

広島大学 大学院 先進理工系科学研究科

# 多人数利用環境におけるスマートデバイス制御のための論理式作成支援

広島大学 大学院 先進理工系科学研究科

M223680

高橋朋也

指導教員 西村浩二



広島大学

Graduate School of Advanced Science and Engineering, Hiroshima University

# Logical Expression Tutoring System for Controlling Smart Devices in Multi-User Environments

Graduate School of Advanced Science and Engineering,  
Hiroshima University

M223680

Takahashi Tomoya

Supervisor      Nishimura Kouji



Hiroshima University

## 要 旨

多人数が行き来する環境において複数の人がスマートデバイスを共有する場合、インターネットショッピングなどのアプリケーションで資格情報漏洩のリスクを低減するためには、強固な認証および承認メカニズムを組み込むことが非常に重要である。そのため、論理式で表現したアクセス制御条件をスマートデバイスの管理者が事前登録することでアプリケーションのアクセス制御を設定できるシステムを開発した[1]。このシステムを利用するにはデバイスの管理者が関数の仕様書を基にアクセス制御条件を論理式として作成する必要がある。デバイスの管理者は必ずしもプログラマではなく、プログラミング等を学んでいない人も対象にしており、デバイスの管理者が必ずしも論理式を適切に作成するのに十分な IT スキルを持ち合わせていない現状がある。本研究では論理式作成に不慣れなデバイスの管理者がアクセス制御の設定を行えるように論理式作成支援を行う。難易度、正確性、作成時間の 3 つの観点でデバイス管理者を支援することで、デバイス管理者が抱える IT スキルの不足を解消し、スマートデバイスの安全な共有利用を促進することを目的とする。

論理式作成を支援する方法としてチャットボットに着目した[2]。チャットボットは、デバイス管理者との対話を通じて、適切な論理式を作成し、現在の論理式の状態を説明する役割を果たす。デバイス管理者は論理式と自然言語を相互に変換せずに、必要な情報をチャットボットのとの対話内で取得することで論理式が作成される。

情報科学部 2 年生 50 名を対象に、自分で論理式を作成する方法（従来の方法）とチャットボットを用いた方法（提案手法）の比較評価を行った。実施順序により 2 グループに分け、グループ A は従来の方法→提案手法の順で実施し、グループ B は提案手法→従来の方法の順に実施した。チャットボットを利用したアクセス制御の設定難易度について、76%の回答者が従来の方法よりも簡単だと感じていると回答した。チャットボットを用いたアクセス制御設定は難易度が低く、プログラミング未経験者にも利用しやすいと考えられる。提案手法の論理式作成の正解率は、従来の方法に比べ回答者全体で向上した（24.0%→70.0%）。また、提案手法を用いることで論理式作成にかかる時間も短縮できることが確認された。その一方で、チャットボットのユーザエクスペリエンスには改善すべき点がある。

従来の方法では関数リストを参照しながら論理式を手動で作成する必要があったが、提案手法では従来の方法に比べ、プログラミングに不慣れな管理者にも直感的で分かりやすく、かつ正確な論理式を作成できるようになる。時間的な制約がある場合も考慮し、アクセス制御条件の迅速な設定を可能である。

## Abstract

When multiple people share a smart device in an environment, it is very important to incorporate authentication and authorization mechanisms to reduce the risk of credential information leakage in applications such as Internet shopping. We have developed a system that allows smart device administrators to set access control for applications by registering access control conditions expressed in logical expressions [1]. To use this system, device administrators need to create access control conditions as logical expressions based on function specifications. Device administrators are not necessarily programmers, and the system is intended for people who have not studied programming, and device administrators do not necessarily have sufficient IT skills to create logical expressions appropriately. This study provides assistance in creating logical expressions so that device administrators who are not familiar with creating logical expressions can configure access control settings. By supporting device administrators in terms of difficulty, accuracy, and creation time, we aim to eliminate the lack of IT skills among device administrators and promote the safe shared use of smart devices.

We focused on chatbots as a method to support the creation of logical expressions [2]. Chatbots play the role of creating appropriate logical expressions and explaining the current state of logical expressions through dialogue with device administrators. The device administrator does not need to convert between the logical expressions and natural language, and the logical expressions is created by obtaining the necessary information in the dialogue with the chatbot.

We conducted a comparative evaluation of the method of creating logical expressions by oneself (conventional method) and the method using a chatbot (proposed method) on 50 university students studying Information Science. The methods were divided into two groups according to the order in which they were implemented. Group A was conducted in the order of conventional method → proposed method, while Group B was conducted in the order of proposed method → conventional method. Regarding the difficulty level of setting up access control using chatbots, 76% of the respondents answered that they found it easier than the conventional method. The difficulty level of setting up access control using chatbots is low, and it is considered to be easy to use even for inexperienced programmers. The percentage of correct answers for creating logical expressions using the proposed method was higher than that of the conventional method for all respondents (24.0% → 70.0%). It was also confirmed that the proposed method reduced the time required to create logical expressions. On the other hand, the user experience of the chatbot needs improvement.

Compared to the conventional method, which requires manual creation of logical expressions by referring to a function list, the proposed method is intuitive, easy to understand, and accurate even for administrators who are not familiar with programming. The proposed method also allows quick setting of access control conditions, taking into account time constraints.

# 目次

第1章	はじめに	1
1.1	研究の背景と目的	1
1.2	本論文の構成	2
第2章	多人数利用環境におけるスマートデバイス制御について	3
2.1	スマートデバイスの利用	3
2.1.1	スマートデバイス	3
2.1.2	多人数環境におけるスマートデバイスの利用	3
2.1.3	多人数環境におけるスマートデバイス利用時の課題	3
2.2	代表的なアクセス制御手法	4
2.2.1	ロールベースのアクセス制御 (Role-Based Access Control, RBAC)	4
2.2.2	属性ベースのアクセス制御 (Attribute-Based Access Control, ABAC)	4
2.2.3	コンテキストベースのアクセス制御 (Context-Based Access Control, CBAC)	5
2.3	論理式によるアクセス制御管理システムとさらなる課題	5
2.3.1	登場人物	5
2.3.2	論理式によるアクセス制御管理システム	5
2.3.3	論理式作成時の課題	7
第3章	チャットボットを用いた論理式作成支援システム	8
3.1	提案手法	8
3.2	チャットボット作成に関連するサービス	9
3.2.1	Amazon Lex	10
3.2.2	AWS Lambda (Lambda 関数)	11
3.2.3	Amazon DynamoDB	12
3.2.4	Slack	12
3.3	チャットボット的设计	12
3.3.1	チャットボットのフロー	13
3.3.2	インテントとスロット	14
3.4	論理式作成について	17
3.4.1	論理式作成	17
3.4.2	自然言語作成	17
3.4.3	選択肢作成	18
3.4.4	データベース設計	18
第4章	実装例	21
4.1	スマートスピーカの買い物アプリのアクセス制御	21
4.1.1	チャットボットとの対話	22
4.1.2	論理式作成の流れ	24
第5章	評価	25
5.1	評価方法	25
5.1.1	従来の自分で論理式を記述する方法との比較実験	25
5.1.2	評価項目	26
5.1.3	測定項目	28
5.2	結果	28

5.2.1 アンケート結果	28
5.2.2 測定結果	30
5.3 考察	34
5.3.1 チャットボットのユーザエクスペリエンス	34
5.3.2 チャットボットの論理式作成の正解率	34
5.3.3 チャットボットの論理式作成にかかる時間	34
5.3.4 チャットボットの難易度	35
5.3.5 よくある誤答と改善方法	35
5.3.6 チャットボットを用いる方法の正解パターンとそこに至るまでの流れ	36
5.3.7 提案手法で改善できるターゲット	36
5.3.8 生成 AI による論理式の最適化	36
第 6 章 まとめ	38
6.1 まとめ	38
6.2 今後の課題	38
謝辞	39
参考文献	40
学会発表実績	40
付録	41
A1. 論理式作成チャットボット 実装手順	41
A2. 論理式支援システムソースコード	42

# 第1章 はじめに

## 1.1 研究の背景と目的

近年、ネットワーク技術の進歩により、高速かつ安定したインターネット接続が実現され、これに伴いスマートフォン、タブレット、スマートウォッチ、スマートスピーカなどの多様なデバイスが広く普及し、日常生活に深く組み込まれている。これらのスマートデバイスは通話やメッセージングだけでなく、写真や動画の撮影、ソーシャルメディアの利用、ゲームのプレイ、音楽の再生など、様々な活動に利用され、その応用領域は急速に広がっている。また、教育や医療などの分野でも幅広く活用され、オンライン教育や患者のモニタリングなどが可能になり、個々のニーズに合わせた教育や医療が提供され、利用者の利便性が向上している。

スマートデバイスの中には複数人で共有するものも登場しており、共有型デバイスにおいては、デバイスの管理者がデバイスの置かれる環境や利用するユーザに応じて適切なアクセス制御を行う必要がある。例えば、共有型スマートスピーカなどでは、複数のユーザが同一のデバイスを利用し、個々のプライバシーを守りつつ効果的にデバイスを使用する必要がある。このため、デバイスの管理者は各ユーザに適切なアクセス権を与え、同時に個別の利用制限やカスタマイズも行う必要がある。特に音声認識機能を備えたデバイスでは、個々の利用者ごとに異なる音声プロフィールやアクセス許可が不可欠である。さらに、異なるアプリケーションや機能に対する利用制限だけでなく、セキュリティやプライバシーの観点からもデバイスの管理が重要である。学校環境では生徒の個人情報の適切な管理が求められ、デバイスが適切に利用されるためのアクセス制御が必要である。アプリケーションの開発者ではなくデバイスの管理者が、デバイスやアプリケーションから取得できる情報とデバイス周辺から得られる情報を組み合わせたアクセス制御条件を設定できるシステムを開発した[1]。このシステムでは、デバイスの管理者が属性やグループや利用条件を論理式として表現し、アクセス制御を行う。論理式によるアクセス制御は、論理演算子を使用して細かな条件を組み合わせ、柔軟にアクセス条件を設定できる利点がある。これにより、セキュリティ向上が期待でき、かつ条件の変更が容易であるため、セキュリティ要件の変更にも迅速に対応できる。

適切なアクセス制御条件を設定するには論理式の仕組みや関数の知識などのITスキルが必要になるが、デバイスの管理者が必ずしも論理式を適切に作成するのに十分なITスキルを持ち合わせていない現状がある。例えば教育現場では、生徒が自分の権限でこのアプリを自由に使用できるようにするには、教師がデバイスの管理者として論理式を設定する必要がある。論理表現設計の自由度は教師の持つITスキルによって大きく異なり、教員がアクセス制御条件を設定する際には、情報セキュリティ対策への対応を考慮する必要がある。しかし、このような教員が専門的な知識を身につけるような取り組みは、教員の負担を増加させる可能性がある指摘されている[3]。この例のようにデバイスの管理者は必ずしもプログラマではなく、プログラミング等を学んでいない人も対象にしている。また、日本の高校生を対象に論理的思考力を評価する研究が行われており、「and」や「or」といった論理コネクタの意味を十分に理解していない受験者もいることが分かっている[4]。そのため、プログラミングに不慣れなデバイスの管理者が適切なアクセス制御条件を設定するには直感的で使いやすいインターフェースが不可欠である。

このような課題に対処するために、本研究ではデバイスの管理者が柔軟かつセキュアに設定を行える

ように、多人数利用環境におけるスマートデバイス制御のための論理式作成支援システムを開発し、その実装と効果に焦点を当てる。共有型デバイスの普及が進む中で、デバイスの管理者が抱える IT スキルの不足を解消し、効果的なデバイス管理が実現できる仕組みを提案することで、スマートデバイスの安全な共有利用を促進する。

## 1.2 本論文の構成

本論文は以下のような構成になっている。

2 章では多人数環境におけるスマートデバイスの利用と課題、その課題を解決する論理式によるアクセス制御管理システムとさらなる課題を述べる。3 章では提案手法のシステム設計を述べる。4 章では全体の流れをひとつのシナリオを用いて説明する。5 章ではシステムの評価結果と考察を述べる。最後に 6 章で本研究のまとめを述べる。



## 第2章 多人数利用環境におけるスマートデバイス制御について

### 2.1 スマートデバイスの利用

#### 2.1.1 スマートデバイス

総務省の令和4年通信利用動向調査の結果[5]によれば、モバイル端末の世帯保有率は9割以上に達し、WEBサービスの利用がますます身近なものとなっている。2019年12月に新型コロナウイルスが発生し、感染予防のために三密回避が重要とされ、ウェブ会議やキャッシュレス決済などのWEBサービスが急速に普及した。音声アシスタントは、そんなWEBサービスの普及において重要な役割を果たしている。Amazon Alexa, Google Assistant, AppleのSiriなどがその代表例で、これらはユーザの音声を受け取り、様々な指示に応じて情報提供やタスクの実行を行う。

更に、IoT (Internet of Things) との統合により、これらの音声アシスタントはスマートデバイスと連携し、ユーザが簡単に家庭やオフィスの環境を制御できるようになった。照明、温度調節、セキュリティシステムなどの操作が可能である。また、リモートからの監視や制御が可能で、センサを駆使して異常を検知し、ユーザに通知するといった使い方もできる。

これらのテクノロジーにより、ユーザは日常生活をより便利で快適に過ごせるようになりつつある。また、IoTへのニーズの高まりから、様々なスマートデバイスの普及が予想される。

#### 2.1.2 多人数環境におけるスマートデバイスの利用

スマートデバイスの中には複数人で共有されるデバイスも登場した。具体的には、複数ユーザ機能を持つAndroid端末[6]や、複数人の音声情報を登録し音声を識別することが可能なスマートスピーカ[7]が普及し始めている。特にスマートスピーカのような複数人の世帯で使用されるスマートデバイスはプライバシーや同意に関する研究の対象となっている[8]。現状のスマートスピーカでは、事前に登録する音声IDによる声の識別によるパーソナリティ化や音声によるPINコードによる認証が行われている。最新のカメラ付き端末では、ビジュアルIDによる顔画像による識別も行われている。

現在、共有型のスマートデバイスは主に家庭などのプライベートな空間で利用されているが、今後は公共の場やビジネス環境でもその利用が拡大する可能性がある。そのような不特定多数の人物が行き来する場をこの論文では多人数環境と表現する。多人数環境での共有型スマートデバイスは会議室でのプレゼンテーションやグループディスカッションにおいて、参加者がリアルタイムで情報を共有しやすくする役割を果たし、教育機関や図書館などでは複数のユーザが同じデバイスを利用して学習や情報検索が行える環境を作ることができる。これにより、リソースの有効活用が可能となり、共有型スマートデバイスが知識共有や学習支援の手段として広く普及する可能性がある。その他にも、共有型スマートデバイスは公共交通機関や商業施設での利用において、情報提供や案内・エンターテインメントなどの向上に利用されることが考えられる。

#### 2.1.3 多人数環境におけるスマートデバイス利用時の課題

共有型のスマートデバイスは私たちの日常生活において重要な役割を果たす可能性がある一方で、複数のユーザが同じデバイスを使用する場合、アクセス権の管理が複雑になる。特に機密性の高い情報やプ

ライバシーに関する情報へのアクセス制限を確立することが重要になる。現在、プライベートな環境で使用されているスマートデバイスの一例であるスマートスピーカは、ユーザの音声による生体情報を用いた識別や、PIN コードの発話による認証が行われている。しかし、これを公共の場で使用する場合、セキュリティおよび運用上の問題が発生する可能性がある。大学でスマートスピーカを使用する際には、構成員の音声情報を保持することや、公共の場でPIN コードを発話することがセキュリティ上望ましくない。また、デバイスの使用環境によって大きく影響を受けるため、アクセス制御条件の設定はアプリケーションの開発者ではなくデバイスの管理者が適している。

## 2.2 代表的なアクセス制御手法

多人数環境においては、複数のユーザが同じデバイスを使用し、来客者などの利用登録していない人の出入りも考えられるため、アクセス権の管理が重要になる。ここでは代表的なアクセス制御手法についてまとめる。ここで挙げる手法以外にも多くのアクセス制御の手法があり、組み合わせて使用されることも多い。

### 2.2.1 ロールベースのアクセス制御 (Role-Based Access Control, RBAC)

RBAC はユーザに役割(ロール)を割り当て、その役割に基づいてアクセス許可を管理する仕組みである。ユーザのロールに基づいて条件を設定し、アクセスを制御する。ユーザのグループや属性に基づいて、異なるロールが定義される。例えば、一般ユーザ、管理者、財務担当者などが異なるロールに対応する。各ロールには、特定のアクセス権や機能が割り当てられる。例えば、管理者は全ての機能にアクセスできるが、一般ユーザは限定的な機能しか利用できないといった条件を作成できる。ロールベースのアクセス制御では、ユーザには必要な最小限のアクセス権しか与えない原則が重視され、セキュリティの向上と誤った操作のリスクの軽減が図られる。ユーザが新しい役割に就いた場合、管理者はユーザのロールを変更することでアクセス権を効率的に制御できる。

### 2.2.2 属性ベースのアクセス制御 (Attribute-Based Access Control, ABAC)

ABAC はユーザ、リソース、状況を考慮してアクセス許可を決定する。ABAC の主要素を以下に示す。

- ・ 主体 (Subject) : アクセスを要求するユーザ。
- ・ 対象 (Resource) : アクセス対象となるリソースやデータ。
- ・ アクション (Action) : ユーザが対象に対して行うアクション (例: 読み取り, 書き込み)。
- ・ 環境 (Environment) : アクセス要求が行われる状況や環境の条件。

これらの要素に対して、属性が割り当てられ、これらの属性の組み合わせに基づいてアクセス許可が決定される。例えば、ユーザが所属している組織やグループなどが考慮されアクセスの可否が決定する。RBAC よりも柔軟性があるが、管理が難しい特徴がある。堀らの研究では、属性ベースのアクセス記述言語の XACML (eXtensible Access Control Markup Language)を用いて教育現場の生徒や教師といった属性をもとに閲覧制限を実現している[9]。また、属性ベースのアクセス記述言語を用いた学校間連携を考慮したアクセス制御の研究が行われている[10]。

### 2.2.3 コンテキストベースのアクセス制御 (Context-Based Access Control, CBAC)

CBACでは、ユーザの属性だけでなく、コンテキスト（状況や環境の情報）も考慮する。コンテキストの例を以下に示す。

- ・ ユーザコンテキスト (User Context) : ユーザに関する情報。ユーザのアイデンティティ、ロケーション、デバイス、時間帯などが含まれる。
- ・ リソースコンテキスト (Resource Context) : アクセスされるリソースに関する情報。リソースのセンシティブな性質、所在地、セキュリティポリシーなどが含まれる。
- ・ アクションコンテキスト (Action Context) : 実際に行われるアクションに関する情報。操作の種類、実行者の目的などが含まれる。
- ・ 環境コンテキスト (Environment Context) : システム全体の状況や外部環境に関する情報。ネットワークの状態、脅威のレベルなどが含まれる。

