationship). However, in these requirements, transition conditions such as "water level is appropriate" are defined as "the full water sensor is off, and not all water level sensors are off." Such rephrasing using alternative expressions is sometimes defined in the requirements for the sake of simplicity. Therefore, if a more rigorous definition is to be reflected in the state transition diagram, it is necessary to include the statements defining those conditions along with the input.

5.3 Timer

When using chain of thought prompting in GPT4o, the state, transitions, and events were accurately reflected in the state transition diagram, except for one transition and its events. The transition that could not be reflected was the self-transition caused by the countdown when the timer is activated. However, since this self-transition was not explicitly described in the requirement statement, this omission is understandable. As a result of instructing the inclusion of the remaining time as a variable t in the state transition diagram, actions that add time and transitions that depend on the value of the remaining time were effectively represented. On the other hand, when verifying the semantic behavior represented by the state transition diagram in Fig. 5, considering the actions and guard conditions, some errors were identified. The details are discussed in the model checking for timers presented later.

5.4 Comparison of Rule-Based Method Using Dependency Parsing and LLM-Based Transformation Method

Table 1 shows the Precision and Recall values for the conventional and proposed methods. The numerical values were calculated based on the conventional method, measuring the

Table 1: Comparison of accuracy between conventional and proposed methods

Method	Requirement		Transition conditions	Pre-states	Post-states
Rule-based	CD Player control panel	Precision	1.00	0.75	1.00
		Recall	0.80	0.75	1.00
ChatGPT		Precision	1.00	1.00	1.00
		Recall	1.00	1.00	1.00
Rule-based	Lock button	Precision	1.00	0.67	1.00
		Recall	0.70	0.67	1.00
ChatGPT		Precision	1.00	1.00	1.00
		Recall	1.00	1.00	0.83
Rule-based	Timer	Precision	0.57	0.80	0.80
		Recall	0.36	0.80	0.89
ChatGPT		Precision	0.83	1.00	1.00
		Recall	1.00	1.00	1.00

Table 2: Timer variables used in model checking

Variable name	Value Range	Meaning
state	{STOPPED, RUNNING, TIMEUP}	Timer Status
t	0..3600	Remaining timer time (sec)
button_state	{OFF, ON, LONG_ON}	Button Status
timer_button	boolean	Whether the button is pressed or not (event)
press_time	0..10000	Button press time (msec)
release_time	0..1000	Elapsed time since the button was released (msec)
buzzer	boolean	Buzzer on/off
buzzer_time	{0,50,100}	Buzzer sounding time (msec)
buzzer_count	{0,1,3}	Number of times buzzer sounds (times)
prev_t	0..3600	Variable for inspection (remaining time of one previous step)

extraction or conversion accuracy for one pair of states per sentence. A comparison of the results shows that the transformation accuracy using LLM exceeds the extraction accuracy of the rule-based method. Transition conditions showed higher accuracy compared to the rule-based approach. Additionally, it was confirmed that the transformation using LLM is more effective for complex and long requirement statements compared to the conventional method.

The differences between the proposed LLM-based transformation method and the rule-based method using dependency analysis are discussed below.

5.4.1 Rule-Based Method Using Dependency Parsing

- Advantages

In extraction, as long as there is no omission of the subject or object in the requirement statements, state variables and states can be extracted with high accuracy. This method is particularly effective for languages like English, where word order is important. Since open-source natural language processing libraries are often used as tools, there is almost no financial cost required for their implementation.

