applied sciences

MDPI

Article

Enhancing Conversational Agent Development Through a Semi-Automatization Development Proposal

Ángel Martínez-Gárate [1], José Alfonso Aguilar-Calderón [2,*], Carolina Tripp-Barba [2] and Aníbal Zaldívar-Colado [2]

[1] Tata Consultancy, Zapopan 45080, Mexico; angelmartinez.facite@uas.edu.mx
[2] Facultad de Informática Mazatlán, Universidad Autónoma de Sinaloa, Mazatlán 82017, Mexico; ctripp@uas.edu.mx (C.T.-B.); azaldivar@uas.edu.mx (A.Z.-C.)
* Correspondence: ja.aguilar@uas.edu.mx

Abstract: The development of chatbots is often hindered by high costs and time consumption, with existing technologies and frameworks not fully addressing the diverse requirements for creating robust conversational agents across platforms like WhatsApp. This is a limitation for the implementation of these frameworks in a professional environment, thus representing both opportunities and complexities. This article presents improvements to the Xatkit framework, utilizing Model-Driven Development techniques to simplify chatbot development and extend support for WhatsApp. Additionally, the integration of the GPT-3 autoregressive language model enhances user interaction, offering a more sophisticated and responsive conversational experience.

Keywords: AI applications; AI in software engineering; chatbot development; conversational agent framework; chatbot semi-automatization; low code

check for updates

Academic Editors: Stefan Fischer and Douglas O'Shaughnessy

Received: 4 December 2024
Revised: 19 January 2025
Accepted: 20 January 2025
Published: 23 January 2025

Citation: Martínez-Gárate, A.; Aguilar-Calderón, J.A.; Tripp-Barba, C.; Zaldívar-Colado, A. Enhancing Conversational Agent Development Through a Semi-Automatization Development Proposal. *Appl. Sci.* **2025**, *15*, 1139. https://doi.org/10.3390/app15031139

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A conversational agent, also called chatbot or bot, is a computer program that simulates a conversation; it can communicate with the user through different means, such as voice or text. In [1], it was described as a system that can interact with human users with natural language. However, as [2] points out, it is software that imitates human conversations using artificial intelligence. Its use has grown rapidly in recent years in areas such as education, customer service, and entertainment. However, for the development of chatbots, it is necessary to have proficiency in several specialized areas of knowledge [3], including Software Engineering (SE), Usability, User eXperience (UX), Machine Learning (ML), and Natural Language Processing (NLP) in order to establish the best interaction with the user—making the development process complex [3].

Many technologies and techniques have emerged over the years to improve the design, development, and implementation of conversational agents. A primary goal is to bridge the gap between expert bot developers and users who lack experience in software development [4]. As a critical tool for identifying common errors in software development, modeling has become increasingly important. Using graphical representation techniques or concepts that are accessible to people without technical programming knowledge, it facilitates detailed specification and analysis of a program's functionality. In this context, Model-Driven Development (MDD) has emerged as a paradigm for semi-automated software development. Its goal is to generate source code from functional specifications represented in graphical models [5]. By clarifying the intended behaviors and actions of software products before coding, MDD improves productivity compared to traditional

methods. The collaborative construction of models by the development team results in a clear definition of the structure and functionality of the software. As a result, the processes of testing, rebuilding, and deploying applications can be accelerated. This is especially true when developing multiple applications simultaneously.

Research has increasingly focused on creating comprehensive frameworks for chatbot development, addressing challenges in reducing the time and effort required to build conversational agents [6]. A major hurdle is the complexity arising from the diverse technologies and target platforms involved in chatbot creation, which complicates the development of an integrated environment that accommodates all existing tools. These challenges include ensuring platform compatibility and managing migration issues between bots developed with different approaches. To address these challenges, metamodels have been developed to specify new target platforms and development technologies, particularly in natural language processing. However, understanding how metamodels define syntax and semantics remains a major research challenge. Future work will need to focus on model analysis techniques, performance evaluation, and criteria for high-quality models. Traceability techniques to synchronize system changes with model updates remain an open research area.

To democratize access to conversational agents, frameworks that cater to non-expert users are essential. Current platforms typically require substantial technical expertise, limiting their use to skilled developers. By developing tools that simplify chatbot creation, non-technical users will be able to build custom solutions without specialized knowledge, benefiting industries such as customer support, sales, and marketing. Bridging academic and industry gaps is crucial to accelerate the adoption of these frameworks, allowing faster technology transfer, driving innovation, and improving customer engagement while reducing development costs. This approach supports industry competitiveness by facilitating the integration of powered chatbots into business operations.

The success of frameworks like Xatkit highlights the potential of empowering non-expert users to create custom chatbots, significantly accelerating automation and enhancing customer engagement. Xatkit offers a low-code environment that simplifies chatbot development, reducing the need for advanced programming skills, a critical advantage for organizations seeking to deploy bots efficiently and without technical hurdles [7]. As a chatbot orchestration platform, Xatkit enables developers to flexibly integrate various NLP tools, facilitating seamless transitions between different technologies while maintaining a modular approach. This modularity allows developers to focus on conversation logic rather than technical implementation, thereby improving user engagement and customer support.

Despite these strengths, Xatkit faces notable limitations, particularly in its lack of native support for widely used instant messaging platforms such as WhatsApp. This limitation reduces the broader application of the framework, especially in regions where WhatsApp dominates as the primary communication medium. In January 2024, WhatsApp had more than two billion active users worldwide, with significant adoption in countries such as India (394 million users), Brazil (111 million users), and the United States (77 million users) [8]. The absence of WhatsApp integration limits the utility of Xatkit in key international markets, thus limiting its potential impact in regions where this platform is integral to communication and customer interaction.

This work presents an enhancement to the Xatkit framework [6], a chatbot development platform, addressing its prior lack of WhatsApp integration, a crucial gap given WhatsApp's global dominance. Using MDD, Xatkit's metamodel is extended using Ecore models [9] to formally define chatbot properties, allowing for a structured specification of both structural and behavioral aspects. With a metamodel defined in Ecore and implementing Acceleo M2T transformations [10], this approach can automate the generation

of Java code for chatbots. This automation can reduce the manual coding effort required by developers, as they are only responsible for defining the chatbot's properties within the model. Automation of the code generation process, a key aspect of MDD, not only can reduce the time needed for manual coding and debugging but also can reduce the probability of human errors, leading to more reliable and maintainable code [11]. Furthermore, since the logic and structure of the chatbot are encapsulated within the model, any updates or modifications can be efficiently made at the model level, with the code being automatically regenerated to reflect these changes. The benefits of MDD have been proved in empirical studies such as [12,13]. Critically, we integrate the large language model GPT-3 [14], endowing Xatkit with advanced natural language processing capabilities to generate contextually rich and nuanced responses to provide better interaction with the user when the exact answer to its query is not considered in the programmed chatbot. Communication with WhatsApp [15] is facilitated by integrating it with the Twilio API (Application Programming Interface), which provides Xatkit support for WhatsApp. These combined advances—metamodel extension, automated code generation, and GPT-3 integration, coupled with Twilio API [16] integration for WhatsApp connectivity—transform Xatkit into a more powerful, efficient, and versatile platform for developing sophisticated conversational agents on various platforms, now including the widely used WhatsApp.

The target audience of this research work refers to, but is not limited to, software engineers, information technology managers, independent research developers in the area of conversational agents, and research students.