これらのコンテキスト情報を総合的に評価し、アクセス制御の判断が行われる。

## 2.3 論理式によるアクセス制御管理システムとさらなる課題

### 2.3.1 登場人物

ここで、本論文に登場する人物の役割を以下に示す。

- ・ ユーザ : スマートデバイスを利用するユーザ。学校でスマートデバイスを利用する場合、教員や学生がユーザにあたる。
- ・ デバイスマネージャー : スマートデバイスの管理者。アクセス制御条件を設定する。学校でスマートデバイスを利用する場合、教員がデバイスマネージャーにあたる。
- ・ ユーザ管理者 : ユーザの利用登録を行う管理者。ユーザの属性やグループの登録や変更を行う。学校でスマートデバイスを利用する場合、教員がユーザ管理者にあたる。
- ・ アプリ開発者 : アプリケーションの開発者。

### 2.3.2 論理式によるアクセス制御管理システム

デバイス周辺の人数、彼らの意図、対人関係などの外部情報をアクセス制御の条件に統合することで、周囲の状況に応じてアプリケーションのアクセス制御を適切に行うことが可能になる。複数人が1台のデバイスを共有する場合の運用上およびセキュリティ上の課題を解決するために、外部情報と連携してスマートデバイスのアプリケーションを制御し、デバイス周辺の環境に適応したアクセス制御を可能にするシステムを開発した[1]。参照元では外部連携システムと呼んでいるが、本論文では論理式によるアクセス制御管理システムと呼ぶ。また、参照元ではスキルと呼んでいるが、本論文ではアプリケーション（アプリ）と呼ぶ。論理式によるアクセス制御システムの概要図を図1、図2に示す。論理式は論理式処理部で判定され、論理式処理部はアプリケーションからAPIを通じて呼び出され、論理式をアプリ/ユーザ管理DBから取得し、条件判定を行い、結果をアプリケーションに返す。論理式は真理値が確定した段階で判定を終了する。条件判定中に外部情報が必要になった場合、外部連携を行い、外部情報を取得する。アプリ/ユーザ管理部はユーザ/管理DBを操作するインターフェースであり、ユーザ管理者はユーザの利用登録を行い、デバイスマネージャーは論理式の登録を行う。論理式を用いたスマートデバイスのアクセス制御では、属性やロールを関数で表すことでユーザごとに異なる権限を与えることができる。また、コンテキ

スト情報も関数で表現することが可能なため、デバイスの状況やデバイスの置かれる環境に適応したアクセス制御条件を組み込むことができる。デバイス管理者が外部情報の組み合わせを論理式として設定することで高度な認証と承認機能を選択的に追加することができる。図3の例では、買い物アプリにおいて商品選択から商品購入の間に音声IDやPINコードの認証が行われている(図3(a))。プライベートな空間ではこれらの認証で問題ないが、多人数環境になると盗聴や生体情報の登録等セキュリティ上や運用上の問題が発生する。そこで、論理式によるアクセス制御管理システムを導入することで、スマートスピーカの買い物アプリにおいても、ID・パスワードによる認証や承認機能を追加することができる。この際、デバイス管理者は利用できる関数がまとめられた関数リストやアプリケーションの情報を参考にしながら論理式を作成する。関数リストを表1に示す。利用できる情報を以下に示す。

- ・ ユーザ情報：ユーザ管理者が利用者登録したユーザの属性やグループなどの情報。承認要求の関数にも含まれる。
- ・ アプリ情報：アプリケーションから取得できる固有の情報。例えば、購入機能があるアプリケーションでは「購入リスト」や「購入金額」などがアプリ情報に含まれる。
- ・ 外部情報：人数情報や姿勢情報等、ユーザ情報やアプリ情報以外のデバイス周辺のセンサやデータベース等を参照することで得られる情報。新しい関数を作成することで、機能を拡張することができる。

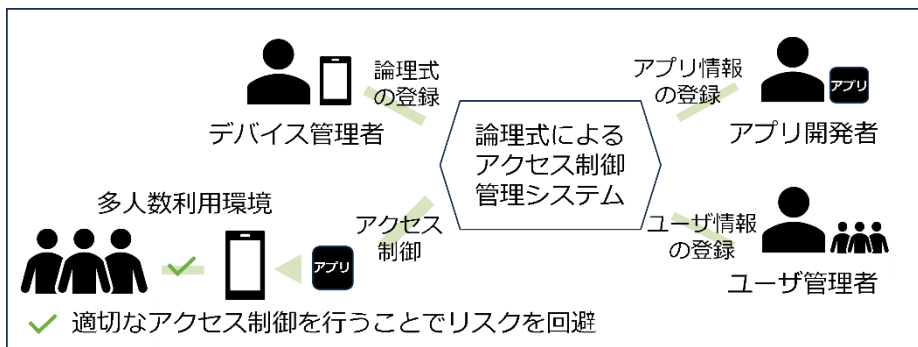


図1 論理式によるアクセス制御システムの概要(1)

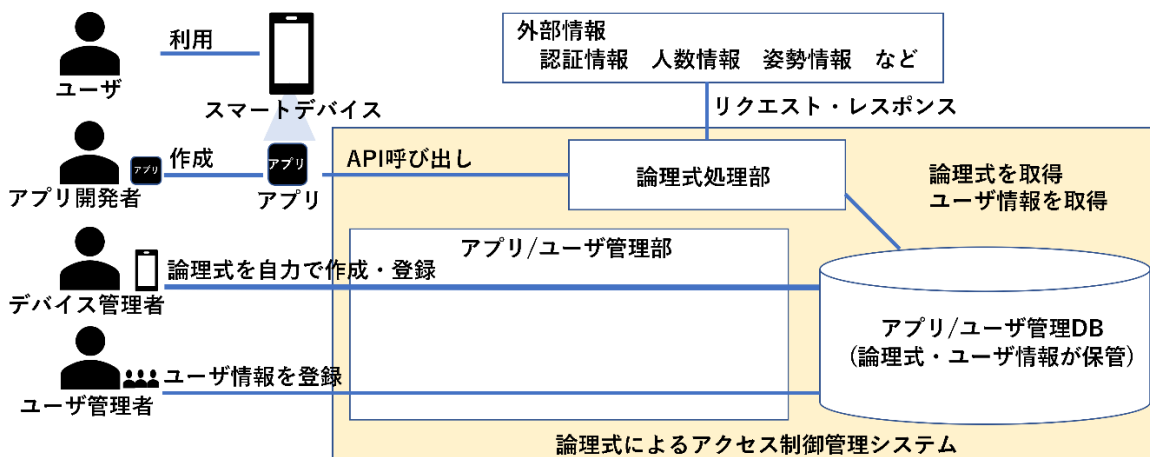


図2 論理式によるアクセス制御システムの概要(2)

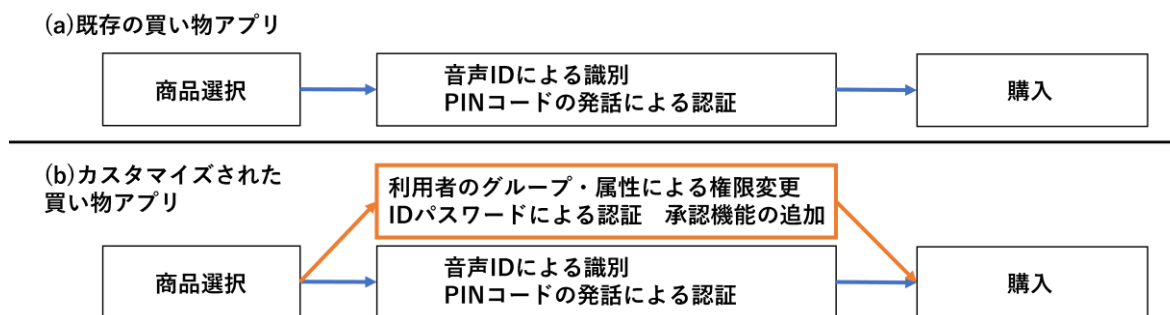


図3 論理式によるアクセス制御システムを導入した買い物アプリの変化

表1 関数リストの例

関数の形	引数	説明
Auth(引数1, 引数2, 引数3)	引数1 “利用者”または “承認者” 引数2 グループ名 引数3 属性名	引数1が“利用者”の場合、利用者のグループと属性が引数2と3で指定したグループと属性に一致していたら TRUE を返す 引数1が“承認者”の場合、引数2と3で指定したグループと属性に一致している人物に承認要求を行い、承認されたら TRUE を返す
Time(引数1, 引数2)	引数1 下限時刻 引数2 上限時刻	現在の時刻が引数1と引数2で指定した時刻の間なら TRUE を返す
Day ()	引数無し	今日の曜日[月, 火, 水, 木, 金, 土, 日]を返す

### 2.3.3 論理式作成時の課題

関数リストを参照して論理式を作成しアクセス制御を設定することは柔軟性や拡張性という点で優れているが、論理的思考力がないと十分ないと適切なアクセス制御条件を設定できないという課題がある。論理式を作成するためには、使用する関数や論理演算子についての深い理解が必要である。論理式を構築するためには、引き数の数とそれらの相互関係を理解し、「and」や「or」などの演算子を組み合わせる能力が求められる。また、設定するアクセス制御条件によっては、単純な論理式ではなく、複雑な組み合わせや条件が必要な場合がある。そして、設定したいアクセス制御条件を一般化し、抽象化して論理式に落とし込む思考力が求められる。

解決策として、問題解決の練習を通じて論理式の作成に慣れることが考えられる。ただし、今回対象としているデバイス管理者は学校の教員や企業の職員であり、時間やリソースには制約がある可能性があるため、デバイス管理者が容易にアクセス制御を設定・変更できる使いやすいツールが必要である。

## 第3章 チャットボットを用いた論理式作成支援システム

### 3.1 提案手法

本研究ではデバイス管理者が柔軟かつセキュアに設定を行えるように、多人数利用環境におけるスマートデバイス制御のための論理式作成支援を行う。従来の論理式作成方法は、関数の引数や戻り値などの説明が書かれている関数リストを見ながら自分で論理式を記述するものである。プログラミングに不慣れたデバイス管理者に対して、従来の自分で論理式を記述する方法よりも直感的にわかりやすく正確な論理式の作成ができるように支援する。また、デバイス管理者は教員のような時間的な制約がある場合もあるため、アクセス制御条件の設定の時間も短い方が好ましい。これらのことから、難易度、正確性、時間の3つの観点でデバイス管理者を支援するため、チャットボットを導入した[2]。このアプローチにより、従来の論理式の作成プロセスにおいて生じる課題を解決することが期待される。チャットボットは、ユーザとの対話を通じて、適切な論理式を作成し、現在の論理式の状態を説明する役割を果たす。ユーザはチャットボットと直感的な対話を行いながら、論理的な構造や制約条件を明確にしていくことができる。ユーザは途中で条件を変更したり、新たな情報を追加したりする際にも、チャットボットを介してシームレスに対話を進めることができる。このシステムの導入により、論理式の作成プロセスが効率化され、ユーザが複雑な論理的な構造を容易に構築できる。論理式作成支援システム全体を図4、図5に示す。チャットボットにはAI型とシナリオ型の2通りがあるが、今回はシナリオ型を採用した。デバイス管理者はSlackを通してチャットボットと対話をしながら論理式を作成する。チャットボットは、会話を通じて論理式の作成を支援し、質問や選択肢を提示することで、デバイス管理者が論理的な条件を直感的に設定できるようにサポートする。同時に、チャットボットがリアルタイムにフィードバックを提供することで、デバイス管理者は状況に応じて調整を行い、誤った論理表現を防ぐことができる。

デバイス管理者が論理式作成をする際の難しさが、自然言語を論理式に変換する部分と作成した論理式の関数の動作や順序を理解する部分にあると仮定し、以下の3つの点を支援するように設計した。第一に、自然言語で論理表現を表現することで、デバイス管理者の理解を助ける。第二に、各機能の優先順位を示すことで、「or」と「and」の影響範囲を理解する手助けを行う。同時に満たすべき条件を横に並べて、一つでも満たせばよい条件を縦に並べる。最後に、関数や引数を選択肢として表示する。選択可能な関数や引数の選択可能なものをデータベースから抽出し表示する。関数の引数の数や変数の型を意識せずに正確な関数を作成することが可能になる。これらの要素をサポートすることで、デバイス管理者は論理式の構造を理解していなくても、論理式を作成するのに必要な情報を自然言語で取得することで、論理式の作成を自動化できる。

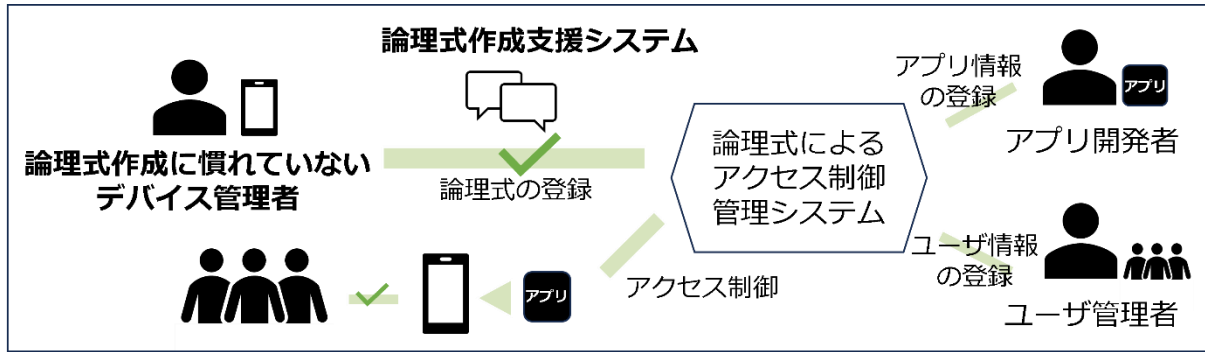


図4 論理式作成支援システム概要(1)

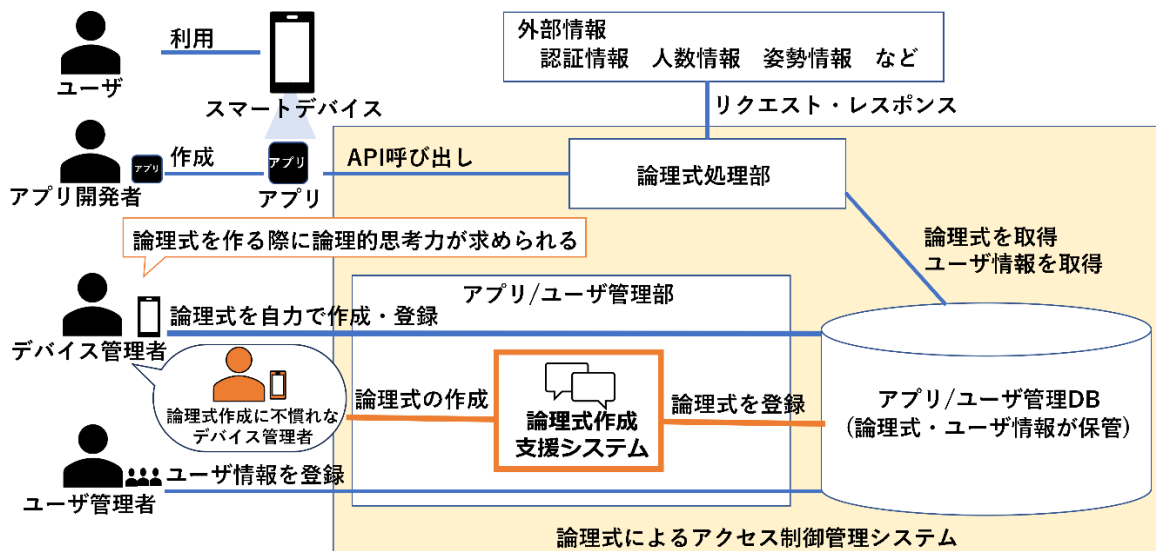


図5 論理式作成支援システム概要(2)

### 3.2 チャットボット作成に関連するサービス

本研究では論理式作成を支援するシステムにチャットボットを導入した。ここでは、チャットボットの開発プロセスと図6に示す利用したサービスについて述べる。

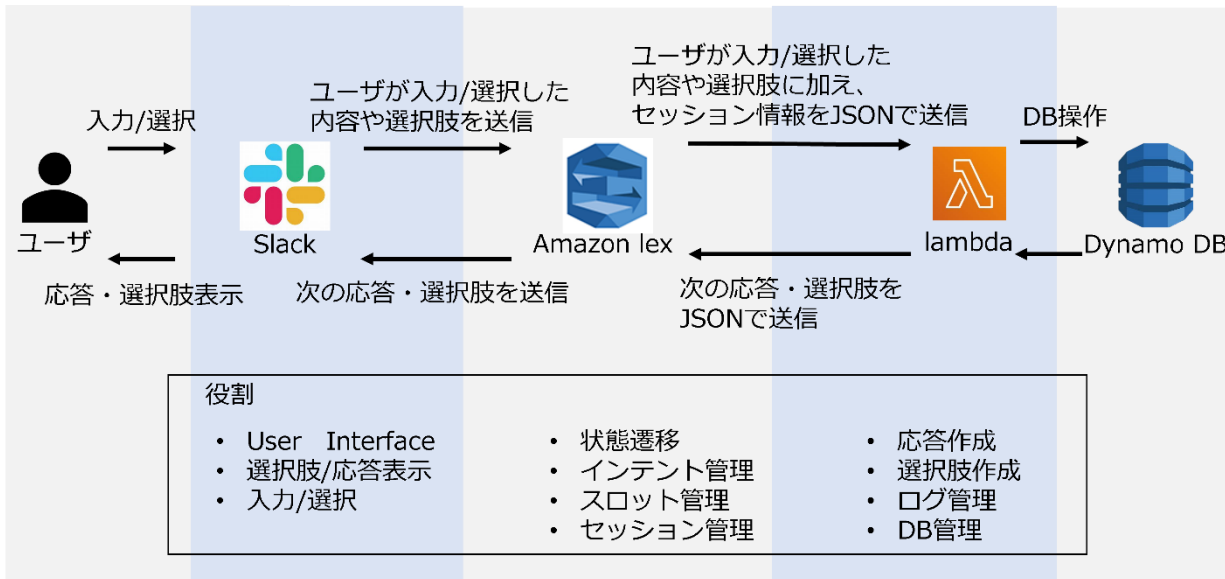


図6 チャットボット作成に関連するサービス

### 3.2.1 Amazon Lex

Amazon Lex は、Amazon Web Services (AWS) が提供するサービスの一つで、機械学習を活用して会話型のアプリケーションやチャットボットを構築するためのプラットフォームである[11]. Amazon Lex は、自然言語理解 (NLU: Natural Language Understanding) エンジンを搭載しており、ユーザの発言や入力を理解しやすい形に変換する。これにより、ユーザとの対話をより自然で効果的なものにすることが可能である。テキストベースのチャット対話や音声ベースの対話のインターフェースが利用できる。Amazon Lex で構築したボットは、Web アプリケーション、モバイルアプリ、メッセージングプラットフォーム、スマートスピーカなどで利用可能である。チャットボットの開発者は、Amazon Lex を使用して独自のカスタムインテントやスロットと各インテントの状態遷移を定義する。