- Disadvantages

When dealing with Japanese requirement sentences, the extraction accuracy may decrease for long sentences with many modifiers or sentences that describe multiple consecutive state transitions in a single sentence. Additionally, when transition conditions are complex, it becomes difficult to clarify and extract the logical relationships between those conditions. Furthermore, the method is unable to handle sentences with improper grammar or bullet-pointed requirements. In this way, the extraction accuracy decreases for certain sentence structures, which is considered a barrier to practical usability. In terms of cost, creating extraction rules requires significant effort. Developing general and comprehensive rules is a time-consuming task. Additionally, in many cases, preprocessing is required before extraction. To create a state transition diagram, after extracting state transition descriptions using a rule-based approach, it is necessary to map the extracted elements onto the diagram. However, in requirement specifications, the same state can sometimes be assigned different names, making the automation of the mapping process difficult. For example, in the case of a timer, multiple expressions such as “停止” (Stop) and “起動していない” (Not Started) are used to describe the stop state. To determine that these refer to the same state, it is necessary to interpret the semantics using similarity measures, and thus, fully automating the

Table 3: Timer verification items and results

Timer Verification Items	Results
Whether the timer is in a stopped or active state, pressing the timer button for more than 100 milliseconds increments the remaining time by one minute, with the seconds unit resetting to 0 seconds.	**False**
If the timer button is pressed for less than 100 milliseconds, the remaining time is not incremented.	**True**
When the timer button is pressed for 100 milliseconds or more, the buzzer sounds for 50 milliseconds.	**True**
If the timer button is pressed for less than 100 milliseconds, the buzzer does not sound.	**True**
The timer value never falls outside the specified range.	**True**
When the timer value is 60min 0sec and the timer button is pressed for 100 milliseconds or more, the timer value resets to 1min 0sec.	**True**
The countdown begins exactly 1 second after the timer button is released.	**True**
While the timer is active and the remaining time is not zero, the remaining time decreases by 1 every second.	**False**
If no remaining time is set on the timer, it does not transition to the active state.	**False**
If the timer button is pressed and held for 3 seconds or more during operation, the timer value resets to 0min 0sec.	**True**
If the timer button is pressed and held for 3 seconds or more during operation, the buzzer sounds for 100 milliseconds.	**True**
When the timer button is held for 3 seconds or more outside of the active state, the timer value is not reset to 0min 0sec.	**True**
When the timer button is pressed for less than 3 seconds, the timer value does not reset to 0min 0sec.	**True**
After the set timer value elapses, the time-up buzzer sounds.	**True**

mapping process becomes a highly challenging task.

5.4.2 Conversion Method Using LLM

- Advantages

If the model is simple, it can be created with high accuracy. Unlike rule-based methods, it can also flexibly handle sentence structures such as long sentences and bullet points. While rule-based methods typically process one sentence at a time, LLMs convert the entire requirement sentence in one go, which is a key difference. Additionally, there is little to no cost required for implementation. In this way, compared to rule-based methods, the ability to easily create state transition diagrams enhances its practicality. If a state transition diagram can be created, model checking can be conducted. This allows for the detection and correction of errors, even if there are reliability issues with the LLM-based transformation.

- Disadvantages

Compared to rule-based methods, a disadvantage of LLM-based transformation is that there is variability in the output results. Additionally, elements of the state transition diagram, such as guard conditions and actions, may not be accurately transformed. Furthermore, transformations that require special notations, which are not explained during the prompt, are difficult to handle. For example, generating representations that use history states, When events, or pseudo-states like Choice or Junction used in conditional branches is currently not possible. However, it should be noted that these are also elements that are not achievable with rule-based methods.

Based on the above discussion, from the perspectives of transformation accuracy and ease of creation, it is believed that the use of LLM-based methods is preferable in software development.