This article is structured as follows: Section 2 provides the foundational background; Section 3 presents a comprehensive review of related work; Section 4 details the proposed approach; Section 5 presents the results obtained from the Xatkit enhancement, illustrating the proposal through a practical application; and finally, Section 6 offers concluding remarks and outlines ongoing and future work.

2. Fundamentals

This section presents an overview of the fundamental concepts pertinent to research, thus establishing a foundation to understand the significance and innovation of the proposed solution.

2.1. Model-Driven Development

Model-Driven Development (MDD) is a paradigm for software development centered on models as primary development artifacts [17]. A model, defined as a formalized representation of an object, concept, or system within a specific medium [18], serves as a powerful tool to understand complex concepts by illustrating key characteristics. Unlike traditional coding-centric approaches, MDD emphasizes system construction through modeling, using tools to automatically transform models into executable code, thus reducing software production costs and effort [5] while improving application quality across diverse platforms. This is achieved by separating the logic of the problem from the technical details specific to the platform through abstraction. Crucially, at the core of MDD lies the metamodel, a model representing another model. Using the Object Management Group (OMG) as a model specifying the abstract syntax of a modeling language [19], the metamodel defines the fundamental concepts, rules, and components used to define other models, essentially specifying the elements available to developers during model creation, thus establishing the foundational principles and structures for subsequent model development.

2.2. Xatkit: A Multimodal Low-Code Chatbot Development Framework

The Xatkit [20] Multimodal Low-Code Chatbot Development Framework, previously known as Multi-Platform Chatbot Modeling and Deployment with the Jarvis Framework, is a framework designed for the development of low-code chatbots. The proposal referenced in [21] is another of the most frequently cited in the literature. This is an open-source framework for the development of MDD-based conversational agents. The proposed methodology offers a fundamental structure that must be established for each development undertaking.

Xatkit offers significantly enhanced flexibility in the design and implementation of conversational agents compared to platform-restricted tools such as Dialogflow. As an open-source framework, Xatkit provides complete access to its source code, allowing extensive customization and modification, and supports a customized development process adaptable to specific project needs. This flexibility extends to automated deployment and management of conversation logic across various platforms, facilitating changes without requiring alterations to the underlying code, a key advantage when switching NLP providers, unlike the constraints imposed by tools such as Dialogflow. Using MDD strategies, Xatkit defines platform-agnostic agents, ensuring adaptability throughout development and enabling the independent evolution of components due to its modular architecture, thus facilitating seamless integration of new features and updates. This combination of customization, flexible development, modular design, and simplified platform switching without altering conversational logic makes Xatkit particularly well suited for complex and sophisticated conversational agent projects and justifies its selection over tools such as Dialogflow in this work.

Xatkit Metamodel

The Xatkit metamodel facilitates the development of conversational agents by providing a conceptual framework integrated with Java, a widely used programming language. This integration allows developers to leverage existing Java tools and libraries during the creation, debugging, and testing of chatbots, simplifying the development process. The metamodel's extensibility allows integration with various NLP providers and external services, providing a standard framework for defining chatbot components and ensuring manageable, scalable systems. Key components include the following:

1. Dialog Flows: Define the conversational paths and decisions during user interactions.
2. Intents: Capture the user's goals or purposes, such as seeking information or executing actions.
3. Entities: Represent specific data points (e.g., dates, locations) necessary for understanding user input.
4. Actions: Specify the chatbot's responses to inputs, including database queries or API calls.
5. Contexts: Maintain the conversation state, enabling contextually aware interactions.
6. Slots: Act as placeholders for user-provided values like dates or times.
7. Variables: Store temporary data for dynamic interactions.
8. User Inputs and Outputs: Cover the interaction modes (e.g., text, voice) and responses (e.g., text, images).

Xatkit's Platform package delineates the functionalities of novel platforms, enabling the definition of executable actions and the emission of events based on the provider definition (class 'Provider Definition'). These capabilities are explicitly employed within the execution model to facilitate platform interaction. As illustrated in Figure 1, initial platform definition (class 'Platform') is crucial, requiring a name attribute and a path used by the Xatkit Runtime (the event-driven execution engine responsible for the deployment

and management of chatbots) to establish a connection between the model and the implementation of the platform. Furthermore, a set of actions (class 'Action Definition'), each requiring parameterized input (class 'Parameter' with a key attribute), must be defined for each path. The metamodel supports the creation of abstract platforms, devoid of concrete action implementations, to represent families of similar platforms, thus enabling generic bot definition. Each platform implementation is realized as an independent Java project utilizing the Xatkit platform interface. This interface enables the Xatkit Runtime to dynamically load the platform, instantiate actions, and execute them upon satisfaction of specific conditions. Each implementation comprises a principal Java class that contains access data for its permitted actions and a set of classes that represent each executable action. Communication with the various platforms is typically achieved through REST (Representational State Transfer) requests.

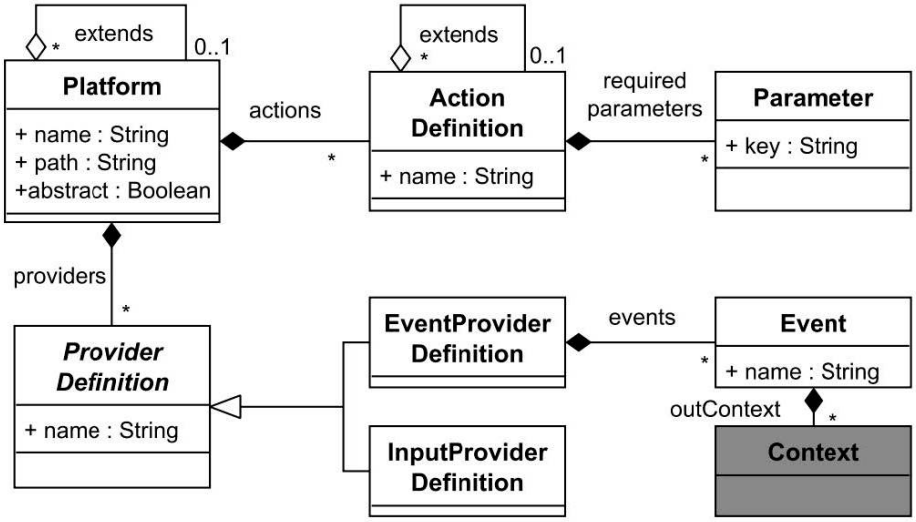


Figure 1. Diagram of the Xatkit metamodel where the platform package is defined. The asterisk at the end of an association indicates one or many-to-many relationship.

2.3. Autoregressive Language Model (GPT-3)

GPT-3 (Generative Pre-Trained Transformer) is an advanced autoregressive language model developed by OpenAI [14]. Using deep neural networks, it simulates text generation that is analogous to that of a human writer. With 175 billion parameters, the model demonstrates the capability to handle a broad range of tasks, including text classification, natural language understanding tasks, question answering, text summarization, and complex conversational interactions. As one of the most advanced models for NLP, GPT-3 has markedly influenced the evolution of intelligent conversational systems, expanding the potential of virtual assistants and AI-driven dialogue systems in a range of applications.

2.4. Twilio

Twilio is a cloud-based communication platform that facilitates the transmission of text messages, voice calls, video conferencing, and emails through an intuitive API [16]. One of its most notable attributes is its integration with WhatsApp, which facilitates uninterrupted communication through this widely used platform. Twilio's architectural design is highly flexible, allowing businesses to integrate it into their existing systems with minimal effort while offering extensive customization options to enhance user experiences. In addition, Twilio provides organizations with comprehensive analytics and tracking tools that enable them to monitor, evaluate, and optimize their communication strategies effectively.