- **インテント (Intent) :** ユーザが実行したい特定のアクションや目的を表す。ユーザが対話の中で行いたい要求や操作に対応する。例えば、「商品の選択」、「商品の購入」、「商品の検索」などがインテントの例である。開発者はこれらのインテントを事前に定義し、それに基づいて対話の遷移を設計する。
- **スロット (Slot) :** インテントに関連付けられた情報を取得するためのパラメータ。インテントがアクションや目的を表すのに対し、スロットはそのアクションや目的に必要な具体的なデータを指定する。例えば、商品名、購入数、色などがスロットになる。

ユーザとの対話の流れでは、図7のようなプロセスが発生する。まず、ユーザが対話の中で何を言おうとしているかを理解するために、Amazon Lex は自然言語処理を使用してユーザの発言を解釈する。解釈された発言から、ユーザが何をしたいのか、どのアクションや目的が該当するのかを判定し、対応するインテントを特定する。インテントが特定されたら、そのインテントに必要な情報を取得するためにスロットを抽出する。ユーザが提供した具体的なデータがスロットに格納される。インテントとスロットが抽出されたら、それに基づいて対応するアクションや処理を実行する。例えば、購入の場合は購入を確定するなどである。ただし、アクションの実行及び応答の作成については、Lambda 関数と連携している場合に



は細かい処理を Lambda 関数内で行うこともできる。

Amazon Lex の開発プロセスを図 8 に示す。まず、Amazon Lex コンソールや API を使用して、ボットに対応するインテントを作成する。次にインテントに関連付けられたスロットを定義し、それに対するユーザの入力を収集する。次にボットがユーザの発話を理解できるように、インテントごとにサンプルの発話を追加する。これにより、ボットが異なる表現や言い回しにも対応できるようになる。インテントがトリガーされた際に実行される処理やビジネスロジックを定義するために、Lambda 関数と連携する。インテント、スロット、発話文などの構成が完了したら、ボットをビルドしてデプロイする。デプロイが完了したらボットを作成し、エンドポイントを設定する。Amazon Connect, Facebook Messenger, Slack など、様々なプラットフォームでボットを利用可能にするためのインテグレーションが提供されている。

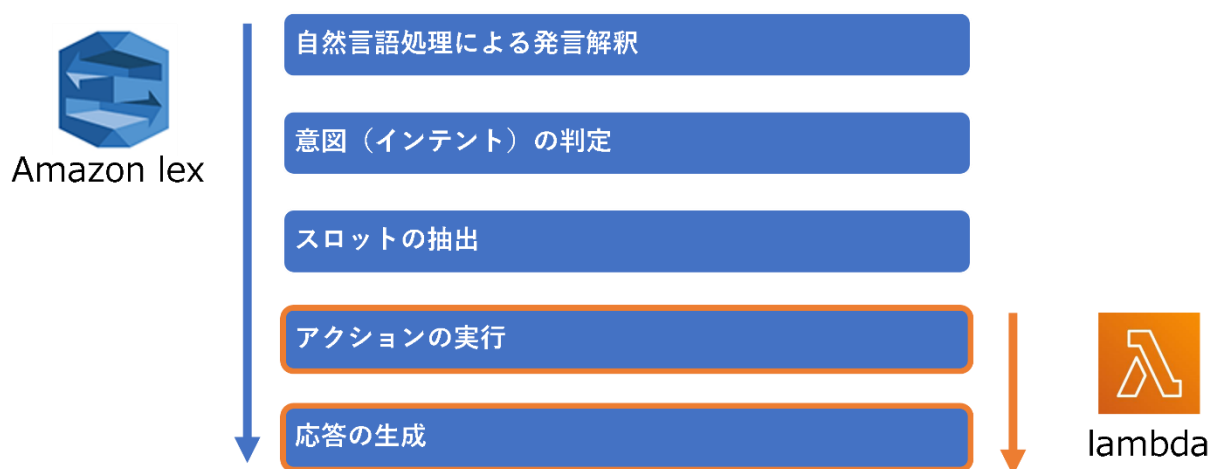


図 7 Amazon Lex のユーザとの対話プロセス

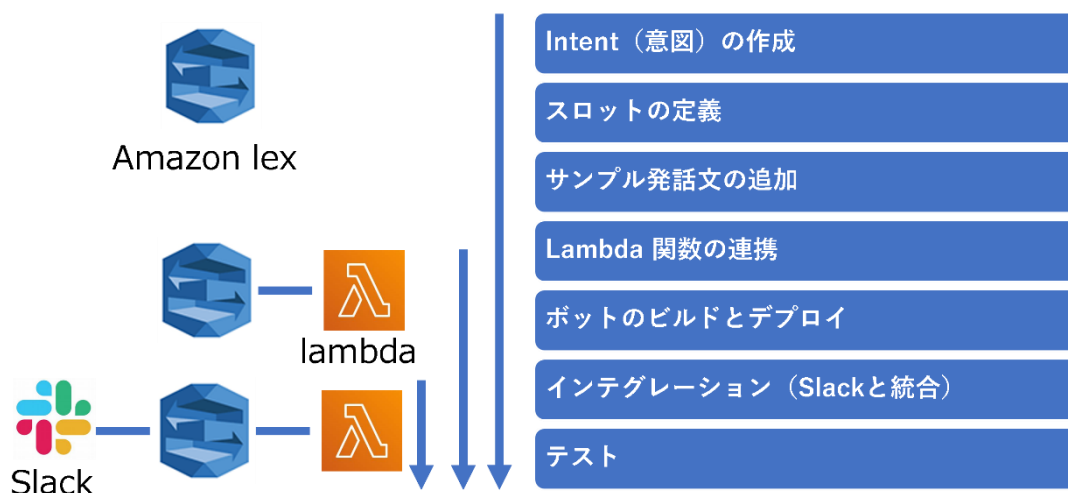


図 8 Amazon Lex の開発プロセス

### 3.2.2 AWS Lambda（Lambda 関数）

Amazon Lex のバックエンドでは Lambda 関数が動いており、選択肢の作成やデータベースの処理を行

っている。Amazon Lex が特定の意図やスロットに基づいて Lambda 関数をトリガーすると、Lambda 関数が実行されボットの特定のロジックやデータ処理が行われる。Lambda 関数からの結果が得られたら、それをもとに Amazon Lex がユーザに対して応答する。

### 3.2.3 Amazon DynamoDB

Amazon DynamoDB は、AWS が提供するフルマネージドな NoSQL(ノンリレーショナルまたは Not Only SQL) データベースサービスである[12]。NoSQL データベースは、ドキュメント指向、キー値、グラフデータベースなど、さまざまなデータモデルや用途に特化した種類があるが、DynamoDB のデータモデルでは、各アイテムが主キーによって特定されるキー値型のデータベースである。図 9 に示すように、主キーは Partition Key と Sort Key から構成される。Partition Key はアイテムの分散に使用され、Sort Key はパーティション内でアイテムをソートするために使用される。主キー以外の Attribute は Item 間で不揃いでも問題ない。Lambda 関数内で主キーを指定することでデータベースの操作を行うことができ、AWS の各種サービスとの連携が容易なため採用した。

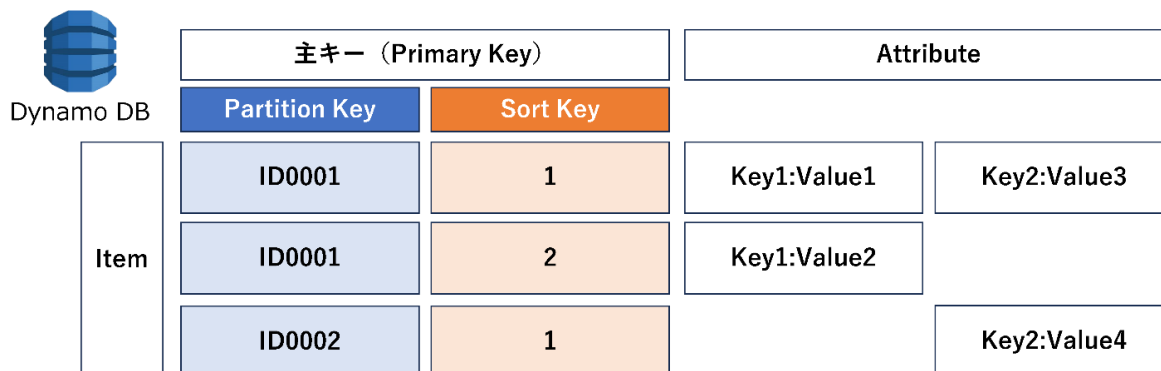


図 9 Dynamo DB の仕組み

### 3.2.4 Slack

Slack はビジネス向けのコミュニケーションプラットフォームで、チームや組織がメッセージングやファイル共有を行うためのツールである。Slack 上で Amazon Lex を使用してボットを作成し、ユーザとの対話を可能にすることができる。Slack 上で Amazon Lex を使用して、ボットを介した対話やタスクの実行ができ、ユーザが Slack 上でボットにメッセージを送信すると、Amazon Lex はそのメッセージを解析し、適切な応答を作成する。Amazon Lex は Facebook などの他のメッセージングサービスとの連携も可能であるが、使い慣れている Slack を採用した。

## 3.3 チャットボットの設計

チャットボットのシステム構成図を図 10 に示す。チャットボットは AWS Cloud 上に構成し、名前を AccessContorolSettingBot とした。リージョンは us-west(オレゴン)を選択した。TestBotAlias という名前のエイリアス上にバックエンドとなるプログラムを動かす lambda 関数がある。Lambda 関数は Python で記述しており、データベースである AccessContorolSettingBot\_DB のデータベース操作ができる。ユーザインターフェースは Slack を使用し、質問と選択肢を表示させる。デバイス管理者が設定した内容をもとに、

チャットボットは\_intentとslotを取得し、バックエンドのプログラムに取得したデータを渡す。プログラムがデータを処理して、intentの状態遷移をし、次のslotを取得するための質問と選択肢を作成する。チャットボットは質問と選択肢を表示させる。

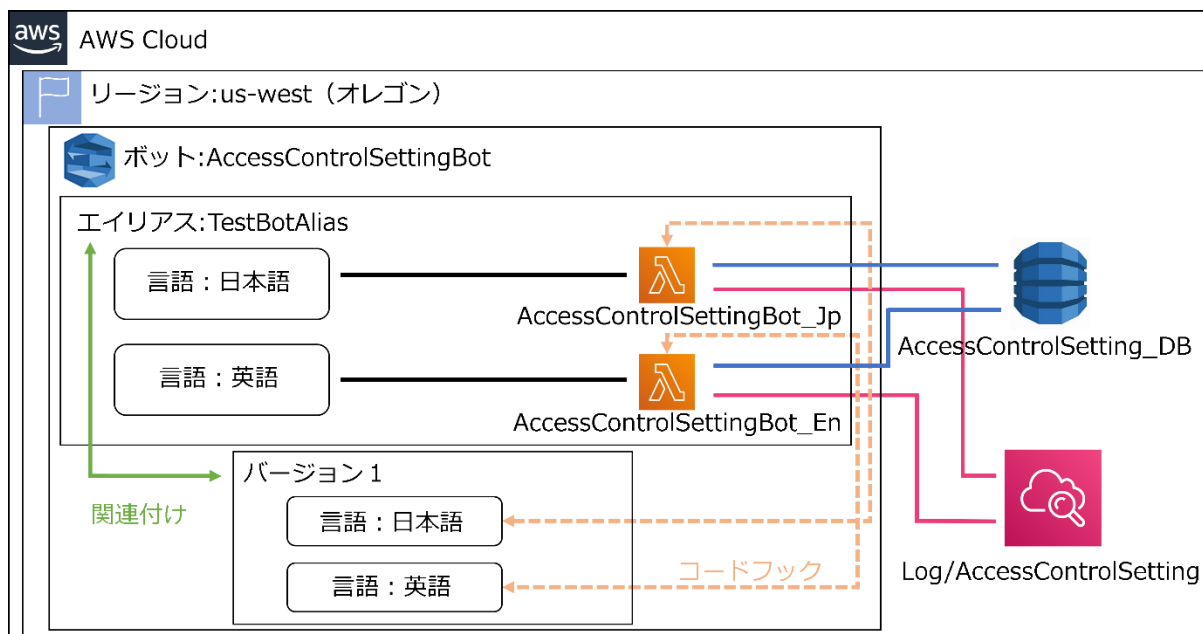


図 10 チャットボットのシステム構成図

### 3.3.1 チャットボットのフロー

チャットボットのフローを図 11 に示す。まず、デバイス管理者は管理者 ID とパスワードを使ってログインをする。デバイスに登録していてアクセス制御条件をカスタマイズできるアプリケーションが表示される。アプリケーションを選択すると、現在のアクセス制御条件が表示され、条件の削除と追加の選択肢が表示される。条件の追加を選択するとグループ名と属性名を選択でき、それぞれを入力すると条件名の選択肢が表示される。条件名を取得すると引数の選択肢が表示され、引数をすべて選択すると現在の条件で決定するか新しい条件を追加するかを選択できるようになる。これを繰り返すことで論理式が作成されていく。

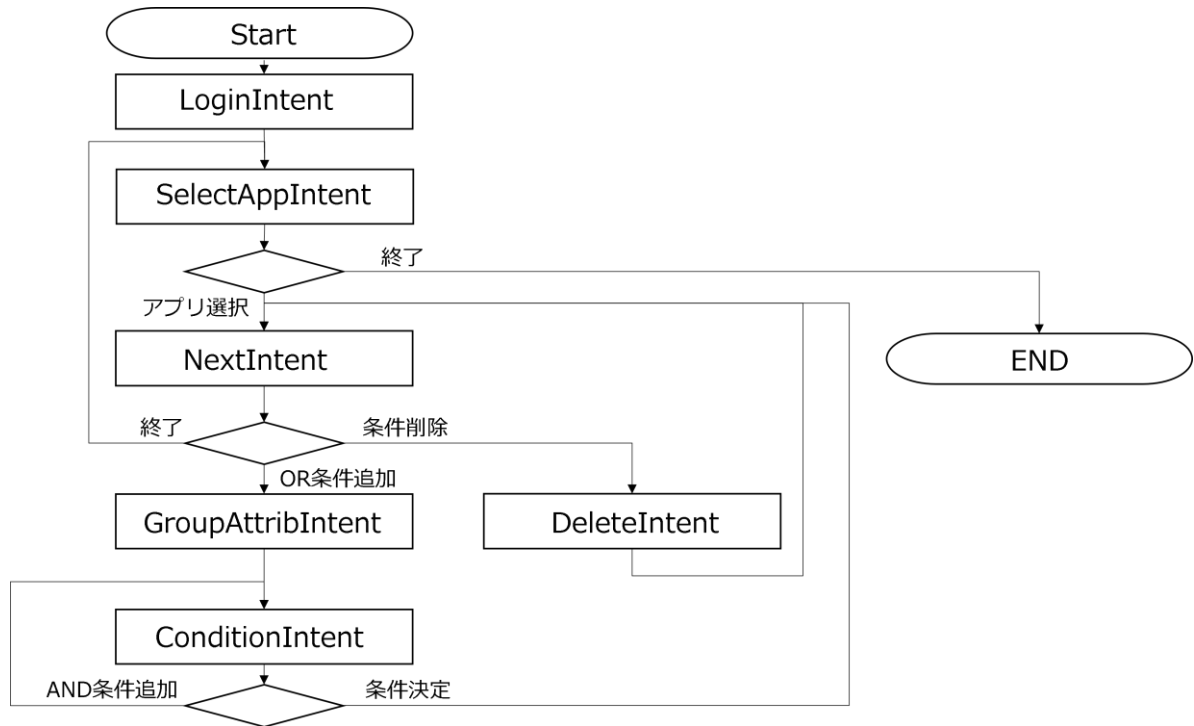


図 11 チャットボットのフロー

### 3.3.2 インテントとスロット

チャットボットのインテント一覧を表 2 に、スロット一覧を表 3 に示す。インテントとスロットは対応しており、それぞれのインテントで必要なスロットをユーザに質問して取得する。本システムに採用したチャットボットは AI 型ではなくシナリオ型のため、スロットがすべてそろったら次の状態に遷移する。

表2 チャットボットのインテント一覧

インテント名	対応スロット	アクション
LoginIntent	{ID} {Password}	スロット{Id}が無かったら管理者 ID を聞く スロット{Password}が無かったらパスワードを聞く 管理者 ID とパスワードが正しかったら、セッション属性に ID を追加する
SelectAppIntent	{AppName}	スロット{AppName}が無かったら、DB から管理者 ID を用いて AppName 一覧を取得する AppName 一覧と”終了”を選択肢にしてアプリケーション名を聞く スロット{AppName}があればセッション属性に AppName を追加する ※条件作成開始：計測を開始する 終了を選択していたら、セッションを終了する
NextIntent	{Next}	管理者 ID とアプリケーション名を用いて現在の条件（自然言語）を DB から取得する スロット{Next}が無かったら、現在の条件を示し、次の操作[“条件追加”, “条件削除”, “終了”]を選択肢にする スロット{Next}があれば次に進む ※スロット{Next}が”終了”であれば、条件作成終了：計測を終了する
GroupAttribIntent	{group} {attribute}	スロット{group}が無かったら、DB から管理者 ID を用いてグループ一覧を取得する グループ一覧と“指定しない”を選択肢にしてグループを聞く スロット{attribute}が無かったら、DB から管理者 ID を用いて属性一覧を取得する 属性一覧と“指定しない”を選択肢にして属性を聞く スロット{group}, {attribute}がそろっていたらセッション属性に group と attribute を追加する
ConditionIntent	{ConditionName} {Condition 1} {Condition 2} {Condition 3}	スロット{ConditionName}が無かったら、DB から AppName を用いてアプリケーション固有情報を取得する “決定”とアプリケーション固有の条件名とデフォルトの条件名を選択肢にして条件名を聞く “決定”が選択されていたら、セッション属性から、条件式を作成し、DB に保存し、次に進む スロット{ConditionName}があれば、DB から ConditionName を用いて条件情報を取得する 条件情報の引数に合わせて選択肢を作成し、{Condition N N=[1,

		2, 3]]を聞く 引数をすべて聞いたら, セッション属性に引数を追加し, スロット {ConditionName} を NULL にする 論理式と自然言語を作成・更新する
DeleteIntent	{Delete}	管理者 ID とアプリケーション名を用いて現在の条件 (自然言語) を DB から取得する スロット {Delete} が無かったら, 現在の条件を示し, 削除する条件を聞く スロット {Delete} があれば, DB からその行の条件を削除する 論理式と自然言語を削除・更新する

表3 チャットボットのスロット一覧

スロット名	データの型	データ種類と取得方法
{ID}	Int 型	管理者 ID. ユーザに ID を聞くことで取得.
{Password}	Int 型 (テスト版のため)	パスワード. ユーザにパスワードを聞くことで取得.
{AppName}	文字列	アクセス制御条件を設定するアプリケーションの名前. アプリ情報から選択肢を作成し, デバイス管理者が選択することで取得.
{Next}	文字列	次のステップ. 現在の条件 (自然言語) を表示し, ["条件追加", "条件削除", "終了"] を選択してもらうことで取得.
{group}	文字列	アクセス制御を設定する対象のグループ. ユーザ情報のグループ一覧から選択肢を作成し, デバイス管理者が選択することで取得.
{attribute}	文字列	アクセス制御を設定する対象の属性. ユーザ情報の属性一覧から選択肢を作成し, デバイス管理者が選択することで取得.
{ConditionName}	文字列	設定するアクセス制御の条件名. デフォルトの条件にアプリ情報からアプリケーション固有の条件を加え選択肢を作成し, デバイス管理者が選択することで取得.
{Condition N N= [1, 2, 3]}	文字列	設定するアクセス制御の引数. 関数情報から選択肢を作成し, デバイス管理者が選択することで取得.
{Delete}	Int 型	削除する条件の行数. 現在の条件 (自然言語) を表示し, 削除するアクセス制御条件を選択してもらうことで取得.