5.5 Model Checking for Timers

To verify whether the state transition diagrams of the generated timers semantically represent the intended behavior, the state transition diagrams of the generated timers were model checked. In the experiment, NuSMV was used as the model checking tool [22]. In this experiment, the author manually wrote the transition model code for the timer based on the generated state transition diagram. Table 2 shows the timer variables used in the model checking. Additionally, the author identified possible verification items from the requirement statements and wrote them as verification formulas. There are 14 validation items in total. Table 3 shows these verification items. In the model checking, 17 verification formulas were described using LTL (Linear Temporal Logic) and CTL (Computational Tree Logic) formulas based on 14 created verification items. Table 4 shows the verification formulas and results. The correct state transition diagram for the timer, created in advance by ourself, satisfies all of these verification formulas. Therefore, if the state transition diagram created by ChatGPT is correct, all of these verification formulas should be satisfied. As a result, the generated state transition diagram for the timer failed to satisfy 4 out of the 17 verification formulas. In other words, in terms of verification items, it did not meet 3 out of the 14 requirements. The first verification item that was not satisfied was the process where “when the timer is in the stopped or active state, pressing the timer button for 100 milliseconds or more adds 1 minute to the remaining time.” The verification results indicated that the remaining time of the timer could not be set to more than 60 seconds. This occurred because the entry action for the stopped state was specified to reset the remaining time to 0min 0sec, causing the remaining time to be reset to 0 every time a self-transition led to re-entering the state. As a result, it was not possible to set the time, and there was no

Table 4: Timer verification formulas and results

Verification Equation	Results
$LTLSPEC\ G((state = STOPPED \mid state = RUNNING)\ \&\ button\_state = ON \&\ t\ !\ = 3600 \Rightarrow F(t = ((prev\_t/60) + 1) \times 60))$	T
$CTLSPEC\ EF(state = STOPPED\ \&\ 60 < t)$	F
$CTLSPEC\ EF(state = RUNNING\ \&\ 60 < t)$	F
$LTLSPEC\ G((state = STOPPED \mid state = RUNNING)\ \&\ press\_time\ < 100 \Rightarrow F(t = prev\_t))$	T
$LTLSPEC\ G!(t < 0 \mid t > 3600)$	T
$LTLSPEC\ G(t = 3600\ \&\ button\_state = ON \Rightarrow F(t = 60))$	T
$LTLSPEC\ G((state = STOPPED \mid state = RUNNING) \& button\_state = ON \Rightarrow F(buzzer\ \&\ buzzer\_time = 50\ \&\ buzzer\_count = 1))$	T
$LTLSPEC\ G((state = STOPPED \mid state = RUNNING) \& press\_time < 100 \Rightarrow F(!buzzer))$	T
$LTLSPEC\ G(state = STOPPED \& t > 0\ \&\ release\_time = 1000 \Rightarrow F(state = RUNNING))$	T
$LTLSPEC\ G(state = STOPPED \& t = 0\ \&\ release\_time = 1000 \Rightarrow X(state = STOPPED))$	F
$LTLSPEC\ G(state = RUNNING\ \&\ button\_state = OFF \& t > 0 \Rightarrow F(t = prev\_t - 1))$	T
$CTLSPEC\ EF(state = RUNNING\ \&\ 0 < t\ \&\ t < 60)$	F
$LTLSPEC\ G(state = RUNNING\ \&\ button\_state = LONG\_ON \& t > 0 \Rightarrow F(t = 0))$	T
$LTLSPEC\ G(state = RUNNING\ \&\ button\_state = LONG\_ON \Rightarrow F(buzze\ \&\ buzzer\_time = 100\ \&\ buzzer\_count = 1))$	T
$LTLSPEC\ G(state\ !\ = RUNNING\ \&\ button\_state = LONG\_ON\ \&\ t > 0 \Rightarrow F(t\ !\ = 0))$	T
$LTLSPEC\ G(state = RUNNING\ \&\ press\_time < 3000\ \&\ t > 0 \Rightarrow F(t\ !\ = 0))$	T
$LTLSPEC\ G(state = TIMEUP \Rightarrow F(buzzer\ \&\ buzzer\_time = 100\ \&\ buzzer\_count = 3))$	T

path where the remaining time of the timer would exceed 60 seconds in the future. The second unmet verification item was the process where "while the timer is active and the remaining time is not 0, the remaining time decreases by 1 every second (countdown)." This failure occurred because the requirement statement lacked a specific description of the countdown process. The third unmet verification item was the constraint that "if the remaining time is not set, the timer does not transition to the active state." This occurred because the guard condition $t > 0$ was not included in the transition from the stopped state to the active state. Despite the requirement stating, "After setting the timer value, the timer will start," this requirement could not be reflected in the state transition diagram, which is why it is considered an LLM transformation error.