3. Related Work

This section provides detailed information on articles that explore the use of MDD techniques to automate the development of conversational agents. The section summarizes various perspectives on current research and development trends, highlights the key advantages, and addresses the challenges that need to be addressed.

Pérez et al. [3] provide an in-depth analysis of chatbot development tools, categorizing selection factors into technical and administrative considerations. The study highlights challenges in tool selection, particularly when adding unforeseen features, often necessitating migration to alternative platforms. It emphasizes the need for improved design patterns, quality metrics, and testing methods, especially for closed-domain bots reliant on predefined training phrases, prompting the W3C to pursue standardization in chatbot development.

The authors in [22] introduce CONGA, an Integrated Development Environment (IDE) with a DSL, a specialized programming language tailored to the specific needs, terminologies and constraints of the conversational agent domain, which allows tool-agnostic chatbot modeling. The CONGA recommendation module guides development, and its code generator extracts properties from conversational agent models. However, its reverse engineering process is limited to Dialogflow. Furthermore, CONGA requires manual intervention for certain elements, such as emojis, and faces challenges in automatic migration due to differences in native technologies.

In [23], an MDD tool is presented to generate Facebook chatbots. It employs an ECORE metametamodel and Model-to-Text (M2T) transformation rules, which support integration with REST services. However, the tool lacks an NLP provider, limiting its natural language processing capabilities.

The research by [24] introduces a framework for video game conversational agents using a DSL with custom natural language expressions, implemented as an Eclipse-based tool. Although effective within the gaming context, its applicability is limited to specific cases and a single language.

In [25], a tool is presented to develop chatbots that interact with OpenAPI using the Xatkit platform. The tool converts OpenAPI specifications into UML models that define the agent. Despite its automation potential, the tool supports only specific API types and requires manual configuration for incomplete data.

The work presented in [26] describes a framework for goal-oriented chatbots that manage discussion groups, using Python classes for various types of interaction. However, it requires programming expertise, and changes require reloading of the entire transcript file. The validation of the framework was performed through a "Wizard of Oz" experiment, which lacks real-world application examples.

The study by [27] generalizes Xatkit's DSL to support various types of conversational agents, facilitating the design of complex user interaction flows. The framework integrates an Interaction Flow Modeling Language (IFML) extension for describing interactions with other software components, promoting semi-automated chatbot creation. However, the metamodel needs improvements, such as access control features.

In [28], the authors propose a scheme for chatbots to perform natural language queries, linking intents to query types through an EMF plugin integrated with Xatkit. This setup requires a domain metamodel and synonym-enhanced models for effective NLP training, extending the capabilities of Xatkit for query-based chatbots.

The architecture presented in [29] defines metamodels for NLU services, such as Dialogflow, allowing developers to easily switch NLP providers. The metamodel supports role and entity group definitions, enhancing chatbot customization.

The framework in [30] automates natural language support for DSLs through chatbots. It generates a general configuration, refined manually by developers, before synthesizing and implementing the chatbot, streamlining conversational agent integration in DSLs.

The research by [31] introduces a framework for the creation of chatbots using deep learning for text classification and generation, supporting generative and retrieval-based approaches. The bot undergoes a training phase, building a knowledge base, with a focus on the development of generative chatbots.

The study in [32] proposes a framework for conversational assistants to aid in software interactions. Using MDD techniques, it generates components from models, simplifying the creation process for developers with limited knowledge of NLP. However, it lacks NLP services that use deep learning. In this context, ref. [33] introduces a standardized graphical dialog flow symbology, enhancing traceability and documentation. It includes transformation rules to convert between Watson Assistant dialog flows and the proposed model, streamlining chatbot flow development.

In [34], an adaptable architecture integrates an empathetic controller using dialogue history to tailor responses based on user emotions. It employs data-driven generative models and reinforcement learning, enabling the development of emotionally intelligent conversational systems.

Lastly, ref. [35] presents an architecture with two new programming languages for voice bots and chatbots, based on DSL concepts. These languages offer flexibility, operating independently of specific systems or hardware, facilitating conversational agent development.

The growing interest in conversational agent development has driven research into various methodologies, including frameworks emphasizing reuse and scalability, DSL-based integration, and NLP-enhanced custom architectures. While generative chatbots with broad conversational capabilities are being explored, standardization in chatbot construction remains a key challenge. As evidenced in Table 1, although previous research has focused mainly on proposing new methods, the evaluation and validation of existing approaches is increasingly recognized as crucial. Prioritizing this evaluation research is essential to develop robust and reliable tools and to foster greater standardization, ultimately leading to more impactful advancements in the field.

Table 1. Chatbot development proposals through the years.

Year	Reference	Advantages
2006	[32]	Framework for developing conversational assistant agents with minimal NLP knowledge required, leveraging Model-Driven Architecture (MDA) techniques.
2019	[30]	Integration of natural language support for DSLs through chatbots, streamlining the modeling process and enhancing usability.
2020	[33]	Utilizes MDD concepts to map chatbot conversation flows, enhancing traceability, testing, maintenance, and documentation.
2020	[36]	Automated approach for building conversational agents from OpenAPI specifications, reducing development time and effort.
2021	[3]	Analysis of essential tools for chatbot development, highlighting the need for standardization and improved design patterns.
2021	[22]	CONGA IDE with DSL for platform-independent chatbot modeling, including a recommendation module and code generator.
2021	[23]	Tool for generating Facebook chatbots using MDD approach, enhancing interoperability with REST services.
2021	[24]	Framework for video game conversational agents using DSL, tailored for specific gaming contexts.
2021	[25]	Tool for developing chatbots with OpenAPI integration, supporting multiple API types and automated development.
2021	[26]	Framework for goal-oriented chatbots in discussion groups, offering configurable interaction types.
2021	[27]	Generalization of Xatkit's DSL to support complex user interactions, integrated with IFML for system component relationships.
2021	[28]	EMF plugin integration with Xatkit framework for query-based chatbots, enhancing natural language query capabilities.
2021	[29]	Metamodel architecture for NLU services, enabling easy switching between providers and flexible chatbot customization.
2022	[34]	Architecture for empathetic conversational agents using data-driven generative models and reinforcement learning for emotional intelligence.
2022	[31]	Framework for generative chatbots using deep learning for text classification and generation, supporting generative and retrieval-based approaches.
2022	[35]	Introduction of two programming languages for voice bots and chatbots, independent of operating systems and cloud environments.

4. Proposal

This proposal introduces an MDD approach to semiautomatization for chatbot development within the Xatkit Chat Platform framework, specifically targeting instant messaging applications. Using MDD's inherent advantages in abstraction and automation, the proposed solution streamlines conversational agent creation through the definition of a metamodel and a series of model-to-text transformations using generic templates. Furthermore, the integration of the GPT-3 autoregressive language model enhances the framework by enabling automated response generation, effectively addressing instances where the chatbot's predefined knowledge base is insufficient, and thereby facilitating more dynamic and contextually relevant user interactions. This MDD-based methodological framework was adopted primarily to reduce development time and promote higher levels of abstraction. MDD requires less development effort while achieving similar or better design quality, leading to a reduction in developer errors and an increase in productivity [12,13] (for a detailed discussion of the principles and benefits of MDD, see Section 2.1). This proposal extends Xatkit's capabilities, positioning it as a versatile tool for developing conversational agents deployable on WhatsApp, one of the world's most popular messaging platforms, but also addresses a critical gap in platform support. The choice of Xatkit is further justified by its inherent strengths within the current landscape of conversational agent development tools, which often present varying functionalities and platform limitations. Xatkit's key features, including model-based agent generation via M2T transformations, open source extensibility through a multiplatform platform package, DSL integration for customization and modularity, and support for integrating external NLP providers, provide new functionalities for developers that align perfectly with the goals of this research.