## 3.4 論理式作成について

### 3.4.1 論理式作成

論理式の作成は ConditionIntent の最後に行われる。図 12 に示すように、セッション属性にアクセス制御対象ユーザのグループと属性、そして関数名とその引数のすべてがスロットに入っている状態になったら、論理式が作成される。一つのセクションでは同時に満たすべき条件をすべて取得するので各関数を and でつなぐ。その時点で論理式が存在していたら or でつなぐ。

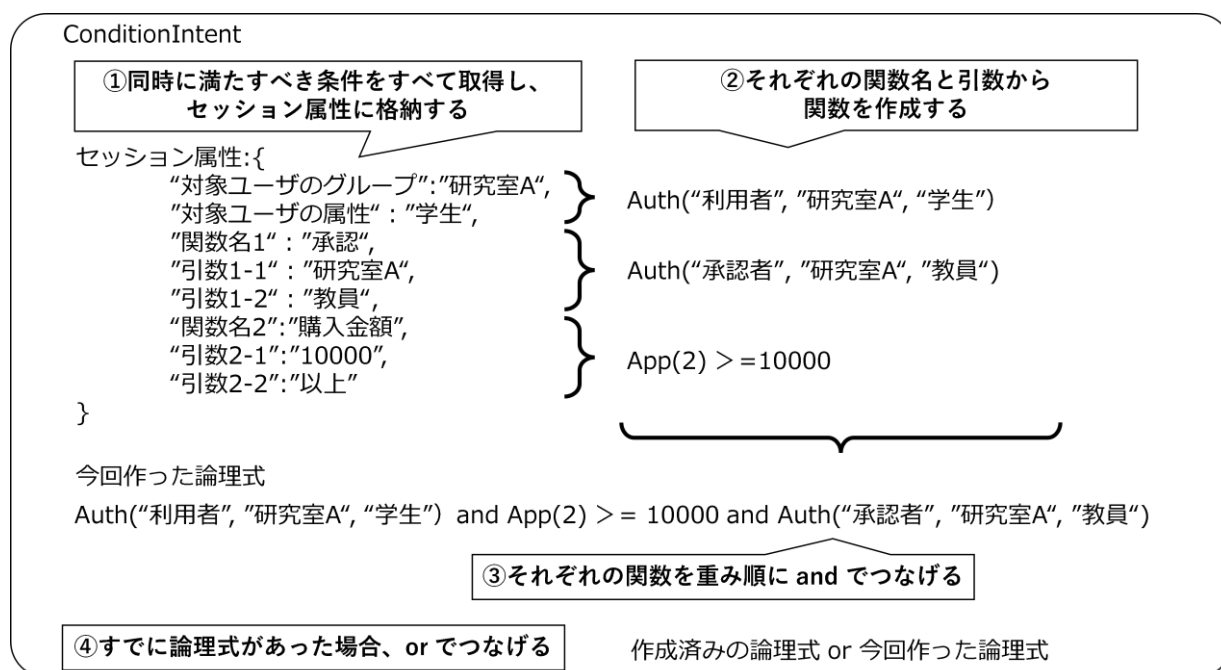


図 12 論理式作成の手順

### 3.4.2 自然言語作成

自然言語の作成は ConditionIntent の最後に行われる。図 13 に示すように、セッション属性にアクセス制御対象ユーザのグループと属性、そして関数名とその引数のすべてがスロットに入っている状態になったら、自然言語が作成される。一つのセクションでは同時に満たすべき条件をすべて取得するので各関数を横に並べる。その時点で自然言語が存在していたら改行して表示する。

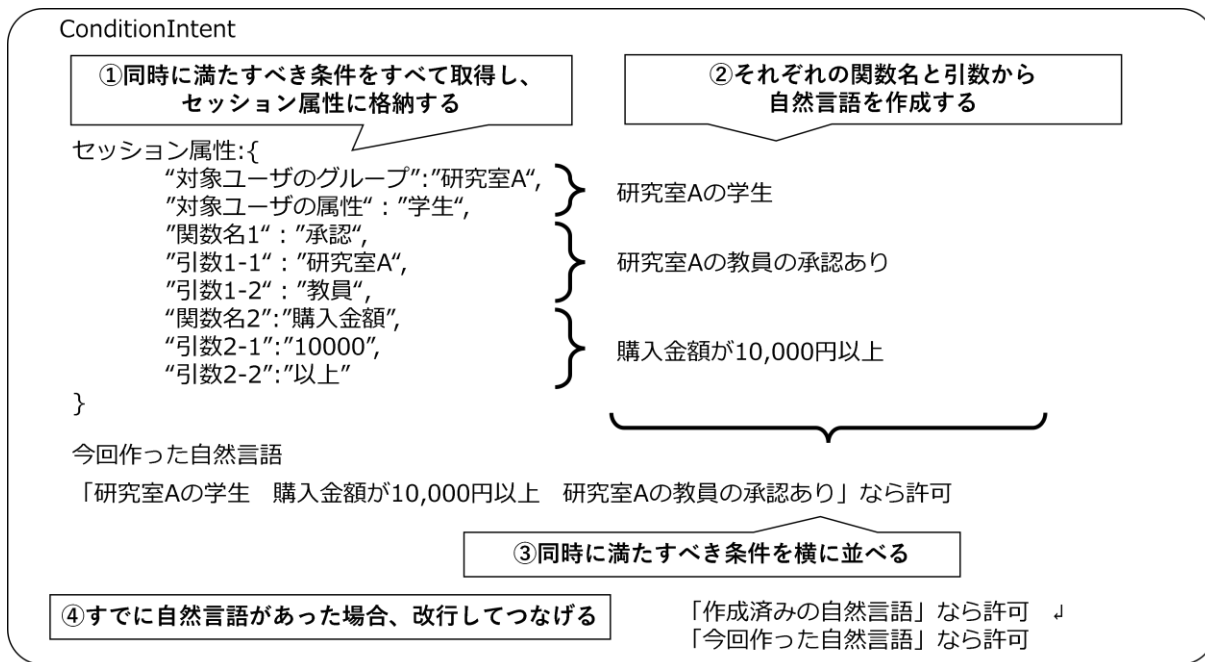


図 13 自然言語作成の手順

### 3.4.3 選択肢作成

\_intentで足りないスロットを取得するためにデータベースにある関数情報やアプリ情報を使って選択肢を作成する。アプリケーションの選択肢はデバイス管理者がデバイスにインストールしているアプリケーションの一覧から取得し作成する。Auth 関数の属性とグループの選択肢は事前にユーザ管理者によって利用者登録された属性一覧とグループ一覧から作成される。そのほかの関数の質問と選択肢はデータベースにある関数情報を取得して作成する。ただし、アプリケーション固有の条件は事前にアプリ開発者が事前に登録したアプリ情報から質問と選択肢が作成される。

### 3.4.4 データベース設計

データベース設計は次のようになっている。

アプリ情報のデータベース設計を図 14 に示す。アプリ情報はアプリ開発者が事前に登録しておく情報で、アプリケーション名や条件に使える関数とその引数を聞く質問と選択肢が書かれている。また、自然言語と論理式の作成規則が書かれている。作成規則は型のようなもので自然言語と論理式の形がどんなものになるのかが書かれている。

関数情報のデータベース設計を図 15 に示す。関数情報はデフォルトで使用できる関数が登録されており、引数を聞く質問や選択肢が書かれている。また、自然言語と論理式の作成規則が書かれている。

ユーザ情報のデータベース設計を図 16 に示す。ユーザ情報はユーザ管理者が登録するもので、利用者のグループや属性、メールアドレスが登録されている。メールアドレスは承認要求の時に使われるもので承認者にプッシュ通知が送られる。

デバイス情報のデータベース設計を図 17 に示す。デバイス情報は、デバイスに登録されているアプリケーションやユーザ登録されているグループや属性が保存されている。



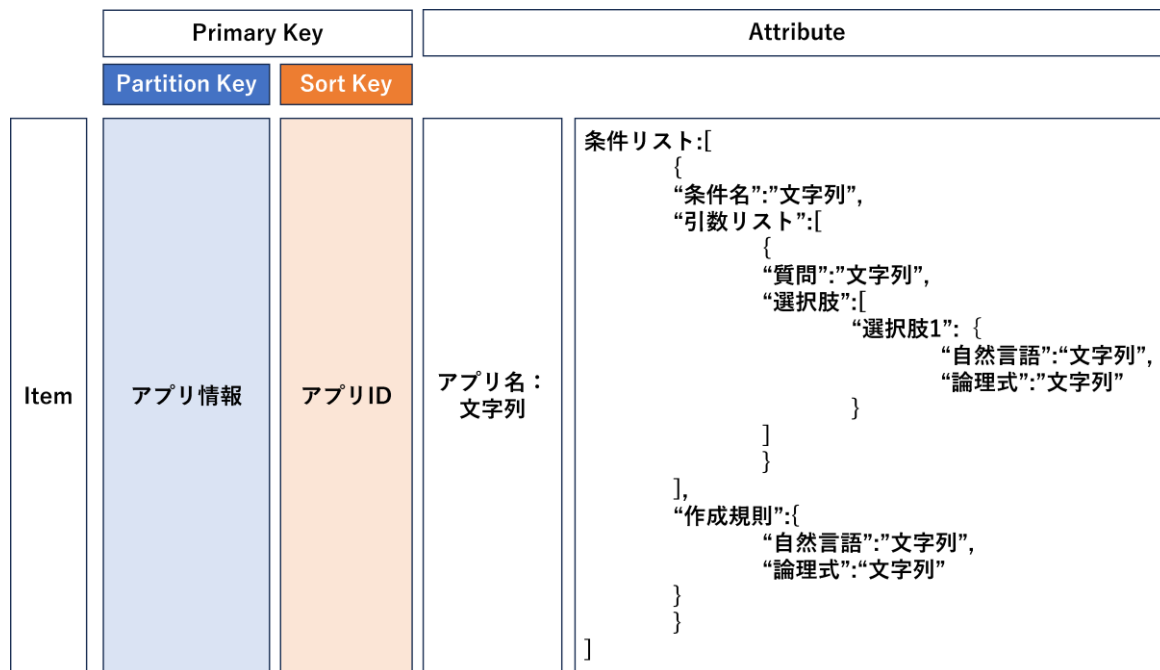


図 14 アプリ情報のデータベース設計

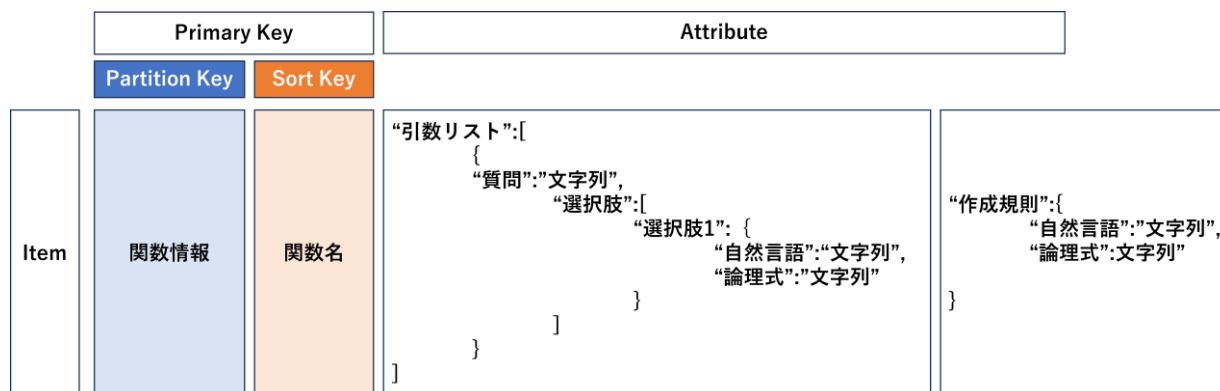


図 15 関数情報のデータベース設計

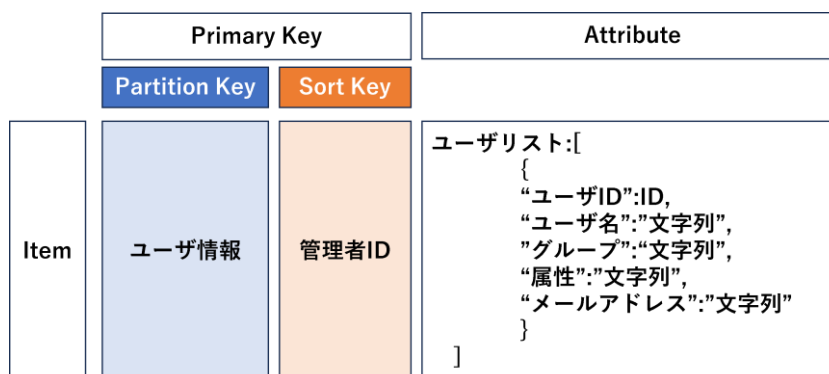


図 16 ユーザ情報のデータベース設計

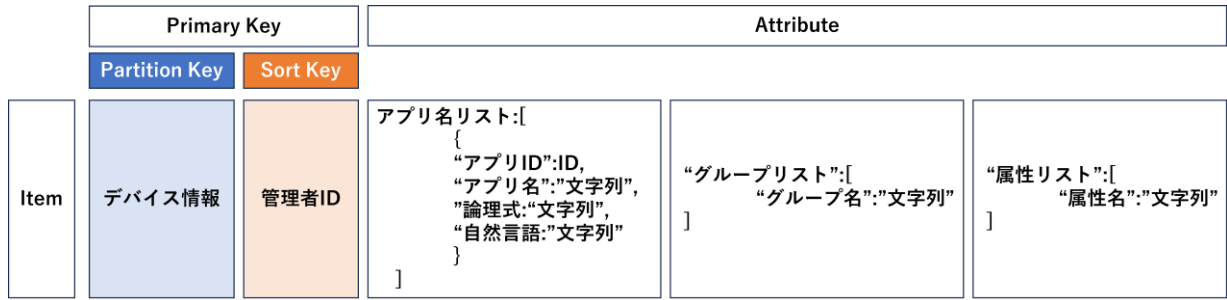


図 17 デバイス情報のデータベース設計

## 第4章 実装例

### 4.1 スマートスピーカの買い物アプリのアクセス制御

研究室にあるスマートスピーカの買い物アプリのアクセス制御の設定を、本システムを用いて行う場合の例を示す。利用者は研究室 A と B の教員と学生で、デバイス管理者及びユーザ管理者は研究室 A の教員の一人が担当する。特定の属性を持つ利用者が利用できる機能やアプリケーションに制限をかけるなどのアクセス権限をデバイス管理者が設定する。例えば、学生には買い物アプリの利用上限金額の設定や教員による承認等を設定できる。

アプリ開発者は買い物アプリのアプリ情報として図 18 に示す情報を登録する。図 15 の引数リストの質問は選択肢を聞き出すためのメッセージを格納する場所だが、スペースの関係上、一部省略している。デバイス管理者はスマートスピーカに買い物アプリを登録する。ユーザ管理者は研究室 A の教員と学生と研究室 B の教員と学生の利用者登録をする。

デバイス管理者は買い物アプリのアクセス制御条件として、「研究室 A の教員は無制限に購入可能で、研究室 A の学生は、10,000 円以内は購入可能だが 10,000 円を超えた場合は研究室 A の教員の承認が必要」という条件を設定する。この条件を最も簡略化した論理式で表すと、

Auth(“利用者”, “研究室 A”, “教員”) or Auth(“利用者”, “研究室 A”, “学生”) and

(App(2) <=10000 or Auth(“承認者”, “研究室 A”, “教員”))

となる。なお、App(2)では購入金額が戻り値になる。

Primary Key			Attribute
Partition Key	Sort Key		
Item	アプリ情報	ID001	アプリ名： 買い物アプリ
			<pre> 条件リスト:[   {     "条件名":"購入金額",     "引数リスト":[       {         "質問":"購入金額を入力してください(省略)",         "形式":"数値の入力",       },       {         "質問":"購入金額の条件を指定してください",         "選択肢":[           {             "自然言語":"[引数1]と等しい",             "論理式":"=="           },           {             "自然言語":"[引数1]より大きい",             "論理式": "&gt;"           },           {             "自然言語":"[引数1]以上",             "論理式": "&gt;="           },           {             "自然言語":"[引数1]より小さい",             "論理式": "&lt;"           },           {             "自然言語":"[引数1]以下",             "論理式": "&lt;="           }         ]       }     ]   },   {     "作成規則":{       "自然言語":"[引数2]",       "論理式": "App(2)[引数2][引数1]"     }   } ]                     </pre>

図 18 買い物アプリのアプリ情報

#### 4.1.1 チャットボットとの対話

上記のアクセス制御条件を設定する時の対話の一部を図 19 に示す。図 19 は、本来選択肢を選ぶと選択画面が消えるが、説明のため選択肢の画面を挿入している。利用者登録している属性一覧が { 教員, 学生 } でグループ一覧が { 研究室 A, 研究室 B } なので、それぞれ選択肢に表示されている。条件を選択する場面では、デフォルトの条件 { 承認, 曜日 } にアプリケーション固有の関数の { 購入金額 } の選択肢が追加されている。最終的に自然言語で表示されるアクセス制御条件は、