Additionally, the results of the model checking revealed deficiencies in the timer's requirement statements.

1. Lack of requirements specifying the detailed countdown process (i.e., during operation, reducing the remaining time by 1 every second).

2. Lack of requirements defining the behavior of the buzzer when the timer button is pressed during the buzzer sounding in the timeout state (i.e., whether interrupt handling is allowed during the timeout buzzer sound).

For the second point, the model checking revealed that at the moment of timeout, if the timer button is long-pressed, the timeout buzzer and the buzzer action triggered by the long press could overlap. In this way, model checking can help identify errors in the created state transition diagram. Additionally, by closely examining the results, developers may be able to determine the causes of these errors. For example, in the timer of this experiment, the missing guard condition of t¿0 is an LLM transformation error, while the insufficient definition of interrupt handling during buzzer sound is an ambiguity in the requirement specification. Thus, by combining proposed method with model checking, developers can identify deficiencies and ambiguities in the requirement statements.

6 CONSIDERATION

One of the characteristics of modeling using ChatGPT is that there are almost no syntax rule errors in PlantUML. The experiment also confirmed that ChatGPT properly understands and processes Japanese words that represent logical relationships. For example, when a requirement sentence contains the words "or" to indicate a disjunction relationship, ChatGPT understands the relationship and reflects it appropriately in the state transition diagram. ChatGPT was also able to understand sentences that represent two transitions in a single sentence, such as "from state A to state B and then to state C," and output two transitions from a single sentence.

On the other hand, in the preliminary experiment comparing ChatGPT-3.5 and ChatGPT-4o, the quality of the generated state transition diagrams differed significantly. The difference in results was particularly apparent when no prompt was devised. This result suggests that GPT3.5 understands the PlantUML syntax rules for describing state transition diagrams, but does not understand how to create state transition diagrams from requirement statements. On the other hand, GPT4o can create a state transition diagram that satisfies the description of the requirement statement better than the dia-

gram generated by GPT3.5, even without any prompting devices. In both models, the use of chain of thought prompting to create state transition diagrams was effective. By using chain of thought prompting and providing a process of transformation patterns from requirement statements to PlantUML descriptions, ChatGPT can determine the elements of states, transitions, and events from requirement statements and reflect them in state transition diagrams. To create state transition diagrams that more accurately represent the intended semantics, it would be beneficial to use GPT4o in conjunction with chain-of-thought prompting.

The following challenges have been identified in the proposed method:

- While it is possible to extract action elements, it is necessary to reflect them in the state transition diagram while considering execution order and avoiding redundancy.
- For actions described within a state, it is important to distinguish between actions and activities and appropriately use notations such as entry or do.
- Accurately reflecting guard conditions and constraints involving numerical values, such as range limits.
- Representing the state transition diagram with an appropriate hierarchical structure.
- Utilizing special notations such as history states and When events.

In a state transition diagram, conditional actions and entry actions within a state are executed within the same cycle. Therefore, it is generally recommended to unify the processing for the same target or variable within either one of them. In the proposed method, the prompt explains the notation for both internal state actions and conditional actions. However, it does not provide guidance on how to distinguish between them. As a result, errors and redundancies related to action execution order have occurred In fact, errors caused by this issue were detected in the timer's model checking. If the generated state transition diagram contains such errors, it may be difficult to identify them at a glance. However, these errors in the generated results can be identified through model checking.

In order to obtain meaningful verification results through model checking, it is essential to reflect at least the variables required for the verification and the states that those variables can take in the state transition diagram. For the timer, essential variables include the timer state, the remaining time, the state of the timer button, the duration of the button press, and the buzzer. If there are difficulties in applying the proposed method and model checking in actual software development, it could be due to issues such as missing extraction of variables or their states, which may result in the inability to perform the necessary checks on the verification items. In the small model targeted in this experiment, no state omissions were observed, but caution will be needed in larger models. One possible countermeasure is to separate the extraction and conversion process to PlantUML. Variable names are generally found in the subject or object positions of the sentences [24]. By using this, first, the essential variables can be extracted, and then explicit instructions can be given in the prompt to incorporate them into the state transition diagram. Additionally, if the essential variables are already known, it would be better to explicitly instruct the prompt to incorporate them into the state transition diagram in advance.