Figure 2 illustrates the proposed approach. The transformation process commences with the conceptual definition of the conversational agent, represented as an Ecore model conforming to the defined metamodel. This model abstractly and independently of the platform encapsulates key agent elements, including intents, states, transitions, and responses, and incorporates user-specified Twilio credentials (username and token). Subsequently, an Acceleo-based template transforms this model into executable Java chatbot source code, implementing the agent and containing the API call to GPT-3. The generated source code is then integrated with the Xatkit platform via the Xatkit engine, enabling deployment and operation on target platforms such as messaging applications or web interfaces. This process culminates in the final chatbot code. The subsequent sections provide a detailed description of each component, elucidating their respective roles and contributions to the overall methodology.

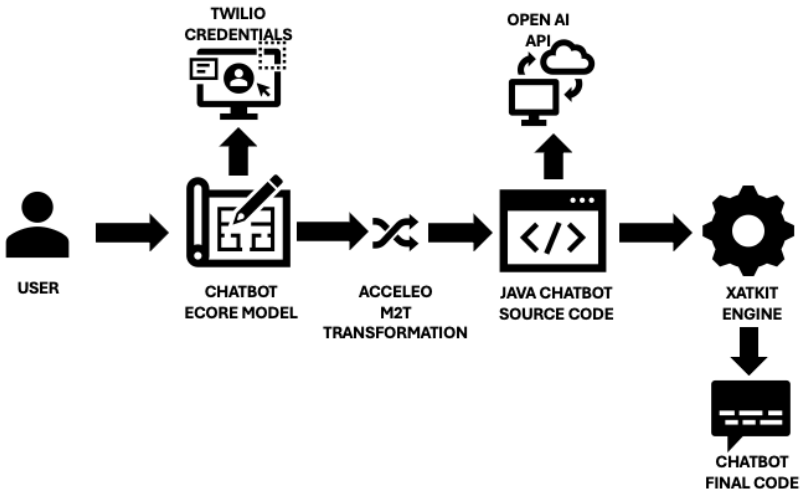


Figure 2. MDD chatbot semiautomatic development process.

4.1. Definition of a Domain-Specific Language Through the Implementation of a Metamodel

Within this proposal, a DSL was meticulously defined and implemented as a metamodel (MM) designated XatbotMM. This metamodel was developed using the Eclipse Modeling Framework (EMF), a component of the Eclipse Modeling Project (EMP) [37], and encapsulates the core abstractions and relationships necessary to define and manage conversational agents. Through this metamodel, chatbots can be conceptually represented as models composed of instances of its metaclasses. Figure 3 shows the design of the XatbotMM metamodel implemented in Eclipse. This metamodel comprises nine precisely defined metaclasses, each representing a distinct concept within the conversational agent domain of the Xatkit Chat Platform framework.

- Platform: Specifies the platform used by the bot to interact with users, such as WhatsApp or React, as supported by XatbotMM.
- Provider: Represents the NLP provider. Currently, it supports integration with DialogFlow and NLP.JS, the latter being a library for natural language understanding tasks such as entity extraction, sentiment analysis, and automatic language detection.
- Intents: Captures the conversational capabilities of the agent through attributes such as intent names and training sentences, which are essential for the NLP provider to recognize user inputs accurately.
- State: Defines the states of the bot within the conversational flow.
- Transition: Models the possible state transitions that determine the bot's behavior.
- ConditionalTransition: Allows transitions between states to be triggered based on specific conditions.
- TestMessage and MultimediaMessage: Represent test cases and multimedia content, respectively, which can be included in the conversation logic.

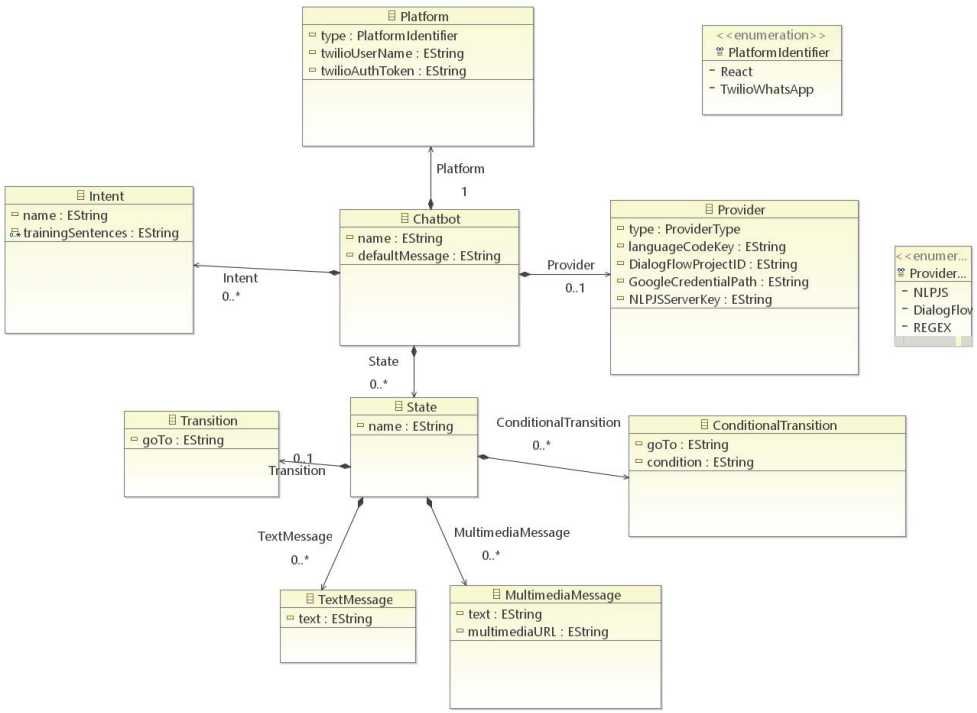


Figure 3. The proposed XatbotMM. The asterisk at the end of an association indicates one or many-to-many relationship.

Through the use of XatbotMM, multiple conversational agents can be efficiently constructed by defining models that specify their logic. Users need only define the intents, training sentences, states, transitions, and corresponding responses in the model. These

high-level abstractions streamline the chatbot development process, reducing the complexity of implementation, and promoting consistency.

Regarding the structure and behavior of the agent, the agent's logic is defined through four states, which collectively dictate its behavior:

- stateInit: This state serves as the initialization point of the chatbot. Upon activation, the bot performs any necessary setup tasks and transitions control to the next state.
- stateWaiting: In this state, the agent waits for the user's input. It analyzes the text received to determine whether it corresponds to a greeting or a request for information about the agent.
- stateGreeting: If user input is classified as a greeting, the chatbot transitions to this state. Here, it generates and delivers an appropriate response to acknowledge the user's salutation.
- statePresentation: Upon user inquiry about the chatbot itself (statePresentation), the agent provides relevant self-information before returning to the stateWaiting state, ensuring continuous readiness for subsequent user input. This process is also followed after the initial greeting (stateGreeting).

For integration with NLP and deployment platforms, the model includes several additional components to enhance its functionality.