「研究室 A の教員 なら許可

研究室 A の学生 購入金額が 10,000 円以下 なら許可

研究室 A の学生 購入金額が 10,000 円より大きい 研究室 A の教員の承認あり なら許可」

となる。このアクセス制御条件をチャットボットで作成する場合、ステップ数は最短で 20 回になる。

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:33  
現在、すべての利用者にアクセス権限がありません。グループ、属性、条件の順で指定してアクセス権限を付与してください。

次のステップを選択してください

条件を追加する 条件設定を終了する

次のステップを選択してください  
条件を追加する

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:34  
グループを指定してください

研究室A 研究室B 指定しない

グループを指定してください  
研究室A

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:35  
属性を指定してください

学生 教員 指定しない

09:36 属性を指定してください  
学生

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:53  
作成中の条件は「研究室Aの学生」許可  
同時に満たす必要がある条件を足すか、現在の条件で決定するか選択してください

現在の条件を決定する 購入金額の条件を足す 承認の条件を足す  
曜日の条件を足す

作成中の条件は「研究室Aの学生」許可  
購入金額の条件を足す

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:54  
購入金額を入力してください 例) 1000  
購入金額を入力後に「等しい」「以上」「以下」「より小さい」「より大きい」の選択肢が表示されます

m223680 09:55  
10000

**AccessControlBot\_JAPANESE\_Ver2** アプリ 09:55  
購入金額の条件を指定してください

10000と等しい 10000以上 10000以下 10000より大きい 10000より小さい

購入金額の条件を指定してください  
10000以下

**AccessControlBot\_JAPANESE\_Ver2** アプリ 10:03  
作成中の条件は「研究室Aの学生 購入金額が10000以下」許可  
同時に満たす必要がある条件を足すか、現在の条件で決定するか選択してください

現在の条件を決定する 購入金額の条件を足す 承認の条件を足す  
曜日の条件を足す

作成中の条件は「研究室Aの学生 購入金額が10000以下」許可  
現在の条件を決定する

**AccessControlBot\_JAPANESE\_Ver2** アプリ 10:04  
現在の条件は以下のようにになっています。  
研究室Aの学生 購入金額が10000以下 なら許可

図 19 アクセス制御設定時の対話の一部

#### 4.1.2 論理式作成の流れ

論理式の作成の過程の一部を図 20 に示す。最終的に完成する論理式は、  
 (Auth(“利用者”, “研究室 A”, “教員”)) or (Auth(“利用者”, “研究室 A”, “学生”) and App(2) <=10000 )or  
 (Auth(“利用者”, “研究室 A”, “学生”) and Auth(“承認者”, “研究室 A”, “教員”))  
 となる。作成される論理式は、チャットボットのフローの関係上、利用者のグループと属性が重複した記述になる場合がある。

The screenshot shows a chatbot conversation with the following steps:

- Step 1:** The bot asks to select the next step. Buttons: "条件を追加する" (Add condition), "条件設定を終了する" (End condition setting).
- Step 2:** The bot asks to select a group. Buttons: "研究室A", "研究室B", "指定しない" (None). **① 最初は必ずグループと属性の関数** Auth(“利用者”, “グループ”, “属性”)
- Step 3:** The bot asks to select an attribute. Buttons: "学生", "教員", "指定しない". **② 関数に必要な引数の取得**
- Step 4:** The bot asks to select a condition. Buttons: "現在の条件を決定する", "購入金額の条件を足す", "承認の条件を足す", "曜日の条件を足す". **③ 関数の作成** Auth(“利用者”, “研究室A”, “教員”)
- Step 5:** The bot asks to enter a purchase amount. Buttons: "10000と等しい", "10000以上", "10000以下", "10000より大きい", "10000より小さい". **③ 追加する関数の選択** App(2) [引数2] [引数1]
- Step 6:** The bot asks to specify the purchase amount condition. Buttons: "10000以下", "10000以上", "10000以下", "10000より大きい", "10000より小さい". **④ 関数に必要な引数の取得**
- Step 7:** The bot asks to specify the purchase amount condition. Buttons: "10000以下", "10000以上", "10000以下", "10000より大きい", "10000より小さい". **⑤ 関数の作成** App(2) <= 10000
- Step 8:** The bot asks to decide on the current condition. Buttons: "現在の条件を決定する", "購入金額の条件を足す", "承認の条件を足す", "曜日の条件を足す". **⑥ ここまで作成した関数をandでつなぐ** Auth(“利用者”, “研究室A”, “教員”) and App(2) <=10000
- Step 9:** The bot displays the final condition: "研究室Aの学生 購入金額が10000以下 なら許可".

図 20 論理式の作成の過程の一部

## 第5章 評価

### 5.1 評価方法

#### 5.1.1 従来の自分で論理式を記述する方法との比較実験

情報科学部の2年生を対象に、従来のやり方である関数リストを参照して自身で論理式を作成する方法と提案手法であるチャットボットを用いた方法をそれぞれ体験してもらいアンケートをとった。従来の方法は、関数の引数や戻り値がまとめられた関数リストを参考にしながら自分で論理式を作成する方法である。まず両方のやり方でマニュアルとなる資料を用意し、資料を読みながらチュートリアルをしてもらい使い方に慣れてもらった。その後、事前アンケート、本実験、事後アンケートの順に進めた。本実験は両方のやり方でそれぞれ1問ずつ設定してもらったアクセス制御条件の問題を用意し、それを設定してもらった。また、チャットボットでは時間の計測とステップ数の計測を行った。従来の方法においても、Formsの回答時間をもとに時間の計測を行った。本実験は従来の自分で作成する方法から解くグループと提案手法であるチャットボットから解くグループの2グループに分けて行った。従来の方法から解くグループをグループA、提案手法から解くグループをグループBとする。本実験に用いた問題と解答例を表4に示す。本実験に使用した関数リストを表5に示す。問題1と問題2の時のアプリ関数(App)の内容は異なる内容だが、ここでは便宜上まとめて書いた。また、これ以降アプリAにおけるアプリ関数を $App_A$ 、アプリBにおけるアプリ関数を $App_B$ と表現する。事前アンケートは対象者の属性を聞くもので、所属などに加えて論理式に関する知識を問う問題を1問出題した。事後アンケートではチャットボットの使用感と自分で論理式を作成する方法と比較した際の難易度を聞いた。

表4 本実験に用いた問題と解答例

設問	解答例
研究室Aと研究室Bの教員と学生が利用登録をしています。アプリAは研究データが閲覧できるアプリです。研究室Aのデータ閲覧時のアクセス制御を次のように設定します 研究室Aの教員は、閲覧レベルが1と2の時、閲覧可能。 研究室Aの学生は、閲覧レベルが1の時は閲覧可能だが、閲覧レベルが2の時は研究室Aの教員の承認が必要である。	$Auth(\text{“利用者”, “研究室 A”, “教員”}) \text{ and } App_A(2) \leq 2 \text{ or } Auth(\text{“利用者”, “研究室 A”, “学生”}) \text{ and } (App_A(2) == 1 \text{ or } (App_A(2) == 2 \text{ and } Auth(\text{“承認者”, “研究室 A”, “教員”})))$
研究室Aと研究室Bの教員と学生が利用登録をしています。アプリBは商品が購入できるアプリです商品購入の前のアクセス制御を次のように設定します 研究室Bの教員は無制限に購入可能。 研究室Bの学生は購入金額上限が10,000円、10,000円を超えるときは研究室Bの教員の許可が必要である。	$Auth(\text{“利用者”, “研究室 A”, “教員”}) \text{ or } Auth(\text{“利用者”, “研究室 A”, “学生”}) \text{ and } (App_B(2) \leq 10000 \text{ or } Auth(\text{“承認者”, “研究室 A”, “教員”}))$

表 5 本実験に用いた関数リスト

関数の形	引数	説明
Auth(引数 1, 引数 2, 引数 3) 使用例) Auth(“利用者”, “研究室 A”, “ALL”) Auth(“承認者”, “研究室 A”, “教員”)	引数 1 “利用者”または “承認者” 引数 2 グループ名 引数 3 属性名	引数 1 が”利用者”の場合, 利用者のグループと属性が引数 2 と 3 で指定したグループと属性に一致していたら TRUE を返す 引数 1 が”承認者”の場合, 引数 2 と 3 で指定したグループと属性に一致している人物に承認要求を行い, 承認されたら TRUE を返す
Time(引数 1, 引数 2)	引数 1 下限時刻 引数 2 上限時刻	現在の時刻が引数 1 と引数 2 で指定した時刻の間なら TRUE を返す
App(引数 1) 使用例) App (2) == 2 App (2) > 1000	引数 1 添え字 (0,1,2...)	アプリ使用時のアプリの情報を返す. アプリ A では引数 1 が 2 の時, 閲覧レベル[1,2]を返す アプリ B では引数 1 が 2 の時, 購入金額を返す

### 5.1.2 評価項目

アンケートの内容を表 6 に示す。アンケート内では、従来の方法を、関数リストを用いた方法と表現している。論理式の真偽を問う問題は、事前アンケートとして、論理式の理解度を問うために出題した。真になる論理式が 3 つで偽になる論理式が 3 つある。真になる論理式をすべて選び、偽になる論理式をひとつも選ばないことを正解とした。被験者が与えられた論理式の中で真となるものを選択することで、論理的思考力やプログラムの論理式に対する理解度を測定した。事後アンケートではユーザエクスペリエンスの時間的側面やストレスに関する評価と従来の方法との難易度の比較、プログラミング未経験者の利用可能性の評価を行った。



表6 アンケート内容と出題意図

設問	選択肢	意図
以下は論理式のプログラム一例です。真偽が真 (True) になる式をすべて選択してください。(複数選択)	<input type="checkbox"/> $(1 < 0 \text{ or } 0 < 1) \text{ and } (1 == 0 \text{ or } 1 == 1)$ <input type="checkbox"/> $(0 < 1 \text{ or } 1 < 2) \text{ and } (1 == 0 \text{ or } 1 == 1)$ <input type="checkbox"/> $(1 < 0 \text{ or } 2 < 1) \text{ and } (1 == 0 \text{ or } 1 == 1)$ <input type="checkbox"/> $(1 < 0 \text{ and } 0 < 1) \text{ or } (1 == 0 \text{ and } 1 == 1)$ <input type="checkbox"/> $(0 < 1 \text{ and } 1 < 2) \text{ or } (1 == 0 \text{ and } 1 == 1)$ <input type="checkbox"/> $(1 < 0 \text{ and } 2 < 1) \text{ or } (1 == 0 \text{ and } 1 == 1)$ <input type="checkbox"/> 上記の論理式は全て偽になる	被験者が与えられた論理式の中で真となるものを選択することで、論理的思考力やプログラムの論理式に対する理解度を測定した。
チャットボットの入力から応答までの時間はどのように感じましたか	<input type="checkbox"/> とても長く感じた <input type="checkbox"/> 長く感じた <input type="checkbox"/> 短く感じた <input type="checkbox"/> とても短く感じた	チャットボットとの対話において、被験者が応答までの時間をどのように感じたかを評価した。
チャットボットの対話数はどう感じましたか	<input type="checkbox"/> とても多いと感じた <input type="checkbox"/> 多いと感じた <input type="checkbox"/> 少ないと感じた <input type="checkbox"/> とても少ないと感じた	対話の頻度が適切であるかどうかを評価した。
操作する際にストレスを感じましたか	<input type="checkbox"/> とても感じた <input type="checkbox"/> 感じた <input type="checkbox"/> あまり感じなかった <input type="checkbox"/> 全く感じなかった	ユーザがシステムを利用する際のストレスや負担を評価した。
ストレスを感じた・とても感じたと答えた方はどの部分にストレスを感じましたか(自由記述)	自由記述	具体的にどの部分がストレスを感じるかを聞くために自由記述を用意した。
関数リストとチャットボットを利用した場合で難易度の違いはどう感じましたか	<input type="checkbox"/> 関数リストの方がとても簡単 <input type="checkbox"/> 関数リストの方が簡単 <input type="checkbox"/> チャットボットの方が簡単 <input type="checkbox"/> チャットボットの方がとても簡単	関数リストとチャットボットを用いて同じ課題をする場合、被験者がどちらを利用する方が簡単だと感じるかを評価した。
もしプログラミングを学んでいない人がチャットボットを利用しても設定ができると思いますか	<input type="checkbox"/> 全くそうは思わない <input type="checkbox"/> そうは思わない <input type="checkbox"/> そう思う <input type="checkbox"/> とてもそう思う	プログラミング未経験者がチャットボットを使用して設定を行うことができるかどうかを評価した。
その他に感じたことがありましたら自由に記述してください	自由記述	その他の感想や意見の収集のために出題した。

### 5.1.3 測定項目

チャットボットを用いた方法は図 21 に示すように、アプリを選択してから条件設定の終了を選択するまでの測定区間で時間の計測を行った。従来の自分で論理式を作成する方法は回答用の Forms を別途用意し、回答にかかった時間（提出までの時間）を測定した。また、チャットボットにおいてはステップ数の測定を行った。ステップ数は論理式作成時に選択した回数と値を入力した回数の和によって算出する。

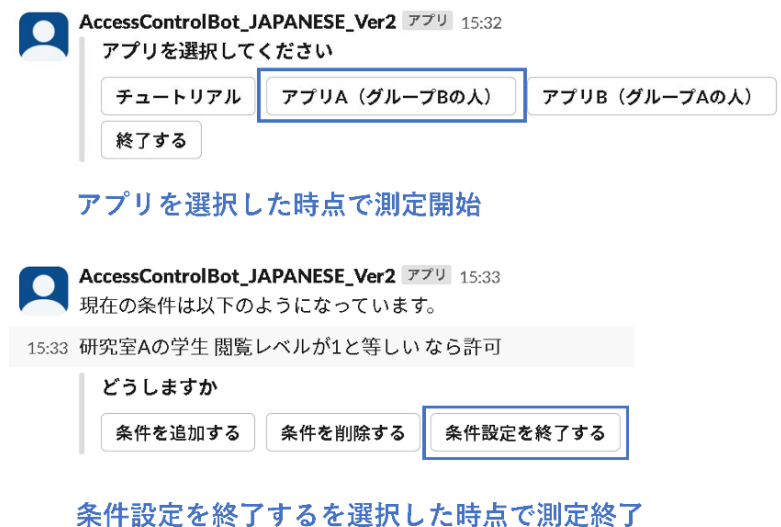


図 21 チャットボットの論理式作成の時間測定区間

## 5.2 結果

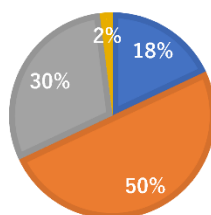
### 5.2.1 アンケート結果

対象者は情報科学部の 2 年生で、そのうちアンケートの回答者数は 50 名となった。グループ A で回答したのは 22 名、グループ B で回答したのは 28 名であった。事前アンケート内の論理式の理解度を測る問題では、完答した人が 25 名で、論理式を正しく理解している人の割合は 50%となった。

事後アンケートの結果を図 22 に示す。対話数が多いと感じる人が 70%、応答までの時間が長いと感じる人が 68%おり、チャットボットの操作に対してストレスを感じる人が 58%存在する。ストレスを感じた人に対してどの部分にストレスを感じたかについて質問したところ、「応答が遅いことがある（たまにある・よくある）」と答えた人が 12 人で最も多く、「修正が大変（細かく修正したい）」と答えた人が 5 人、「説明不足」と答えた人が 3 人、「途中でセッションが切れる（ログインに戻る）」と答えた人が 2 人、「エラーが出た」と答えた人が 1 人となった。また、難易度やプログラミング未経験者の利用可能性を問う質問では、従来の方法に比べチャットボットの方が簡単である・とても簡単であると答えた人の合計が 76%、プログラミング未経験者でもチャットボットを利用して設定できるという肯定的な意見が 78%だった。

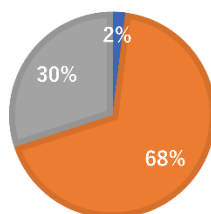
チャットボットの入力から応答までの  
時間はどのように感じましたか

■ とても長く感じた ■ 長く感じた ■ 短く感じた ■ とても短く感じた



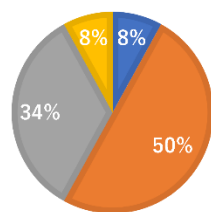
チャットボットの対話数はどう感じましたか

■ とても多いと感じた ■ 多いと感じた ■ 少ないと感じた ■ とても少ないと感じた



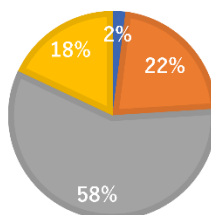
操作する際にストレスを感じましたか

■ とても感じた ■ 感じた ■ あまり感じなかった ■ 全く感じなかった



関数リストとチャットボットを利用した場合で  
難易度の違いはどう感じましたか

■ 関数リストの方がとても簡単 ■ 関数リストの方が簡単  
■ チャットボットの方が簡単 ■ チャットボットの方がとても簡単



もしプログラミングを学んでいない人がチャットボットを  
利用しても設定ができると思いますか

■ 全くそうは思わない ■ そうは思わない ■ そう思う ■ とてもそう思う

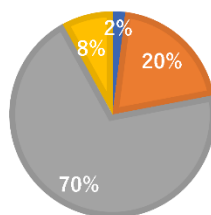


図 22 論理式作成に関する利用評価アンケート結果

### 5.2.2 測定結果

作成された論理式の正確率を表 7 と図 23 に示す。全体・グループ A・グループ B のいずれにおいてもチャットボットを用いた方法の正解率が従来の自分で論理式を作成する方法の正解率よりも高かった。従来の方法は全体の正解率が 24%と低い正解率であった。従来の方法のグループ A の正解率 (13.6%) とグループ B の正解率 (32.1%) を比較するとグループ B の方が高い正解率を示している。チャットボットを用いた方法は全体の正解率が 70%と高い正解率であった。チャットボットを用いた方法のグループ A の正解率 (81.8%) とグループ B の正解率 (60.7%) を比較するとグループ A とグループ B のいずれも高い正解率であるが、グループ A の方がやや高い傾向である。

論理式作成にかかった時間、ステップ数を表 8、表 9 と図 24 の上段に示す。全体・グループ A・グループ B のいずれにおいてもチャットボットを用いた方法の回答時間が自分で論理式を作成する方法の回答時間よりも短かった。最も回答時間として多かったのはチャットボットを用いた方法が 120 秒から 240 秒の区間で、自分で論理式を作成する方法が 240 秒から 360 秒の区間であった。不正解の回答には、回答時間が 1 分未満のものも含まれており、これが平均時間や平均ステップ数に影響を及ぼす可能性があるため、正解のみに焦点を当て、そのデータをまとめた。正解のみに関する論理式作成にかかった時間とステップ数を表 10、表 11 と図 24 の下段に示す。正解のみのデータを用いた場合においても、チャットボットを用いた方法の回答時間が自分で論理式を作成する方法の回答時間よりも短かった。自分で論理式を作成する方法は平均 433 秒であったのに対し、チャットボットを用いた方法では平均 277 秒と、2 分 30 秒近くの差があった。正解のみのデータのうち最も回答時間として多かったのは、チャットボットを用いた方法では 120 秒から 240 秒の区間で変わらず、自分で論理式を作成する方法は 480 秒から 600 秒の区間であった。