If the cause of the error is an LLM conversion mistake, there is a possibility to correct that mistake afterward using prompt techniques. By re-inputting the generated result into the LLM and using prompt methods that evaluate its validity or select the correct logic, the LLM can check and correct the generated result [23]. This approach is expected to reduce errors due to conversion. Therefore, proposed method has potential for further extension.

The practicality of this method is then discussed. State transition diagrams are used in software development for purposes such as scenario analysis and implementation specifications [11]. This method is considered to be effectively applicable in scenario analysis during requirements definition and requirement analysis. This is because experimental results have shown that it can accurately extract states, transitions, and events from requirement statements and convert them into state transition diagrams. During the requirements definition phase, this method can be used to automatically create state transition diagrams from use cases and requirements. This helps in verifying the system's behavior and ensures that no states or events are overlooked. On the other hand, when using this method for implementation or analysis of specifications, additional manual adjustments to the state transitions created by ChatGPT would be necessary. This is because there may be errors or omissions related to action symbols of states and guard conditions of transitions that involve execution timing. In practice, during the model checking of the timer, there were cases where the requirements were not met due to issues with actions and guard conditions. On the other hand, combining this method with model checking enables the identification of errors in the generated state transition diagrams, as well as the detection of ambiguities and deficiencies in the requirements. Therefore, when using this method for specification implementation or analysis, it would be beneficial to integrate it with model checking. In this experiment, the creation of verification items and verification formulas was performed manually. However, accurately and comprehensively covering verification items for each requirement is a challenging and time-consuming task. Therefore, in the future, we aim to incorporate the use of LLMs to automate the generation of verification formulas from requirements. By leveraging LLMs to automate the creation of state transition diagrams, the output of verification formulas, their conversion into verification code, and the subsequent model checking, a cycle can be established. Repeating this cycle is expected to optimize both the state transition diagrams and the requirement statements.

The purpose of this study is to automatically create state transition diagrams from requirement statements and to check for missing or ambiguous requirements. Therefore, it is required that the state transition diagram be created only ac-

cording to what is described in the requirement statement. In this method, ChatGPT extracts the state transition description from the requirement statement and converts it to PlantUML format according to the conversion rules described in the prompt. In the process, there were few self-indulgent additions by ChatGPT to states, transitions, or events. Most of the time when a change or additional element is included by ChatGPT, it is considered to be a case where the status is omitted or not stated in the requirement statement. prior research on creating class diagrams using ChatGPT has indicated that having knowledge of the target problem domain is crucial for LLMs [10]. In contrast, the key to creating state transition diagrams lies in whether the requirement statements explicitly specify the elements of pre-state, post-state, and event. The requirement statements for the CD player operation panel, which were created with high quality, clearly included all these elements. Some related papers also recommend explicitly stating pre-state, post-state, and event in the requirement statements [25]. Conversely, if the requirement statements omit many details about the states, it becomes difficult to create high-quality state transition diagrams. This can lead to significant variations in the state transition diagrams generated by LLMs. Therefore, to create more accurate state transition diagrams in the future, it will be necessary to devise requirement templates that explicitly include elements such as pre-state, post-state, and events.

This experiment focused on models with a small number of states. In the future, experiments will be conducted on systems with a larger number of states and more complex structures. This will expand the applicability of the proposed method and verify its effectiveness in more practical software development processes.