- NLP Provider: The chatbot's ability to interpret user input and generate responses relies on an NLP provider. In the model, this element specifies the NLP service to be utilized, enabling intent recognition and language understanding.
- Platform Specification: In the model, a platform element is defined to determine the deployment environment of the agent. In this example, the chatbot is configured to operate on the React platform, ensuring compatibility with this widely used development framework.

Figure 4 presents an example of an ecore model of a conversational agent. This model was created using the model editor available in the Eclipse framework, using the Eclipse EMF plugin. The model showcases the capabilities of a chatbot that can greet users and provide an introduction during interactions.

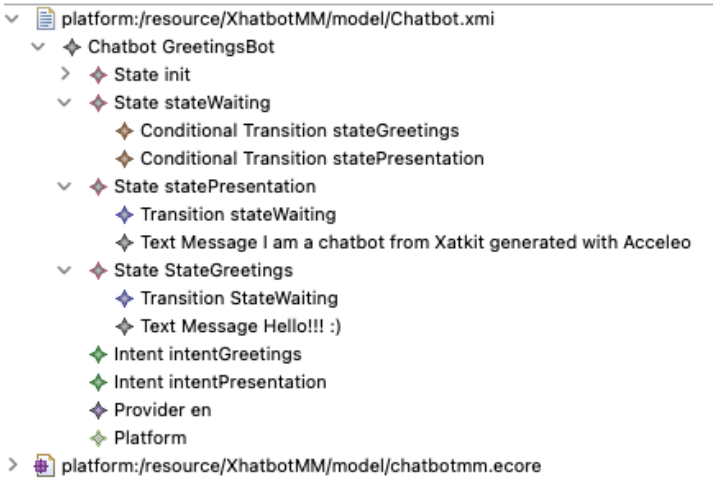


Figure 4. Example of model defined from XatbotMM.

The final step in the implementation process involves transforming the conceptual model into executable code. This transformation is facilitated by the Acceleo Model-to-Text (M2T) plugin, which is integrated into the Eclipse framework. The M2T template defined for this project converts the model, adhering to the metamodel's structure, into Java

source code. This automated process ensures that the chatbot's conceptual design is faithfully translated into a functional implementation, minimizing the likelihood of errors and reducing development time.

Model-to-Text Transformations for Source Code Generation

In the proposal presented in this article, a meticulously designed M2T transformation template is presented to generate Java source code from XatkitMM metamodel instances, utilizing the Acceleo M2T plugin within the Eclipse framework. The transformation extracts essential information from XatkitMM-compliant models to build and operationalize chatbots in the Xatkit framework, ensuring versatility across various models. Acceleo features, such as reusable functions, conditional statements, and loops, enable precise and flexible code generation tailored to the requirements of the chatbot model. Developers only need to define the chatbot logic, while Acceleo automates the transformation into Java code, including the necessary libraries for platform integration and NLP services. For example, the transformation template imports essential libraries required for the following: (1) integration with the Xatkit platform, (2) compatibility with the selected NLP provider, and (3) execution of platform-specific functionalities, such as message handling and user interaction workflows.

By automating these steps, the transformation reduces the potential for human error and accelerates the deployment process. Figure 5 illustrates a segment of the Acceleo template designed for the M2T transformation. This snippet demonstrates the use of Acceleo's functions, conditional logic, and iterative constructs to translate model elements into well-structured Java source code.

```
  public static void main(String ['['/][']'/] args) {

[generateIntentDefinitions(aChatbot)/]
[generatePlatformDefinitions(aChatbot)/]
[generateStatesDefinitions(aChatbot)/]
[generateStatesBehaviour(aChatbot)/]
[generateDefaultFallback(aChatbot)/]
[generateBotModel(aChatbot)/]

      Configuration botConfiguration = new
 BaseConfiguration();

[generateIntentProviderConfigurations(aChatbot)/]
[generatePlatformConfigurations(aChatbot)/]

      XatkitBot xatkitBot = new XatkitBot(botModel,
 botConfiguration);
      xatkitBot.run();
      /*
       * The bot is now started, you can check http://
localhost:5000/admin to test it.
       * The logs of the bot are stored in the logs
folder at the root of this project.
       */
  }
```

Figure 5. Part of the implementation of the M2T transformation using the Eclipse Acceleo plugin.

The implementation of a robust M2T transformation within the Xatkit framework underscores the value of MDD in the development of conversational agents. By automating the transition from conceptual models to executable code, developers can focus on designing high-quality chatbot logic while minimizing time-consuming and error-prone manual coding tasks. This approach not only enhances efficiency, but also promotes scalability and maintainability in the development of conversational systems.

4.2. Integration with GPT-3 as an Alternative for Resolving User Queries

Closed-domain conversational agents, relying on predefined intents and training sentences to classify user queries, exhibit inherent limitations in addressing unanticipated

requests. Although effective for common, predefined questions, this approach struggles with user requests not explicitly considered during development, hindering proper NLP service detection and response. Consequently, research has explored alternative strategies to enhance query resolution beyond predefined intents. One such approach, as detailed in Section 3, involves consulting external documentation (e.g., manuals, FAQs) owned by the bot provider [38]. This expands the bot's knowledge base, potentially transitioning it towards open-domain capabilities. When these sources prove insufficient, recourse to general knowledge sources, such as Transformer models, becomes a viable option. These models, which are capable of generating human-like text, offer broader knowledge bases applicable to a wider range of user queries.

To mitigate the aforementioned limitations of closed-domain conversational agents, this article introduces, as a key novelty, the implementation of the autoregressive natural language model GPT-3 [39], a model trained on approximately 175 billion parameters with extensive knowledge of the cross domain. GPT-3 was selected as a robust alternative for addressing user queries when the primary natural language processing service is unable to detect user intent and generate a suitable response. The proposed methodology leverages GPT-3 in instances where the agent encounters unclassifiable messages. Instead of simply replying with a default "cannot answer" message, the bot offers contextually related information, guiding the user towards reformulating their query and maintaining conversational flow. This approach aims to re-engage the user and mitigate the negative impact of failed intent detection.

The first step consisted of designing a suitable input for the model, considering OpenAI's recommendations for designing a good prompt. Subsequently, a series of tests were performed for different question scenarios, and it was concluded that the model should contain four elements to be able to generate useful suggestions for the user when the bot requires it. The elements defined are as follows:

- Provide information about the context in which the chatbot is being used. This information can be extracted directly from the chatbot's training sentences and can be implemented in the form of a list of options in the prompt.
- Precise instructions on how the model should act. Here, you specify in brief and clear words the task that the model should perform—for example, in the prompt, you detail how it is explicitly instructed to generate a suggestion from the user's message. The suggestion should be selected from the list of options mentioned above. The instructions will depend on the needs or preferences of the chatbot designer; for example, it can be specified to generate the suggestions in a specific language or use the same language as the user's message.
- A list of examples. This list shows how the suggestions are expected to be generated. The examples should follow the instructions of the previous item. Generally, increasing the number of examples results in a higher quality of suggestions, but it should be considered that increasing the number of characters in the prompt implies a higher cost and response time.
- The user's message. Finally, at least the last message written by the user should be included. The reason for doing it this way is that it is the key piece of information from which the model will generate a suggestion.

In most scenarios between the user and the conversational agent, the first three elements will be static, i.e., previously defined, while the fourth will change dynamically according to the conversation. The least complicated way to do this is to concatenate the last message at the end and add a start sentence so that the model generates the text from what was previously displayed.