表 7 作成された論理式の正解率

回答方式	回答グループ	正解数 (割合)	不正解数 (割合)	未回答数 (割合)
自分で論理式を作成する方法	全体	12 (24.0%)	33 (66.0%)	5 (10.0%)
	グループ A	3 (13.6%)	19 (86.4%)	0 (0.0%)
	グループ B	9 (32.1%)	14 (50.0%)	5 (17.9%)
チャットボットを用いた方法	全体	35 (70.0%)	13 (26.0%)	2 (4.0%)
	グループ A	18 (81.8%)	3 (13.6%)	1 (4.5%)
	グループ B	17 (60.7%)	10 (35.7%)	1 (3.6%)

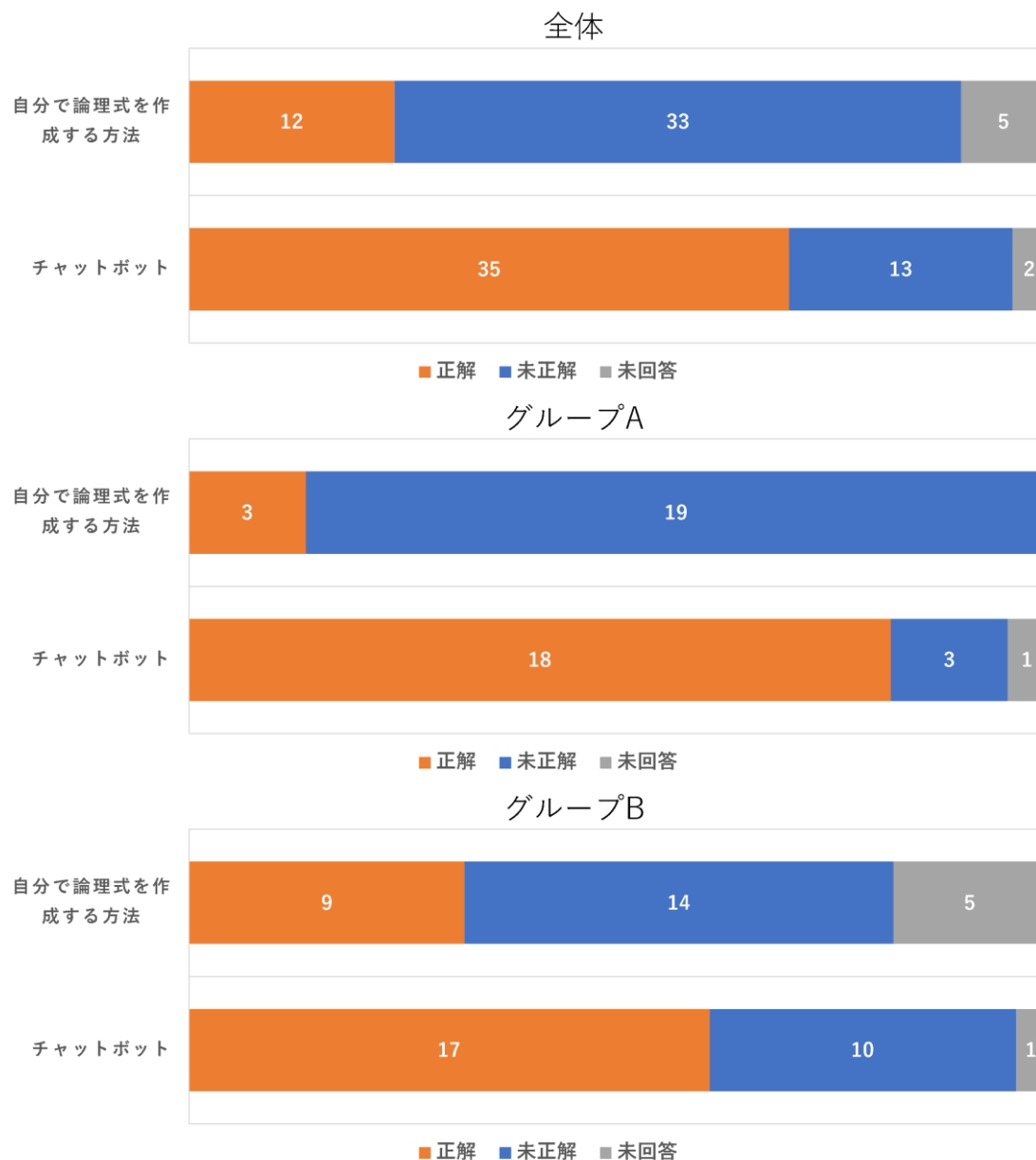


図 23 作成された論理式の正解者と不正解者の人数 (人)

表 8 論理式作成にかかった時間 (正解・誤答含む)

回答方式	回答グループ	平均時間	最大時間	最小時間
自分で論理式を作成する方法	全体	357 秒	843 秒	19 秒
	グループ A	417 秒	843 秒	19 秒
	グループ B	299 秒	554 秒	24 秒
チャットボットを用いた方法	全体	268 秒	631 秒	87 秒
	グループ A	215 秒	631 秒	87 秒
	グループ B	312 秒	500 秒	185 秒

表 9 論理式作成にかかったステップ数（正解・誤答含む）

回答方式	回答グループ	平均ステップ数	最大ステップ数	最小ステップ数
チャットボット	全体	29.5 回	70 回	5 回
	グループ A	23.6 回	42 回	5 回
	グループ B	34.1 回	70 回	13 回

表 10 論理式作成にかかった時間（正解のみ）

回答方式	回答グループ	平均時間	最大時間	最小時間
自分で論理式を作成する方法	全体	434 秒	843 秒	209 秒
	グループ A	644 秒	843 秒	534 秒
	グループ B	363 秒	554 秒	209 秒
チャットボットを用いた方法	全体	277 秒	631 秒	87 秒
	グループ A	226 秒	631 秒	87 秒
	グループ B	331 秒	490 秒	188 秒

表 11 論理式作成にかかったステップ数（正解のみ）

回答方式	回答グループ	平均ステップ数	最大ステップ数	最小ステップ数
チャットボット	全体	30.9 回	70 回	20 回
	グループ A	25.5 回	40 回	20 回
	グループ B	36.5 回	70 回	23 回

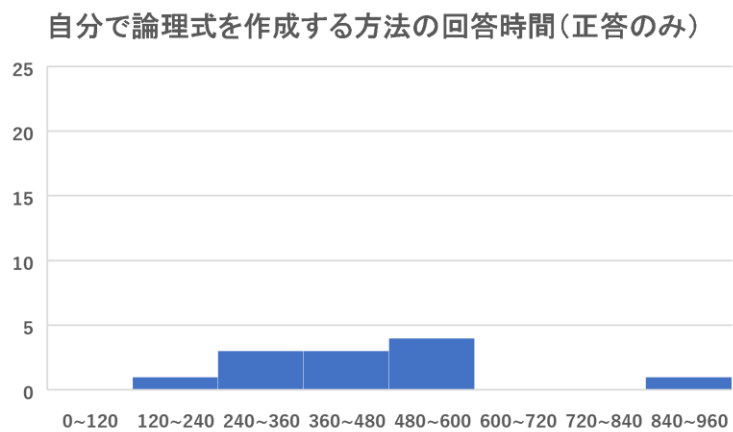
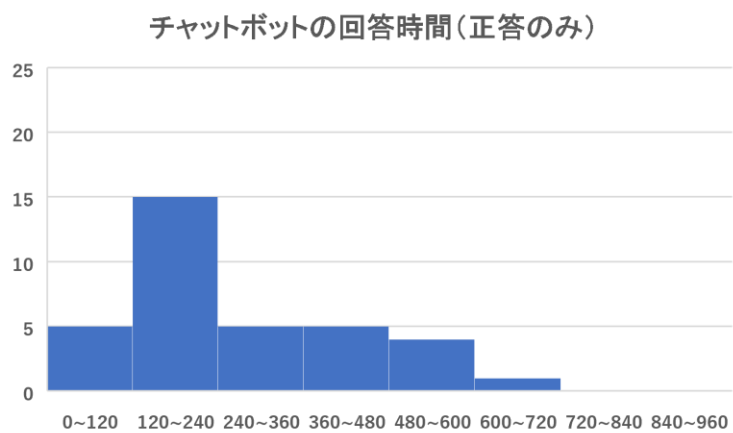
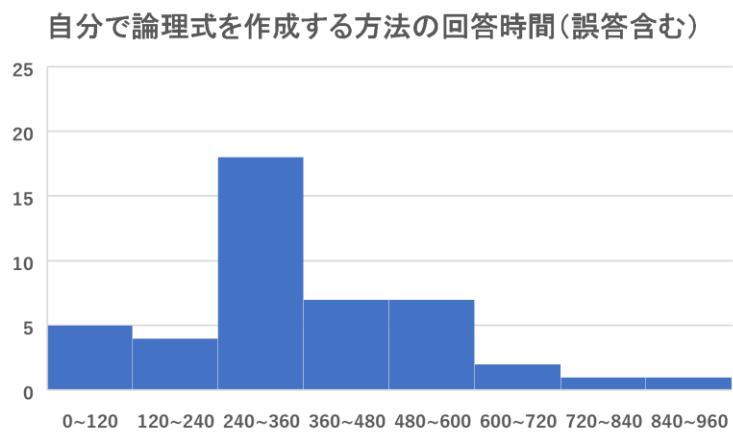
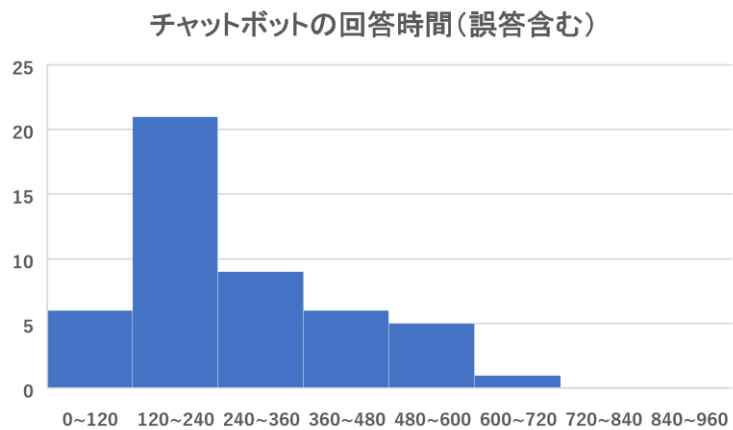


図 24 論理式作成にかかった時間分布 (正解・誤答含む)

## 5.3 考察

### 5.3.1 チャットボットのユーザエクスペリエンス

チャットボットのユーザエクスペリエンスについてはいくつか改善すべき点が見られる。

一つ目は、応答が遅いことである。応答に遅延があることに加え、セッションが途中で切れ、進行中の作業や対話が中断されることがストレスの要因となっている。研究室のメンバーを対象に少人数で行った実験時には応答の遅延やセッション切れは起こらなかったため、この応答の遅延やセッション切れはテスト時のアクセスの集中が原因であると考えられる。志和らの研究によれば、対話ロボットの応答時間は1秒から2秒の間で返すことが理想とされている[13]。そのため、アクセス集中時の処理の改善を検討していく。

二つ目は修正が大変という点である。戻る機能が実装されていないため、条件を作成時に細かい修正ができず、ストレスを感じる原因になっている。現段階ではブロック単位での条件の削除しかできないため、引数一つの変更等、細かい単位で修正を可能にできる仕組みを導入する必要がある。

三つめは説明不足という点である。チャットボットを使用してもらうにあたり、マニュアルとチュートリアルを用意したが、使い方が分からないと3人の回答者が感じていた。マニュアルの充実に加え、ヘルプインテントの追加等チャットボット操作時の支援をより強化する必要がある。

四つ目は対話数の多さである。重複した内容を入力することがあるため、アルゴリズムを見直し、少ない対話数に改善すべきである。本研究では1から論理式を作成することに焦点を当てたが、テンプレートから作成するなどの機能の追加も検討していく必要がある。

これらの要因から、ユーザエクスペリエンスを向上させるためには、特に応答時間の最適化、セッションの安定性向上、修正作業の容易化、説明の充実が重要な要素となる。

### 5.3.2 チャットボットの論理式作成の正解率

従来の自分で論理式を作成する方法では、全体の正解率が24%にとどまり、グループAでは13.6%、グループBでも32.1%という正解率であった。対照的に、チャットボットを使用した方法では、全体の正解率が70%にまで向上し、特にグループAでは81.8%、グループBでも60.7%の高い正解率が示された。グループAにおいては先に自分で論理式を作成する方法を実施したため、自分で論理式を作成する方法の正解率がより低く、チャットボットの正解率がより高くなったと考えられる。逆にグループBにおいてはチャットボットを先に実施したため、自分で論理式を作成する方法の正解率が比較的高く、チャットボットの正解率が比較的低くなったと考えられる。

チャットボットを利用した論理式作成方法は、論理式の正確性の向上に寄与したといえる。ただし、アクセス制御条件は正確なものを作ることが必須であるため、今回間違った論理式を作成した人には追加で何らかの支援が必要である。

### 5.3.3 チャットボットの論理式作成にかかる時間

アクセス制御条件の設定はスピードよりも正確性の方が重要であるが、教員のような時間的リソースが足りないデバイス管理者のことを考慮すると、より早く作成できることに越したことはない。自分で論理式を作成する方法では、正しい論理式を作成するのに全体平均で434秒かかっているが、これに対してチャットボットを用いた方法では277秒と、明らかに短い時間で条件を作成できている。また、最大



時間や最小時間、そして回答時間の分布を見たときにばらつきが少ないことから安定した時間で論理式作成ができると考えられる。

### 5.3.4 チャットボットの難易度

自分で論理式を作成する方法と比較して、「チャットボットの方が簡単」と回答した参加者は合計 38 人であり、対照的に自分で論理式を作成する方法が簡単と感じた参加者は 12 人と、回答者の多くがチャットボットの方が簡単であると感じている。自分で論理式を作成する方法はアクセス制御設定のために関数に対する知識の理解や論理的思考力が必要なのに対し、チャットボットは対話形式で質問に答えていくだけのため、直感的でわかりやすいことがこの結果になった要因と考えられる。

また、プログラミング未経験者がチャットボットを利用しても設定ができるかという問いに対し、回答者のうち「そう思う」または「とてもそう思う」と回答した人数は合計 39 人であり、チャットボットを介することで技術的なハードルが低くなったと考えられる。

総じて、チャットボットを利用したアクセス制御設定の難易度が低く、プログラミング未経験者にとっても利用しやすいと考えられる。

### 5.3.5 よくある誤答と改善方法

誤答の仕方を 4 つの場合に分けた。一つ目は、かっこの数が合っていない、引数の数が合っていない、演算子が足りないものを「論理式として成立していない」とした。二つ目は、論理式として成立しているが、引数に不適切な変数が入力されている、戻り値に誤解があるものを「関数の知識・理解不足による間違い」とした。三つ目は、論理式は成立しており関数の使い方は合っているが、論理演算子の影響範囲が間違っている、適切な論理演算子が選択されていないものを「論理演算子の知識・理解不足による間違い」とした。最後に、上記の 3 つには当てはまらないが、正しいアクセス制御条件が設定できていないものを「問題の理解不足による間違い」とした。

従来の自分で論理式を作成する方法における間違いは、「論理式として成立していない」が 19 人で最も多く、「関数の知識・理解不足による間違い」が 7 人、「論理演算子の知識・理解不足による間違い」が 2 人、「問題の理解不足による間違い」が 5 人であった。「論理式として成立していない」は、エディタを用いることで間違いを指摘できるが、デバイス管理者がエディタに慣れている必要がある。19 人の「論理式として成立していない」の回答をさらに細分化すると演算子が抜けている人が 9 人で最も多かった。式と式の間には必ず演算子が必要になるが、演算子が抜けていることを見落としていることが原因と考えられる。

チャットボットを用いた方法では、自動で論理式が作成される性質上「論理式として成立していない」、「関数の知識・理解不足による間違い」、「論理演算子の知識・理解不足による間違い」ということは起きない。しかし、同時に満たすべき条件を追加する箇所でも同時に満たさない条件を追加するなどの間違いが 4 人の回答であった。例えば、「閲覧レベルが 1 または閲覧レベルが 2」もしくは「閲覧レベルが 1 以上かつ閲覧レベルが 2 以下」とするべきところを「閲覧レベルが 1 かつ閲覧レベルが 2」とする間違いが見受けられた。より正確なアクセス制御条件の設定を目指すには、このようなありえない条件を指摘する仕組み、または、間違いから本来設定すべき条件を予想して提案する仕組みが必要である。

### 5.3.6 チャットボットを用いる方法の正解パターンとそこに至るまでの流れ

チャットボットを用いた方法では、回答者がどのようなルートで答えにたどり着いたかが選択のログからわかる。ここでは2つの正解パターンとそのルートについて述べる。

一つ目は少ないステップ数でほぼ最短ルートで正解にたどり着いたパターンである。このパターンで作成された論理式の一つは次に示す。この論理式を作成するには作成をはじめの前に頭の中で作成すべきアクセス制御条件が理解できている必要がある。主に20~25ステップで作成している回答者がこのパターンに当てはまると言えて、全体で15人いた。

二つ目は何度も繰り返し修正しながら、正解にたどり着いたパターンである。こちらは適宜修正しながら作成していったパターンである。ステップ数が35を超えた11人の回答者はこのパターンといえる。こちらのパターンにおいて最終的に完成された論理式は様々な形になると想定していたが、実際は似た形の論理式が作成されていた。

当初は最終的な論理式の形から場合分けができると考えていたが、実際に出来上がる論理式はステップ数に関係ないことが分かった。

### 5.3.7 提案手法で改善できるターゲット

従来のやり方では間違えていたが、提案手法のチャットボットを用いた方法では正解していた回答者は22人おり、その回答者がどのような回答をしていたかを述べる。

22人のうち、事前アンケートの論理式問題の正解者は9人で不正解者は13人であった。回答者全体の正解率が50%であったため、この22人は論理式の知識がない人が比較的多いと考えられる。

従来の自分で論理式を作成する方法における間違いは、「論理式として成立していない」が11人で最も多く、「関数の知識・理解不足による間違い」が6人、「論理演算子の知識・理解不足による間違い」が2人、「問題の理解不足による間違い」が4人であった。「論理演算子の知識・理解不足による間違い」をしていた回答者は全体でも2人であったため、サンプル数が少ないが提案手法により全員が改善された。「問題の理解不足による間違い」をしていた7人のうち6人が改善されたため、提案手法の効果がある間違いといえる。また、「問題の理解不足による間違い」をしていた5人のうち4人が改善されたため、この間違いも提案手法の効果がある間違いといえる。一方で、「論理式として成立していない」間違いをしていた人は19人のうち11人の改善と、提案手法の効果は中程度であったといえる。