7 CONCLUSION

This study used a large language model to generate state transition diagrams by converting requirement statements into PlantUML format. The Chain-of-thought prompting technique was employed as the prompting method, explaining the process of converting requirement statements into PlantUML. As a result of the experiment, it was confirmed that the state transition diagrams created using the proposed method correctly reflected the information related to states, transitions, and events. In addition, when comparing the proposed method with the traditional rule-based method using dependency parsing, it was concluded that the proposed method is superior in terms of conversion accuracy and ease of creation, making it more effective for generating state transition diagrams. On the other hand, when the generated state transition diagram was subjected to model checking, some omissions and misrepresentations were found in the detailed transition conditions and actions. However, by combining the proposed method with model checking, it was possible to identify errors in the generated state transition diagram and also confirm ambiguities or deficiencies in the requirements.

Acknowledgement

This research is being partially conducted as Grant-in-Aid for Scientific Research C (21K11826).

REFERENCES

[1] T. Tamai: “Fundamentals of Software Engineering,” Iwanami bookstore (2004).

[2] D. Aceituna, H. Do, and S.-W. Lee: “Interactive Requirements Validation for Reactive Systems through Virtual Requirements Prototype,” Model-Driven Requirements Engineering Workshop, Trento, Italy, pp. 1-10 (2011).

[3] J. Fitzgerald, and P. G. Larsen: “Modelling Systems: Practical Tools and Techniques in Software Development,” Cambridge University Press (2009).

[4] Y. Omori, and K. Araki: “Toward a Quality Improvement of Specifications in Natural Language Based on the Semi-equivalent Formal Models,” Journal of Information Processing Society of Japan, Vol.3, No.5, pp. 18-28 (2010).

[5] M. Onishi, S. Ogata, K. Okano, and D. Bekki: “A Method for Matching Patterns Based on Event Semantics with Requirements,” Proceedings of 14th International Joint Conference on Knowledge-Based Software Engineering (JCKBSE), pp. 181-192 (2022).

[6] M. Onishi, S. Ogata, K. Okano, and D. Bekki: “Reducing Syntactic Complexity for Information Extraction from Japanese Requirement Specifications,” Proceedings of 29th Asia-Pacific Software Engineering Conference (APSEC), pp. 387-396 (2022).

[7] N. Nakamura, R. Yamamoto, N. Yoshida, and H. Takada: “An Extraction of State Transition Descrptions in a Requirements Specification Document for an Embedded System,” IPSJ SIG Technical Report, vol.2018-SE-198, No.5, pp. 1-8 (2018).

[8] M. W. Anwa, I. Ahsan, F. Azam, and W. Haider: “A Natural Language Processing (NLP) Framework for Embedded Systems to Automatically Extract Verification Aspects from Textual Design Requirements,” 12th International Conference on Computer and Automation Engineering, pp. 7–12 (2020).

[9] M. Ohto, H. Ii, K. Okano, and S. Ogata: “Proposal of Extracting State Variables and Values from Requirement Specifications in Japanese by using Dependency Analysis,”Proceedings of the 25th International Conference on Knowledge-Based and Intelligent Information Engineering Systems, pp. 1649-1657 (2021).

[10] J. Cámara, J. Troya, L. Burgueño, and A. Vallecillo: “On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML,” Software and Systems Modeling (SoSyM), Vol.22, No.3, pp. 781-793 (2023).

[11] T. Kubo: “State Transition Design Techniques for Embedded Engineers - State Transition Diagram and State Transition Table description techniques for use in the field -,” TechShare Corporation (2012).

[12] A. M. Alashqar: “Automatic Generation of UML Diagrams from Scenario-Based User Requirements,” Jordanian Journal of Computers and Information Technology (JJCIT), Vol.07, No.02 (2021).