The three elements mentioned above were designed for a prompt that will be used in one of the GPT-3 base models. If fine tuning is planned, elements 1 and 2 should be replaced or removed, and a large number of instances of element 3 should be used instead. In addition, it is recommended to filter the text generated from the model to standardize the response format, in order to prevent or minimize inappropriate responses, e.g., by checking the length of the text provided by the model or checking if it contains inappropriate words. In addition, use can be made of the temperature parameter and stop sequences at the convenience of the chatbot designer. By following this procedure, the recommendation system can be built quickly regardless of the area of application of the conversational agent. However, it is important to mention that this proposal was designed especially for chatbots used in customer service, so adaptations may be necessary depending on the designer's objectives. Figure 6 shows an example in which this proposal is applied, in which the OpenAI playground is used to generate a suggestion; the text formed by GTP-3 is shown in green.

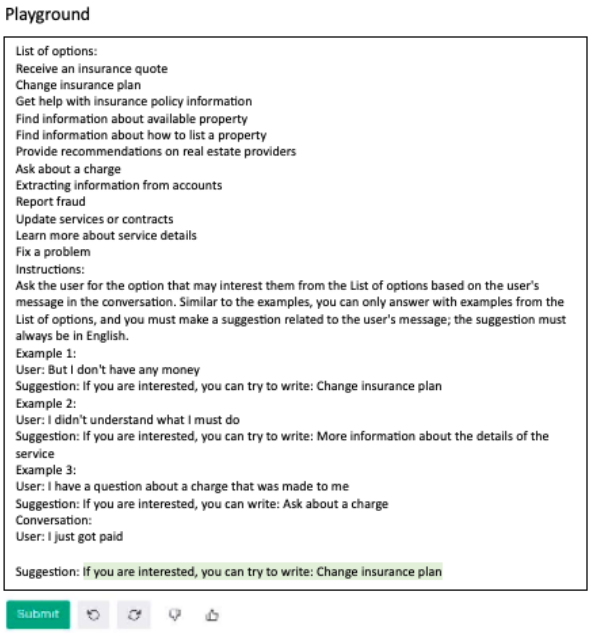


Figure 6. Prompt recommendation example.

GPT-3 Integration with Xatkit Chat Platform

The procedure for generating suggestions via GPT-3, as detailed in this article, was seamlessly integrated into a Xatkit project to enhance the responsiveness of conversational agents in scenarios where user input cannot be classified into predefined intents. This integration represents a significant step toward improving the adaptability and user experience of chatbots developed within the Xatkit framework.

The suggestion generation process is activated when a user submits a message that the NLP service fails to classify into any of the intents defined in the conversational agent's model. In such cases, the system transitions to the fallbackState, a designated state where the suggestion generation mechanism is executed. Within the fallbackState, a dynamic prompt is constructed to facilitate the GPT-3 text generation process. The prompt comprises three predefined components that establish the context and structure for the generation of GPT-3 response. To personalize the prompt, the user's message, retrieved programmatically through the context.getIntent().getMatchedInput() method, is appended when necessary. This approach ensures that GPT-3 has sufficient context to generate relevant and coherent suggestions aligned with the user's query. Once the prompt is completed, a request is sent to the OpenAI API to initiate the text generation process. To enable seamless communication

between the Xatkit project and the OpenAI GPT-3 service, the TheoKanning openai-java library [40] is used. This library serves as a robust interface for interacting with the OpenAI API within Java-based projects.

The workflow for generating GPT-3-based suggestions within the Xatkit framework can be summarized as follows:

1. User Input Detection: The conversational agent identifies user messages that do not match any predefined intents, triggering the fallbackState.
2. Prompt Construction: A dynamic prompt is created, incorporating predefined components and user input retrieved from the agent context.
3. API Request: The constructed prompt is sent to the OpenAI API via the TheoKanning openai-java library.
4. Text Generation: GPT-3 processes the prompt and generates a suggestion or response.
5. Response Delivery: The generated suggestion is returned to the conversational agent, who provides it to the user.

The integration of GPT-3 into the Xatkit framework addresses a common limitation in rule-based conversational agents: their inability to handle unclassified user inputs effectively. By invoking GPT-3 in fallback scenarios, the system ensures that users receive meaningful responses even when predefined intents are insufficient. Furthermore, the use of a dynamic prompt construction mechanism improves the flexibility and relevance of generated suggestions, aligning them with user queries while maintaining the conversational context. This approach not only improves user satisfaction, but also demonstrates the potential to combine traditional MDD principles with cutting-edge AI technologies to create more adaptive and intelligent conversational systems.

4.3. Integration with WhatsApp Through Twilio API

Communication with the WhatsApp Business (WB) service necessitates the use of a proprietary API, access to which requires prior authorization and a rigorous identification process, posing a significant barrier for developers. To avoid this, WB recommends using official enterprise solution providers. Consequently, Twilio (see Section 2), which offers a REST API to facilitate communication between applications and the messaging platform, was selected for this project. Following the registration of the phone number, this integration enables user interaction with the bot via WhatsApp.

Defining the Twilio Platform in Xatkit to be Integrated with the Instant Messaging Platform WhatsApp

To define the Xatkit platform, a class designated TwilioPlatform was constructed in the proposed metamodel (Figure 3). The class enables the initialization of the platform, the storage of credentials, and the invocation of actions that the bot is capable of performing. The initialization of the Twilio client, together with the twilioEventProvider, is the first step. The latter is responsible for receiving the call from the webhook and converting it into an event. Subsequently, the Twilio server is appropriately informed, thereby indicating the arrival of a new message. The twilioEventProvider class is derived from the abstract WebhookEventProvider class, which receives the webhook call and handles the requests made by the Xatkit server. In this process, an instance of TwilioRestHandler is created. In this final class, the methods are those that convert the body of the request from the application/x-www-form-urlencoded format to the JSON (JavaScript Object Notation) format. Furthermore, the content of the message is extracted and processed as an intent, enabling the bot to recognize the intent conveyed in the text when a user sends a message. Similarly, the telephone numbers involved will be extracted. Ultimately, a session will be established based on the telephone number of the sender of the message for the chatbot

to record the conversational status of each user. Each time a message is transmitted to the telephone number with the active session in the Twilio account, the bot utilizing the Twilio platform will perform this process internally and send an event so that the bot can respond with one of the defined actions.

Once the platform has been defined in Xatkit, the next step is for the developer to create a new instance of the TwilioPlatform class and the TwilioEventProvider class, and to add their credentials to the configuration. Subsequently, the developer must indicate that the Twilio platform is to be used, enabling the chatbot to receive, process, and send WhatsApp messages. The actions defined for the platform are capable of sending messages with text or multimedia content, and these may be utilized as required.

5. Application Example

This section elaborates on the design and implementation of a chatbot as a practical proof of concept to validate the proposed MDD approach to generate conversational agents, as presented in this research. The objective is to demonstrate the feasibility and advantages of leveraging MDD principles to streamline chatbot development, integrating advanced functionalities such as multi-platform deployment and intelligent response generation. The proof of concept was realized using the Xatkit Chat Platform, a framework that enables the semi-automated creation of conversational agents through MDD processes. The implementation supports chatbot deployment on two distinct platforms: WhatsApp, enabled through integration with Twilio, facilitating deployment on a globally dominant instant messaging platform; and React, providing compatibility for web-based conversational interfaces, thus broadening the scope of potential applications.