この19人のチャットボットのステップ数の平均は29.9回と、全体の平均29.5回や正解者のみの平均の30.5回と比較しても差は見られなかった。

### 5.3.8 生成AIによる論理式の最適化

チャットボットを用いた方法では、冗長な論理式が生成される。正解だった回答を生成AIによって最適な論理式に変換できるのかを試した。生成AIはChatGPT[14]のバージョン3.5を使用した。命令文・変換前の論理式と変換後の論理式、そして最適な論理式を以下に示す。

指示1：次に送る論理式を冗長な条件や重複した部分を排除して最適な形に変換してください。

変換前論理式： $(\text{Auth}(\text{"利用者"}, \text{"研究室 B"}, \text{"教員"})) \text{ or } (\text{Auth}(\text{"利用者"}, \text{"研究室 B"}, \text{"学生"})) \text{ and } \text{App}_B(2) < 10000) \text{ or } (\text{Auth}(\text{"利用者"}, \text{"研究室 B"}, \text{"学生"})) \text{ and } \text{App}_B(2) > 10000 \text{ and } \text{Auth}(\text{"承認者"}, \text{"研究室 B"}, \text{"教員"}))$

変換後論理式： $(\text{Auth}(\text{"利用者"}, \text{"研究室 B"}, \text{"教員"})) \text{ or } ((\text{Auth}(\text{"利用者"}, \text{"研究室 B"}, \text{"学生"})) \text{ and } \text{App}_B(2))$

< 10000) or (Auth("利用者", "研究室 B", "学生") and  $App_B(2) > 10000$  and Auth("承認者", "研究室 B", "教員"))

指示 1 の内容だけでは論理式の最適化ができなかったので以下の指示 2 を追加した結果が以下のようになる。

指示 2 : 冗長な条件や重複した部分の削除は、以下のルールをもとに行ってください。

ルール 1 変換前 (A and B) or (A and C) 変換後 (A and (B or C))

ルール 2 変換前 (A or B) or (A or C) 変換後 (A or B or C)

ルール 3 変換前 (A or B) and (A or C) 変換後 (A or B or C)

変換前論理式 : (Auth("利用者", "研究室 B", "教員")) or (Auth("利用者", "研究室 B", "学生") and  $App_B(2) < 10000$ ) or (Auth("利用者", "研究室 B", "学生") and  $App_B(2) > 10000$  and Auth("承認者", "研究室 B", "教員"))

変換後論理式 : Auth("利用者", "研究室 B", "教員") or (Auth("利用者", "研究室 B", "学生") and ( $App_B(2) < 10000$  or ( $App_B(2) > 10000$  and Auth("承認者", "研究室 B", "教員"))))

指示 2 を追加することで最適な論理式に変換することができた。ほかの 3 つの論理式を変換させた場合も同様に最適化できた。完成した論理式を登録する際にこの仕組みを取り入れることで論理式の最適化が可能になる。

## 第6章 まとめ

### 6.1 まとめ

本研究では、デバイス管理者が柔軟かつセキュアにスマートデバイスを制御するための論理式作成を支援するシステムの開発を行った。従来の方法では、関数リストを参照しながら論理式を手動で作成する必要があった。しかし、チャットボットを利用したアクセス制御の設定は、アンケートの回答者の76%が従来のやり方でアクセス制御を設定するよりも簡単と感じており、本システムのチャットボットの導入によりプログラミングに不慣れな管理者にも直感的で分かりやすいものになった。さらに論理式作成の正解率においても、チャットボットを使用すると全体の正解率が24%から70%と大幅に向上しており、正確な論理式を作成できるようになった。さらに時間的な制約がある場合も考慮しており、論理式作成にかかる時間においても平均434秒から平均277秒と、時間の短縮が可能になった。これにより、デバイス管理者のITスキル不足を克服し、多人数環境における共有型のスマートデバイスの安全な共有利用を促進する。

### 6.2 今後の課題

本研究では扱えなかったが、チャットボットのユーザエクスペリエンスには改善すべき点もみられる。チャットボットの応答時間の工夫することでユーザのストレスが少なくなると考えられる。論理式の間違いの指摘や論理式の最適化にはAIを導入することで、より正確なアクセス制御設定を実現できる。

今回の評価では事前アンケートのみで行った論理式の真偽問題を本システムの利用前後で実施することで、本システムによる学習効果について調査が可能であると考えられる。真偽問題では作成された論理式を評価する能力を測ることができ、この能力は自然言語で表現されたアクセス制御条件の理解をする上でも必要になる。ほかにも自分で論理式を作成する能力を測るテストを本システムの前後で実施する等、論理式作成能力の変化についての評価を検討している。

音声によるユーザのアクセス制御を多人数環境におけるスマートデバイスに組み込みたかったが、現在の音声認識の精度は少人数の識別は可能である一方、多人数の認証に使えないという制約がありできなかった。今後、音声認証の精度が向上したら、ユーザフレンドリーな認証が可能になり、デバイス管理者がアクセス制御条件を設定できる仕組みと組み合わせることで共有型のスマートデバイスの利用がより一層拡大するだろう。

## 謝辞

指導教員の広島大学情報メディア教育研究センター西村浩二教授には、多大なご指導・ご協力を賜りました。心より感謝申し上げます。

広島大学情報メディア教育研究センター渡邊英伸准教授、村上祐子助教におかれましては、個別ミーティングや論文執筆の際に多くの助言を頂戴いたしました。心より感謝申し上げます。

広島大学情報メディア教育研究センター近堂徹教授、岸場清悟助教、田島浩一助教、下地寛武特任助教、相原玲二上席特任学術研究員におかれましては、研究ミーティングにおいて多くの助言を頂戴いたしました。心より感謝申し上げます。

副指導教員の広島大学平嶋宗教授には、個別面談や中間発表時にご助言していただきました。心より感謝申し上げます。

轟木皓平さん、中野敦斗さん、森田崇大さんをはじめ研究室の皆さんには、研究活動を通して良い刺激を頂き、評価実験にも協力していただきました。心より感謝申し上げます。

評価実験にあたり、情報科学部の学部生の皆さんにもご協力をいただきました。心より感謝申し上げます。

最後に、様々な面で支えて頂いた家族に心より感謝申し上げます。

## 参考文献

- [1] 高橋朋也, 渡邊英伸, 西村浩二, “音声アシスタントの認証認可機能を高度化する外部連携システムの開発”, インターネットと運用技術 (IOT) Vol.2022-IOT-58, No.6, 1-7, 2022.
- [2] Tomoya Takahashi, Yuko Murakami, Hidenobu Watanabe and Kouji Nishimura. Logical Expression Tutoring System for Controlling Smart Devices in Multi-User Environments. Proceedings of the 31st International Conference on Computers in Education. 926-928, 2023
- [3] Karen E.I. The Impact of Educational Technology on Student Achievement: Assessment of and for Learning. Science Educator, 15(1), 13-20, 2006
- [4] 御園真史, 竹崎修次, 西村隆正, 真玉保浩, “論理的思考力評価のための調査問題の開発”, 日本科学教育学会研究会研究報告, Vol.28, No.7, 49-52, 2018.
- [5] 総務省, 令和4年通信利用動向調査の結果,  
[https://www.soumu.go.jp/johotsusintokei/statistics/data/230529\\_1.pdf](https://www.soumu.go.jp/johotsusintokei/statistics/data/230529_1.pdf) (参照 2024-01-25)
- [6] 仕事にも家族と共有でも使える, Android のマルチユーザでスマートなスマホ利用を,  
[https://www.android.com/intl/ja\\_jp/articles/54/](https://www.android.com/intl/ja_jp/articles/54/)(参照 2024-01-25)
- [7] Amazon カスタマーサービス, Alexa プロフィールとは何ですか?, <https://arcus-www.amazon.co.jp/gp/help/customer/display.html?nodeId=GLE4EVA2VLDRQDLC>(参照 2024-01-25)
- [8] Nicole M, Dilara K, & Kami V. Owning and Sharing: Privacy Perceptions of Smart Speaker Users. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1), 1-29, 2021
- [9] 堀 和彦, 永井 孝幸, “XACML を用いた中継サーバの開発と共同指導における生徒情報閲覧制限の実現”, 研究報告教育学習支援情報システム (CLE) , Vol.2018-CLE-25, No.7, 1-8, 2018
- [10] 堀 和彦, 永井 孝幸, “学校間連携を考慮した分散型生徒情報リポジトリに対するアクセス制御システムの試作”, 研究報告教育学習支援情報システム (CLE) , Vol.2020-CLE-30, No.4, 1-8, 2020
- [11]AWS, Amazon Lex (Alexa と同じテクノロジーを利用した会話型インターフェース) ,  
<https://aws.amazon.com/jp/lex/>(参照 2024-01-25)
- [12]AWS, Amazon DynamoDB とは (マネージド NoSQL データベース) ,  
<https://aws.amazon.com/jp/dynamodb/>(参照 2024-01-25)
- [13] 志和 敏之, 神田 崇行, 今井 倫太, 石黒 浩, 萩田 紀博, 安西 祐一郎, 対話ロボットの反応時間と反応遅延時における問投詞の効果, 日本ロボット学会誌, Vol.27, No.1, 87-95, 2009.
- [14] ChatGPT (openai.com), <https://openai.com/chatgpt>(参照 2024-01-25)

## 学会発表実績

- 1) 高橋朋也, 渡邊英伸, 西村浩二, “音声アシスタントの認証認可機能を高度化する外部連携システムの開発”, インターネットと運用技術 (IOT) Vol.2022-IOT-58, No.6, 1-7, 2022.
- 2) Tomoya Takahashi, Yuko Murakami, Hidenobu Watanabe and Kouji Nishimura. Logical Expression Tutoring System for Controlling Smart Devices in Multi-User Environments. Proceedings of the 31st International Conference on Computers in Education. 926-928, 2023

## 付録

## A1. 論理式作成チャットボット 実装手順

0. AWS アカウントと Slack API のアカウントを作成 スラックのワークスペースも作成
1. AWS コンソールで dynamo DB を選択
  - 1.1 「テーブルを作成」を選択
  - 1.2 任意のテーブル名・パーティションキー名・ソートキー名を入力
  - 1.3 テーブル設定はすべてデフォルトのものを使用し「テーブルの作成」を選択
2. AWS コンソールで amazon lex を選択
  - 2.1 「ボットを作成」を選択
  - 2.2 「空のボットを作成」を選択
  - 2.3 任意のボット名・説明を入力
  - 2.4 IAM アクセス許可はデフォルトのものを使用
  - 2.5 COPPA はいいえを選択
  - 2.6 セッションタイムは 10 分に設定
  - 2.7 言語は日本語を設定し「完了」を選択
  - 2.8 図 3 のスロット名一覧をもとにすべてのスロットを作成
  - 2.9 図 2 のインテント名一覧をもとにすべてのインテントを作成
  - 2.10 各インテントのコードフックオプションを on に設定
  - 2.11 図 11 のチャットボットのフロー図と図 2 のインテント名一覧をもとに各インテントの状態遷移を作成する。図 25 に LoginIntent の状態遷移を示す。Start→code hook→Get slot value→Go to Intent の順につなげる。Go to Intent の行先は図 11 のフロー図をもとに、Get slot value については図 2・図 3 のインテントとスロットの説明をもとに作成する。各 Get slot value で取得したデータはセッションアトリビュート変数に格納しておく。

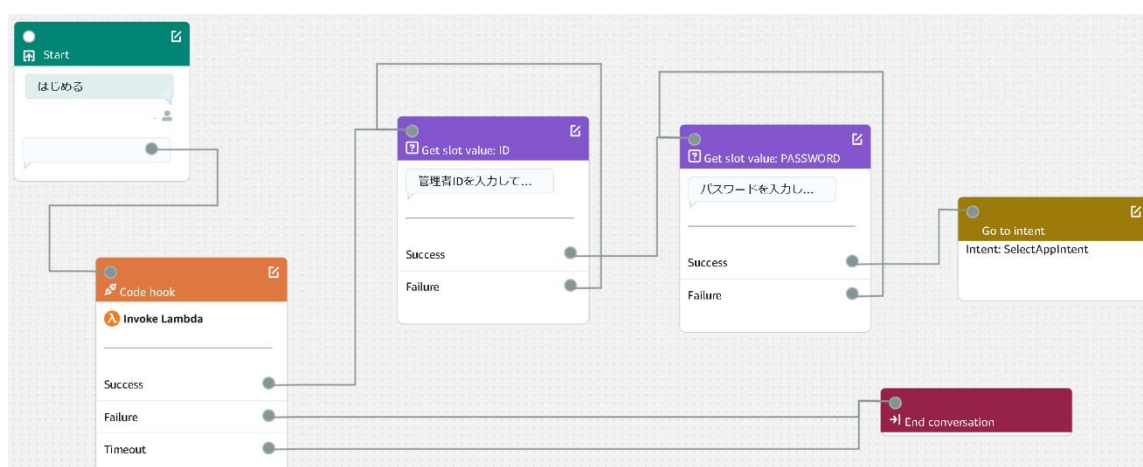


図 25 LoginIntent の状態遷移図

- 2.12 全てのスロット・インテントを作成したら「Build」を選択
  - 2.13 メニューの「ボットのバージョン」から「バージョンの作成」を選択，後の項目はすべてデフォルトのものを使用する。
  - 2.14 メニューのデプロイから「エイリアス」を選択しエイリアスを作成する。2.13 で作成したバージョンを関連付ける。「Lambda 関数 - オプション」は3で作成するものを選択する
  - 2.15 メニューのデプロイから「チャンネル統合」を選択し Slack と連携する。4 で詳細に説明する。
3. AWS コンソールで Lambda を選択
    - 3.1 関数の作成を選択
    - 3.2 任意の関数名を入力し，ランタイムの中から最新の Python を選択
    - 3.3 命令セットアーキテクチャは x86\_64 を選択
    - 3.4 アクセス制御にデータベース操作を追加
    - 3.5 A2 の付録のソースコードを使用
    - 3.6 データベース名を 1 で作成したデータベース名に変更
  4. Slack API で Create App を選択
    - 4.1 「Create New App」を選択
    - 4.2 From Scratch を選択し任意のアプリ名を入力
    - 4.3 Slack のワークスペースを選択（ワークスペースがない場合は作成）
    - 4.4 「Building Apps for Slack」で作成
    - 4.5 クライアント ID/クライアントシークレット/検証トークンを取得
    - 4.6 Amazon lex のチャンネル統合のプラットフォームで Slack を選び，4.5 で取得したクライアント ID/クライアントシークレット/検証トークンを入力する。
    - 4.7 コールバック URL の 2 つを取得
    - 4.8 スラック API のリクエスト URL ・リジェクト URL に 4.7 で取得した 2 つの URL を入力
    - 4.9 ボットトークンスコープに「chat:write」と「team:read」を追加
    - 4.10 「Subscribe to bot events」に「message.im」を追加
    - 4.11 Manage Distribution で Add to Slack を選択

#### A2. 論理式支援システムソースコード（図 10 の AccessControlSettingBot\_Jp (Python3.12)）

```
import json
import boto3
from boto3.dynamodb.conditions import Key
import time

#XXX にデータベース名を記入
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table("XXX")

#DB からパーテーションキーとソートキーを指定してデータを取得
def DB_get(partitionKey,sortKey):
    getitem = table.get_item(
        Key={
            'P_key': partitionKey,
            'S_key': sortKey
```



```
    )
    return getitem

#DB にデータを置く
def DB_put(item):
    table.put_item(Item=item)
    return True

#タイムスタンプをつける(検証用)
def TimeStamp_put(data,status):

    nowtime = int(time.time())
    data["TimeStamp"].append({"Time":nowtime,"Status":status})
    return data

#Slot を話すように促す(選択肢を動的に変化させたい場合)
def elicit_slot_card(slot_to_elicit, messagelist, intent_name, slots):

    return {
        'messages': messagelist,
        'sessionState':{
            'dialogAction': {
                'type': 'ElicitSlot',
                'slotToElicit': slot_to_elicit,
            },
            'intent':{
                'name': intent_name,
                'slots': slots,
                'state': 'InProgress'
            }
        }
    }

#Slot を話すように促す(選択肢は静的, Amazon Lex に従う)
def elicit_slot_default(slot_to_elicit, intent_name, slots):

    return {
        'sessionState':{
            'dialogAction': {
                'type': 'ElicitSlot',
                'slotToElicit': slot_to_elicit,
            },
            'intent':{
                'name': intent_name,
                'slots': slots,
                'state': 'InProgress'
            }
        }
    }

#Slot を話すように促す(セッション属性追加)
def elicit_slot_sessionAttrib(slot_to_elicit, intent_name, slots,sessionAttributes):

    return {
        'sessionState':{
            'dialogAction': {
                'type': 'ElicitSlot',
                'slotToElicit': slot_to_elicit,
            },
            'intent':{
                'name': intent_name,
                'slots': slots,
                'state': 'InProgress'
            },
            'sessionAttributes':sessionAttributes
        }
    }
```

```

#次のセッションに遷移する(遷移は Amazon Lex に従う)
def next_session(intent_name,slots,sessionAttributes):
    return {
        "sessionState": {
            'dialogAction': {
                'type': 'Delegate'
            },
            'intent':{
                'name': intent_name,
                'slots': slots,
                'state': 'InProgress'
            },
            'sessionAttributes':sessionAttributes
        }
    }

#会話を終了させる
def close_Fulfilled(messagelist, intent_name, slots):
    return {
        'messages': messagelist,
        "sessionState": {
            'dialogAction': {
                'type': 'Close',
            },
            'intent':{
                'name': intent_name,
                'slots': slots,
                'state': "Fulfilled"
            }
        }
    }

#論理式作成 (同時に満たす条件)
def create_logical_expression(sessionAttributes):
    if "LE" in sessionAttributes:
        LE = sessionAttributes["LE"]
    else:
        LE = "Auth(¥"利用者¥¥,¥" group¥",¥" attribute¥)"
    if "group" in sessionAttributes:
        LE.replace("group",sessionAttributes["group"])
        LE.replace("attribute",sessionAttributes["attribute"])
        del sessionAttributes["group"]
        del sessionAttributes["attribute"]
    if "conditionName" in sessionAttributes:
        LE2 = "and (" +sessionAttributes["生成規則"]+ "自然言語")+)"
        if "condition1_LE" in sessionAttributes:
            LE2.replace("引数 1",sessionAttributes["condition1_LE"])
            del sessionAttributes["condition1_LE"]
        if "condition2_LE" in sessionAttributes:
            LE2.replace("引数 2",sessionAttributes["condition2_LE"])
            del sessionAttributes["condition2_LE"]
        if "condition3_LE" in sessionAttributes:
            LE2.replace("引数 3",sessionAttributes["condition3_LE"])
            del sessionAttributes["condition3_LE"]
        del sessionAttributes["conditionName"]
        LE = LE + " and " + LE2
        sessionAttributes["LE"] = LE
    return LE

#論理式作成 (or でつなぎ最終的な論理式の確定)
def create_logical_expression2(sessionAttributes,Device_datalist):
    i = sessionAttributes["アプリリスト番号"]
    Device_datalist["アプリ名リスト"][i]["完成"]=Device_datalist["アプリ名リスト"][i]["論理式"][0]
    for j in len(Device_datalist["アプリ名リスト"][i]["論理式"]):
        Device_datalist["アプリ名リスト"][i]["完成"]=Device_datalist["アプリ名リスト"][i]["完成"]+" or "+Device_datalist["アプリ名リスト"][i]["完成"][j+1]
    return Device_datalist