[13] A. Ferrari, S. Abualhaija, and C. Arora: “Model Generation with LLMs: From Requirements to UML Sequence

Diagrams," REW 2024 : Proceedings of the IEEE 32nd International Requirements Engineering Conference Workshops, Vol.24, pp. 291-300 (2024).
[14] B. Wang, C. Wang, P. Liang, B. Li, and C. Zeng: "How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts," 2024 IEEE International Conference on Software Services Engineering (SSE), pp. 249-257 (2024).
[15] S. Zhong, A. Scarinci, and A. Cicirello: "Natural Language Processing for Systems Engineering: Automatic Generation of Systems Modelling Language Diagrams," Knowledge-Based Systems, Vol.259, No.C (2023).
[16] M. Yamamoto, N. Yoshida, and H. Takada: "Survey of model extraction techniques based on embedded systems development artifacts," Computer Software, Vol.39, No.3 , pp. 4-16 (2022).
[17] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou: "Chain-of-thought prompting elicits reasoning in large language models," Advances in Neural Information Processing Systems (2022).
[18] Change Vision, Tutorial UML State Machine Diagrams - State Transition Tables, https://astah.change-vision.com/ja/tutorial/tut-state-machine-diagram-state-transition-table-cd.html (referred June 11, 2024).
[19] SESSAMI, Hot Pot, https://www.sessame.jp.
[20] GiNZA, https://megagonlabs.github.io/ginza/.
[21] M. Asahara, H. Kanayama, Y. Miyao, T. Tanaka, M. Omura, Y. Murawaki, and Y. Matsumoto: "Japanese Universal Dependencies Corpora," Journal of Natural Language Processing Vol.26, No.1, pp. 3-36 (2019).
[22] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri: "NUSMV: a new symbolic model checker," International Journal on Software Tools for Technology Transfer, Vol.2, pp. 410-425 (2000).
[23] X. Zhao, M. Li, W. Lu, C. Weber, J. H. Lee, K. Chu, and S. Wermter: "Enhancing Zero-Shot Chain-of-Thought Reasoning in Large Language Models through Logic," Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING), pp. 6144–6166 (2024).
[24] R. Gröpler, V. Sudhi, E. J. Calleja García, and A. Bergmann: "NLP-Based Requirements Formalization for Automatic Test Case Generation," 29th International Workshop on Concurrency, Specification and Programming Vol.2951, pp. 18-30 (2021).
[25] D. Aceituna, and G. Walia: "Model-based require-ments verification method: Conclusions from two con-trolled experiments," Information and Software Tech-nology Vol.56, No.3, pp. 321-334 (2014).

(Received: December 20, 2024)
(Accepted: April 30, 2025)

Takeki Ninomiya received his B.E. and M.E. degrees from Shinshu University, Japan, in 2023 and 2025, respectively. He has joined NTT COMWARE. His research area includes natural language processing.

Maiko Onishi received her M.S. degree in Computer Science from Ochanomizu University, Tokyo, Japan, in 2021. She completed the coursework for a Ph.D. at the same university in 2024. Her research interests include computational linguistics. In 2024, she joined Aisin Corporation, where she is currently engaged in developing systems for knowledge construction and acquisition.

Shinpei Ogata is an Associate Professor at Shinshu University, Japan. He received his B.E., M.E., and Ph.D. from Shibaura Institute of Technology in 2007, 2009, and 2012, respectively. He served as an Assistant Professor at Shinshu University from 2012 to 2020 and has been an Associate Professor there since 2020. He is a member of IEEE, ACM, IEICE, IPSJ, and JSSST. His current research interests include model-driven engineering for information system development.

Kozo Okano received his BE, ME, and PhD degrees in Information and Computer Sciences from Osaka University in 1990, 1992, and 1995, respectively. He was an Assistant Professor and an Associate Professor of Osaka University. In 2002 and 2003, he was a visiting researcher at the Department of Computer Science of the University of Kent in Canterbury, and a visiting lecturer at the School of Computer Science of the University of Birmingham, respectively. Since 2020, he has been a Professor at the Department of Electrical and Computer Engineering, Shinshu University. From 2023 to 2025, he served as the Director of Center for Data Science and Artificial Intelligence. From 2024 he has been the Vice-Director of Data Science Education Headquarters, Shinshu University. His current research interests include formal methods for software and information system design and applying deep learning to Software Engineering. He is a member of IEEE, IEICE, and IPSJ.