To validate the proposed approach, a chatbot was developed to serve as an assistant for users seeking information about the Postgraduate Program in Information Sciences at the Universidad Autónoma de Sinaloa. This case study was chosen to highlight the chatbot's capability to handle specific, domain-focused inquiries while demonstrating scalability for broader applications.

5.1. Context

The Universidad Autónoma de Sinaloa (UAS) offers advanced academic training in the field of Information Sciences through its Postgraduate Program in Information Sciences (PIS). This program encompasses Master's and Doctorate degrees, with two distinct areas of specialization, Computing and Systems and Geomatics and Geodesy. The program holds official recognition within the National Postgraduate System of the National Secretary of Science, Humanities, Technology and Innovation (SECIHTI) in Mexico, highlighting its academic rigor and alignment with national standards for graduate education.

Although the Postgraduate Program in Information Sciences (PIS) conducts annual admissions with promotional activities, interest in the program extends beyond these periods, necessitating continuous access to information. The current dissemination of information through Facebook and the program website often proves insufficient to address specific user queries and navigation challenges. To address this, a 24/7 accessible conversational agent is proposed to enhance the user experience and support prospective students throughout the year.

In response to the identified challenges, this case study focuses on the design, development, implementation, and evaluation of a conversational agent tailored to the needs of the PIS. This research encompasses three key objectives: (1) developing and deploying a WhatsApp-based chatbot leveraging the Xatkit framework and the proposed MDD methodology to ensure efficient creation and adaptability; (2) conducting a comprehensive user satisfaction survey to evaluate the chatbot's effectiveness, focusing on user preferences

and experiences within the WhatsApp instant messaging context, specifically assessing its ability to provide accurate and relevant information; and (3) evaluating user satisfaction with the chatbot's GPT-3-generated suggestions in scenarios where predefined knowledge is insufficient, analyzing its capacity to provide contextually appropriate responses and maintain conversational flow.

The findings are expected to provide information on the feasibility of deploying conversational agents on widely used platforms such as WhatsApp, the effectiveness of combining structured MDD methodologies with AI-driven functionalities to improve chatbot performance, and the overall impact on user satisfaction and institutional outreach. This research underscores the transformative potential of conversational agents in improving information accessibility, particularly where traditional communication channels fall short.

5.2. Participants

A case study involving 100 prospective students interested in the Postgraduate Program in Information Sciences (PIS) was conducted to evaluate the postgraduate chatbot. The participants, selected based on their alignment with the PIS entry requirements, represented the main target audience of the program. Data collection involved a user satisfaction survey and systematic documentation of all user–chatbot interactions. To minimize bias, no information about the PIS program or chatbot functionality was provided beforehand; Participants received only the chatbot's WhatsApp contact information and a survey link devoid of contextual details.

5.3. Conversational Agent Design

This section presents the design of the components that comprise the conversational agent that was implemented in the present case study. In order to facilitate a strategic approach to design, four categories were created with the objective of establishing a hierarchy of components and planning the design. The following categories were defined specifically to validate the technologies and methods proposed in this article: Conceptual Elements, Functional Elements, Technological Elements, and Prompt Definition.

5.4. Conceptual Elements

The conceptual elements align with the definition of the characteristics of the conversational agent, including its implementation objective, the profile of users with whom it will interact, the manner in which it will communicate with the user (through text and images), and the user's intention or desire when engaging with the bot. In light of the aforementioned considerations, subsequent categories of elements are developed in order to define the essence of the chatbot.

5.4.1. Goal

The goal of the conversational agent is to provide users with prompt and accurate information about the postgraduate course in information science, with a particular focus on prospective students. In particular, the objective is to respond in a reliable and efficient manner to queries posed by the user.

5.4.2. Users

The chatbot is primarily utilized by individuals who have completed or are nearing the completion of their undergraduate studies and are interested in pursuing a master's or doctoral program at the PIS program. The typical user is an adult aged 21 years or older. Given the characteristics of the user, it can be assumed that most of them will be familiar with the use of software. However, the agent is capable of being beneficial for

any individual with the ability to read and write in Spanish or English, regardless of their experience with technology.

5.4.3. Chatbot Identity

The automated system is designed to provide assistance in a formal manner and is capable of responding to the majority of queries posed to it in a manner that is accurate and comprehensive. The system does not utilize an avatar; instead, it displays an image of the PIS logo. The agent is bound by protocol to respond to the user's queries, and in the event that a resolution cannot be provided, an apology must be offered. Furthermore, if possible, it provides an appropriate alternative response.

5.4.4. Communication

The agent is able to express himself through text and images, as well as emojis, which are digital images or icons used in electronic communications to represent an emotion, an object, an idea, etc. The sole language of communication in this case is English. The question is not expected to be received in a language other than English during the process of defining intents, training phrases, and answers. The user is allowed to communicate exclusively with the agent exclusively through text.

5.5. Functional Elements

The functional elements are the activities that the chatbot must be able to perform in each scenario to achieve its main objective. A set of expressions, defined as intents, is utilized to define the scenarios in which a chatbot is required to interact with a user. These expressions are designed to reflect the user's purpose or desire and are conveyed to the user by the conversational agent. In the context of this case study, the bot is not required to perform any automated tasks beyond the resolution of the queries to which it is asked to address.

Currently, most people who have expressed interest in joining the PIS have obtained information through the official website. According to the stipulations set forth by the National Postgraduate System of the Government of Mexico, postgraduate programs are required to provide a website that meets a series of quality standards. The PIS website is in compliance with these requirements and offers up-to-date and reliable information about the program. Consequently, the majority of the intents defined for the conversational agent in this case study were devised with the objective of utilizing these data to provide accurate responses to potential inquiries from prospective post-graduate program applicants.

The intents delineated in this case study are classified into two categories: those related to the bot itself and those pertaining to the dissemination of information about the post-graduate course. In total, 36 intentions were designed and the resulting actions were divided into two categories: text responses and multimedia responses. Furthermore, a number of intents have multiple potential responses, which are selected based on entities extracted from the user's query. As previously stated, the responses to the queries are derived from the official PIS website and from the representatives (administrative staff) of the institution. Subsequently, 30 of the 36 intents pertaining to the post-graduate degree are presented. The final category of intents, which pertains to the functionality of the chatbot, is outlined in the following section.

1. Information about the postgraduate course.
2. Purpose of the postgraduate course.
3. Graduate profile.
4. Duration of the postgraduate course.
5. Requirements.

6. Pre-registration process.
7. Receipt of documents.
8. Registration process.
9. Scholarship process and costs.
10. Existence of scholarships.
11. Admission results.
12. Interview information.
13. Postgraduate mode.
14. Modality of exams.
15. Date of pre-registration.
16. Date of receipt of documents.
17. Date of the preliminary course and knowledge test.
18. Date of psychometric examination.
19. Date of CENEVAL exam.
20. Start of activities.
21. Frequency of convocation.
22. Mobility information.
23. Research lines.
24. Syllabus.
25. Teaching staff information.
26. Test information.
27. Contact details.
28. Address.
29. Test guidelines.
30. Qualification information.

Intents created to provide functionality to the conversational agent are listed as follows:

1. Greeting.
2. Bot information.
3. General information.
4. Offer help.
5. Negative response.
6. Information on the development of the bot.

5.6. Technological Elements

This section provides an overview of the technological ecosystem used to define, develop, and implement the conversational agent in this case study. The operating system, platforms, programming languages, frameworks, and other pertinent details are provided.