```

```

#自然言語作成
def create_condition(sessionAttributes):
    if "cond" in sessionAttributes:
        cond = sessionAttributes["cond"]
    else:
        cond = ""
    if "group" in sessionAttributes:
        if sessionAttributes["group"] == "指定しない":
            if sessionAttributes["attribute"] == "指定しない":
                cond = cond + "全員"
            else:
                cond = cond + "全グループの" + sessionAttributes["attribute"] + ""
        else:
            if sessionAttributes["attribute"] == "指定しない":
                cond = cond + sessionAttributes["group"] + "の全員"
            else:
                cond = cond + sessionAttributes["group"] + "の" + sessionAttributes["attribute"] + ""
    del sessionAttributes["group"]
    del sessionAttributes["attribute"]
    if "conditionName" in sessionAttributes:
        cond2 = sessionAttributes["生成規則"]["自然言語"]
        if "condition1_cond" in sessionAttributes:
            cond2.replace("引数 1", sessionAttributes["condition1_cond"])
            del sessionAttributes["condition1_cond"]
        if "condition2" in sessionAttributes:
            cond2.replace("引数 2", sessionAttributes["condition2_cond"])
            del sessionAttributes["condition2_cond"]
        if "condition3" in sessionAttributes:
            cond2.replace("引数 3", sessionAttributes["condition3_cond"])
            del sessionAttributes["condition3_cond"]
        cond = cond + cond2
        del sessionAttributes["conditionName"]
    sessionAttributes["cond"] = cond
    return cond

def LoginIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionId = intent_request['sessionId']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']

    #管理者 ID を聞く
    if slots['ID'] is None:
        nowtime = int(time.time())
        item = {
            "P_key": sessionId,
            "S_key": sessionId,
            "TimeStamp": [{"Time": nowtime, "Status": "管理者 ID"}]
        }
        DB_put(item)
        return elicited_slot_default('ID', intent_name, slots)

    #PASSWORD を聞く
    elif slots['PASSWORD'] is None:
        data = DB_get(sessionId, sessionId)["Item"]
        data = TimeStamp_put(data, "PASSWORD")
        DB_put(data)
        return elicited_slot_default('PASSWORD', intent_name, slots)

    #本来はここで管理者の ID, PASSWORD を検証し認証するべきだが、今回の実装では省略
    #次の状態へ遷移
    else:
        ID = slots['ID']['value']['originalValue']
        sessionAttributes['ID'] = ID
        data = DB_get(sessionId, sessionId)["Item"]
        try:
            predata = DB_get(ID, ID)["Item"]

```

```

        data["TimeStamp"]=predata["TimeStamp"]+data["TimeStamp"]
    except:
        print("エラー")
    finally:
        item = {
            "P_key": ID,
            "S_key": ID,
            "TimeStamp":data["TimeStamp"]
        }
        DB_put(item)

        return next_session(intent_name, slots, sessionAttributes)

def SelectAppIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']
    ID = sessionAttributes['ID']
    data = DB_get(ID,ID)["Item"]
    Device_data = DB_get("デバイス情報",ID)["Item"]
    #アプリ名を聞く(正確にはアプリ情報のアプリ名リストの何番目のアプリかを聞く)
    if slots['App_name'] is None:
        for i in range(len(Device_data["アプリ名リスト"])):
            buttonlist.append(
                {
                    "text":Device_data["アプリ名リスト"][i]["アプリ名"],
                    "value":i
                }
            )
        buttonlist.append(
            {
                "text":"条件判定を終了する",
                "value":"終了"
            }
        )

        title = "どのアプリのアクセス制御条件を設定しますか"
        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": title,
                "buttons": buttonlist
            }
        }
        messageslist=[]
        messageslist.append(message)
        return elicit_slot_sessionAttrib('App_name', intent_name, slots,sessionAttributes)

#次の状態へ遷移する
else:
    i=slots['App_name']['value']['originalValue']
    data = TimeStamp_put(data,Device_data[i]["アプリ名"]+"選択")
    DB_put(data)
    sessionAttributes["アプリリスト番号"] = i
    sessionAttributes["アプリ ID"] = Device_data[i]["アプリ ID"]
    sessionAttributes["アプリ名"] = Device_data[i]["アプリ名"]
    return next_session(intent_name, slots, sessionAttributes)

def NextIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']
    ID = sessionAttributes['ID']
    data = DB_get(ID,ID)["Item"]
    #つぎのステップを聞く
    if slots['NextStep'] is None:
        #(条件未設定)
        if data["List"] is None::

```

```

messageslist = [
  {
    "content": "現在, すべての利用者にアクセス権がありません. グループ, 属性, 条件
の順で指定してアクセス権を付与してください.",
    "contentType": "PlainText"
  }
]
messageslist.append(
  {
    "contentType": "ImageResponseCard",
    "imageResponseCard": {
      "title": "次のステップを選択してください",
      "buttons": [
        {
          "text": "条件を追加する",
          "value": "追加"
        },
        {
          "text": "条件設定を終了する",
          "value": "終了"
        }
      ]
    }
  }
)
#(条件設定済)
else:
  messageslist = [
    {
      "content": "現在の条件は以下のようにになっています.",
      "contentType": "PlainText"
    }
  ]
  i = sessionAttributes["アプリリスト番号"]
  for x in Device_datalist["アプリ名リスト"][i]["自然言語"]:
    messageslist.append(
      {
        "content": x,
        "contentType": "PlainText"
      }
    )
  messageslist.append(
    {
      "contentType": "ImageResponseCard",
      "imageResponseCard": {
        "title": "どうしますか",
        "buttons": [
          {
            "text": "条件を追加する",
            "value": "追加"
          },
          {
            "text": "条件を削除する",
            "value": "削除"
          },
          {
            "text": "条件設定を終了する",
            "value": "終了"
          }
        ]
      }
    }
  )
  return elicit_slot_card('NextStep', messageslist, intent_name, slots)

#次の状態へ遷移する
else:
  next = slots['NextStep']['value']['originalValue']

```

```

data = TimeStamp_put(data,next)
DB_put(data)
if(slots['NextStep']['value']['originalValue']=="終了"):
    Device_datalist = DB_get("デバイス情報",ID)["Item"]
    Device_datalist = create_logical_expression2(sessionAttributes,Device_datalist)
    App_item = {
        "P_key": "デバイス情報",
        "S_key": ID,
        "アプリ名リスト":Device_datalist["アプリ名リスト"],
        "グループリスト":Device_datalist["グループリスト"],
        "属性リスト":Device_datalist["属性リスト"]
    }
    DB_put(App_item)
return next_session(intent_name, slots, sessionAttributes)

def GroupAttribIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']
    ID = sessionAttributes['ID']
    data = DB_get(ID,ID)["Item"]
    GroupAttrib_data = DB_get("デバイス情報",ID)["Item"]
    #グループ取得
    if slots['group'] is None:
        #ユーザ登録しているグループリストを選択肢に追加
        for i in range(len(GroupAttrib_data["グループリスト"])):
            buttonlist.append(
                {
                    "text":GroupAttrib_data["グループリスト"][i]["グループ名"],
                    "value":i
                }
            )
        buttonlist.append(
            {
                "text":"指定しない",
                "value":i+1
            }
        )

    title = "アクセス制御条件を設定するグループを選択してください"
    message={
        "contentType": "ImageResponseCard",
        "imageResponseCard": {
            "title": title,
            "buttons": buttonlist
        }
    }
    messageslist=[]
    messageslist.append(message)
    return elicit_slot_card('group',messageslist, intent_name, slots)

#属性取得
elif slots['attribute'] is None:
    #ユーザ登録している属性リストを選択肢に追加
    for i in range(len(GroupAttrib_data["属性リスト"])):
        buttonlist.append(
            {
                "text":GroupAttrib_data["属性リスト"][i]["属性名"],
                "value":i
            }
        )

    title = "アクセス制御条件を設定する属性を選択してください"
    message={
        "contentType": "ImageResponseCard",
        "imageResponseCard": {
            "title": title,

```

```

        "buttons": buttonlist
    }
}
messageslist=[]
messageslist.append(message)
sessionAttributes['group']=slots['group']['value']['originalValue']
data = TimeStamp_put(data,sessionAttributes['group'])
DB_put(data)
return elicit_slot_sessionAttrib('attribute', intent_name, slots,sessionAttributes)
#次の状態へ遷移する
else:
    sessionAttributes['attribute']=slots['attribute']['value']['originalValue']
    data = TimeStamp_put(data,sessionAttributes['attribute'])
    DB_put(data)
    return next_session(intent_name, slots, sessionAttributes)

def DeleteIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']
    ID = sessionAttributes['ID']
    data = DB_get(ID,ID)["Item"]
    Device_datalist = DB_get("デバイス情報",ID)["Item"]
    App_data = Device_datalist["アプリ名リスト"][sessionAttributes["アプリリスト番号"]]
    #何行目を削除するか聞く
    if slots['delete'] is None:
        buttonlist=[]
        for i in range(len(Device_data["自然言語"])):
            buttonlist.append(
                {
                    "text":Device_data["自然言語"][i][45]+"...",
                    "value":i+1
                }
            )

        title = "どの条件式を削除するか選択してください"
        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": title,
                "buttons": buttonlist
            }
        }
    }
    messageslist=[]
    messageslist.append(message)
    return elicit_slot_card('delete',messageslist, intent_name, slots)
#次の状態へ遷移
else:
    data = TimeStamp_put(data,slots['delete']['value']['originalValue']+"行目の条件削除")
    i = int(slots['delete']['value']['originalValue'])-1
    del Device_data["自然言語"][i]
    del Device_data["論理式"][i]
    for i in range(len(Device_datalist["アプリ名リスト"])):
        if i == sessionAttributes["アプリリスト番号"]:
            Device_datalist["アプリ名リスト"][i]["自然言語"]=Device_data["自然言語"]
            Device_datalist["アプリ名リスト"][i]["論理式"]=Device_data["論理式"]

    App_item = {
        "P_key": "デバイス情報",
        "S_key": ID,
        "アプリ名リスト":Device_datalist["アプリ名リスト"],
        "グループリスト":Device_datalist["グループリスト"],
        "属性リスト":Device_datalist["属性リスト"]
    }
    DB_put(App_item)
    item = {
        "P_key": ID,

```

```

        "S_key": ID,
        "TimeStamp":data["TimeStamp"]
    }
    DB_put(item)
    return next_session(intent_name, slots, sessionAttributes)

def ConditionIntent(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']
    slots = intent_request['sessionState']['intent']['slots']
    sessionAttributes = intent_request['sessionState']['sessionAttributes']
    ID = sessionAttributes['ID']
    data = DB_get(ID,ID)["Item"]
    Device_datalist = DB_get("デバイス情報",ID)["Item"]
    App_data = DB_get("アプリ情報",sessionAttributes['アプリ ID'])["Item"]
    sessionAttributes=create_logical_expression_and(sessionAttributes)

    #アプリ名を聞く
    if slots['conditionName'] is None:
        #決定ボタン
        buttonlist= [
            {
                "text": "現在の条件を決定する",
                "value": "決定"
            }
        ]
        #アプリ固有の条件を追加
        conditionlist=App_data["条件リスト"]
        for i in range(len(conditionlist)):
            buttonlist.append(
                {
                    "text": conditionlist["条件名"],
                    "value": i
                }
            )
        #デフォルトの関数を追加
        buttonlist.append(
            {
                "text": "承認の条件を足す",
                "value": "承認"
            }
        )
        buttonlist.append(
            {
                "text": "曜日の条件を足す",
                "value": "曜日"
            }
        )

        title = "作成中の条件は「 + sessionAttributes["cond"] + 」許可"
        subtitle = "同時に満たす必要がある条件を足すか、現在の条件で決定するか選択してください"
        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": title,
                "subtitle": subtitle,
                "buttons":buttonlist
            }
        }

        messageslist=[]
        messageslist.append(message)
        return elicit_slot_card('condition',messageslist, intent_name, slots)

    elif slots['conditionName']['value']['originalValue']=="決定":

        data = TimeStamp_put(data,"現在の条件で決定")
        item = {
            "P_key": ID,

```



```

        "S_key": ID,
        "TimeStamp":data["TimeStamp"]
    }
    DB_put(item)
    i = sessionAttributes["アプリリスト番号"]
    Device_datalist["アプリ名リスト"][i]["論理式"].append(create_logical_expression(sessionAttributes))
    Device_datalist["アプリ名リスト"][i]["自然言語"].append(create_condition(sessionAttributes))
    App_item = {
        "P_key": "デバイス情報",
        "S_key": ID,
        "アプリ名リスト":Device_datalist["アプリ名リスト"],
        "グループリスト":Device_datalist["グループリスト"],
        "属性リスト":Device_datalist["属性リスト"]
    }
    DB_put(App_item)

    del sessionAttributes["cond"]
else:
    sessionAttributes["conditionName"] = slots['conditionName']['value']['originalValue']
    for i in range(len(App_data)):
        if(App_data["条件リスト"][i]["条件名"]==sessionAttributes["conditionName"]):
            function_data=App_data["条件リスト"][i]
            sessionAttributes["生成規則"]=App_data["条件リスト"][i]["生成規則"]
#引数 1 を聞く
if (slots['condition1'] is None) and (len(function_data["引数リスト"]) > 0):
    if(function_data["引数リスト"][0]["形式"]=="数値の入力"):
        message={
            "content": function_data["引数リスト"][0]["質問"],
            "contentType": "PlainText"
        }
    }
    else:
        buttonlist=[]
        for i in range(len(function_data["引数リスト"][0]["選択肢"])):
            buttonlist.append(
                {
                    "text":function_data["引数リスト"][0]["選択肢"][i]["自然言語"],
                    "value":i
                }
            )

        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": function_data["引数リスト"][0]["質問"],
                "buttons": buttonlist
            }
        }
    }
    messageslist=[]
    messageslist.append(message)
    return elicit_slot_card('condition1',messageslist, intent_name, slots)

#引数 2 を聞く
elif (slots['condition2'] is None) and (len(function_data["引数リスト"]) > 1):
    if(function_data["引数リスト"][1]["形式"]=="数値の入力"):
        message={
            "content": function_data["引数リスト"][1]["質問"],
            "contentType": "PlainText"
        }
    }
    else:
        buttonlist=[]
        for i in range(len(function_data["引数リスト"][1]["選択肢"])):
            buttonlist.append(
                {
                    "text":function_data["引数リスト"][1]["選択肢"][i]["自然言語"],
                    "value":i
                }
            )

```

```

        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": function_data["引数リスト"][1]["質問"],
                "buttons": buttonlist
            }
        }
        messageslist=[]
        messageslist.append(message)
        return elicit_slot_card('condition2',messageslist, intent_name, slots)

#引数 3 を聞く
elif (slots['condition3'] is None) and (len(function_data["引数リスト"]) > 2):
    if(function_data["引数リスト"][2]["形式"]=="数値の入力"):
        message={
            "content": function_data["引数リスト"][2]["質問"],
            "contentType": "PlainText"
        }
    else:
        buttonlist=[]
        for i in range(len(function_data["引数リスト"][2]["選択肢"])):
            buttonlist.append(
                {
                    "text":function_data["引数リスト"][2]["選択肢"][i]["自然言語"],
                    "value":i
                }
            )

        message={
            "contentType": "ImageResponseCard",
            "imageResponseCard": {
                "title": function_data["引数リスト"][2]["質問"],
                "buttons": buttonlist
            }
        }
        messageslist=[]
        messageslist.append(message)
        return elicit_slot_card('condition3',messageslist, intent_name, slots)

else:
    if (len(function_data["引数リスト"]) > 0):
        i=slots['group']['value']['condition1']
        sessionAttributes['condition1_cond']=function_data["引数リスト"][0]["選択肢"][i]["自然言語"]
        sessionAttributes['condition1_LE']=function_data["引数リスト"][0]["選択肢"][i]["論理式"]
    if (len(function_data["引数リスト"]) > 1):
        i=slots['group']['value']['condition2']
        sessionAttributes['condition2_cond']=function_data["引数リスト"][1]["選択肢"][i]["自然言語"]
        sessionAttributes['condition2_LE']=function_data["引数リスト"][1]["選択肢"][i]["論理式"]
    if (len(function_data["引数リスト"]) > 2):
        i=slots['group']['value']['condition1']
        sessionAttributes['condition3_cond']=function_data["引数リスト"][2]["選択肢"][i]["自然言語"]
        sessionAttributes['condition3_LE']=function_data["引数リスト"][2]["選択肢"][i]["論理式"]
    i = sessionAttributes["アプリリスト番号"]
    Device_datalist["アプリ名リスト"][i]["論理式"].append(create_logical_expression(sessionAttributes))
    Device_datalist["アプリ名リスト"][i]["自然言語"].append(create_condition(sessionAttributes))
    App_item = {
        "P_key": "デバイス情報",
        "S_key": ID,
        "アプリ名リスト":Device_datalist["アプリ名リスト"],
        "グループリスト":Device_datalist["グループリスト"],
        "属性リスト":Device_datalist["属性リスト"]
    }
    DB_put(App_item)
    data = TimeStamp_put(data,slots['condition']['value']['originalValue'])
    DB_put(data)
    return next_session(intent_name, slots, sessionAttributes)

```

# インテントのルーティング

```
def dispatch(intent_request):
    intent_name = intent_request['sessionState']['intent']['name']

    if intent_name == 'LoginIntent':
        return LoginIntent(intent_request)
    elif intent_name == 'SelectAppIntent':
        return SelectAppIntent(intent_request)
    elif intent_name == 'NextIntent':
        return NextIntent(intent_request)
    elif intent_name == 'GroupAttribIntent':
        return GroupAttribIntent(intent_request)
    elif intent_name == 'DeleteIntent':
        return DeleteIntent(intent_request)
    elif intent_name == 'ConditionIntent':
        return ConditionIntent(intent_request)
    else :
        raise Exception('Intent with name ' + intent_name + ' not supported')

def lambda_handler(event, context):
    return dispatch(event)
```