- Operating system: Ubuntu 20.04 (LTS) 64 bits.
- Programming language: Java.
- Framework: Xatkit.
- Natural language processing service: Dialogflow/NLP.JS.
- Middleware: Twilio API.
- Instant messaging service: WhatsApp.
- User query alternative: GPT-3 model text-curie-001, TheoKanning/openaijava.
- Computing equipment: 4 GB RAM, 2 AMD vCPUs 1996 MHz shared, 80 GB NVMe SSD storage.

In order to facilitate the implementation process, the Ubuntu 20.04 (LTS) x64 operating system was selected as it aligned with the case study and allowed for a realistic replication of a typical development environment. Frequently, Ubuntu is utilized as a web server

for such projects. Furthermore, the Xatkit framework and the implementation of the Xatkit WhatsApp via the Twilio platform were employed to develop the conversational agent. The primary rationale for this selection was that this version is the one required by TheoKanning/openai-java to facilitate communication with the OpenAI API and thereby utilize GPT-3.

One of the most significant advantages of Xatkit is its ability to integrate and modify the NLP provider at any stage of the development process, due to its utilization of MDD techniques. Consequently, it is not necessary to designate a specific one in the design stage. This has the advantage of allowing experimentation once the agent is developed, thus enabling the selection of the option that best suits the developer's needs or, alternatively, the option that demonstrates superior performance for the specific use case. In accordance with the aforementioned point, at this juncture of the case study, the natural language provider remains unspecified. Subsequently, one of the two available providers in Xatkit had to be selected. The two principal options were Dialogflow and NLP.JS. The regular expression parser was not considered suitable for inclusion in this study, as its primary function is for testing purposes. Similarly, the Xatkit NLU Engine was not selected for use, given that it is still in an experimental phase.

5.7. Prompt Definition

In the context of artificial intelligence and conversational agents, a prompt refers to a phrase or question that is presented to the system to generate a relevant response. The prompt developed for this case study, shown in Table 2, was designed using the guidelines described in Section 4.2 of this article.

In a conversational agent, the prompt must contain four elements: a list of options, detailed instructions, a list of examples, and the user's question. In this case study, the list of options was derived from the chatbot's training sentences, with a single training sentence utilized for each intent. Inclusion of additional elements would be superfluous and result in an increase in the number of tokens (a form of text division used by GPT-3; the number varies depending on the word encountered) used due to the expansion of the text, which would consequently lead to an increase in cost and processing time. The instructions and examples provided in the prompt design process for this case study are sufficient to ensure that the desired format of the generated text is clear and accurate. Consequently, when designing a prompt in accordance with the guidelines delineated in the Suggestion Generation with GPT-3 section, it is advised to utilize a single option for each intent, thereby facilitating enhanced chatbot comprehension. In addition, a text has been provided that delineates the guidelines that GPT-3 should consider when generating the text. The list of examples comprises a series of hypothetical queries that a user might pose and the corresponding appropriate responses.

The resulting prompt is shown in Table 2. Each string of text corresponding to a question in the prompt is appended with the user's message where appropriate, and the string 'Suggestion:'. In order to maintain the desired format, we experimented with slight modifications such as removing punctuation and instructions. This modification caused only minor changes in text generation, mainly by reducing the number of tokens. In total, an estimated 1376 tokens were obtained. This fact led to the decision to use a GPT-3 engine called Curie, as it is faster and has a lower consumption cost compared to the more powerful Davinci engine. As it did not represent a significant improvement, we decided to keep it for the experiment without significant modifications. The example is detailed in Table 3.

The initial phase of the case study involved the use of the XatbotMM metamodel (Figure 1) to delineate the conceptual framework of the conversational agent designed for

the PIS. This was accomplished through the use of the Eclipse Modeling Framework, employing the Ecore metamodel. This resulted in the generation of a model that corresponds to an instantiation of the Ecore metaclasses. This model delineated the states, intents, transitions, and actions that the bot would utilize. Furthermore, the model properties stipulate the requisite platform and NLP provider, which can be either TwilioPlatform or DialogFlow. For this particular bot, TwilioPlatform was selected. The resulting model is illustrated in Figure 7. Following the creation of the PCIBot agent model, the Acceleo M2T template was used to generate most of the source code for the Xatkit Chat Platform framework. The M2T transformation template facilitated the semi-automated construction of the conversational agent, obviating the necessity to code it from scratch, thus reducing the time required for development. Note that the chatbot generated from the model can be implemented in a real environment. However, due to the inclusion of several additional features and functionalities, such as the use of entities for personalized responses and the implementation of the GPT-3 service to generate suggestions, minimal manual intervention in the source code was required.

Table 2. List of options for the prompt.

Suggested Text String at the Prompt
Do you have a guide to the exam?
Where are they located?
Contact details
Information about the teachers
What is the curriculum of the Master's program?
What are the lines of research?
Information about mobility
Who runs your program?
Information about the frequency of calls
When is the starting date of activities?
When is the date of the Propaedeutic Course and Knowledge Examination?
Information about the date of reception of documents
Information about pre-registration dates
Information about the mode of the exams
Information about the modality of the postgraduate exams
Information about the interview
When do I get my admission results?
Information about scholarships
What is the graduation process?
What do I have to do to enter the postgraduate program?
Information about the pre-registration process
What are the requirements to enter the Master's program?
Information about the duration of the postgraduate program
Information about the postgraduate program
Information about costs
How did they make you?

The next step in the conversational agent development process for this case study was to install the Xatkit Chat Platform framework. To achieve this, we consulted its official documentation [20], where we found the installation of the Java Development Kit (JDK), with Maven (software project management framework, a standard project management and description model) and Git (open-source source code version control software) as prerequisites.

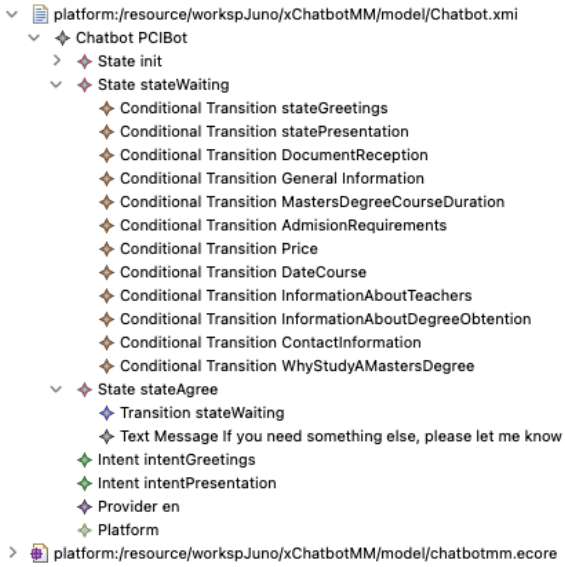


Figure 7. Part of the PCIBot model.

Once the requisite configuration of the development environment had been completed, the next step was to create a Maven project. This was based on the Xatkit Chat Platform sample project. Subsequently, all the intents described in Table 2 were implemented using the Xatkit Domain-Specific Language. This was achieved by creating a variable for each intent, utilizing the intent() function, and the definition of a set of training sentences. Each of these represents a potential reference by the user to a specific intent (question), which the bot is then able to classify as belonging to one of the available intents. Figure 8 illustrates the training sentences for the conversational agent developed in this case study.

```
begin {1stling}[language=Java , frame=none]
    val grettings = intent("greetings")
    trainingSentence("Hi")
    trainingSentence("Good Day")
    trainingSentence("Good Moorning")
    trainingSentence("Whats Up?")
    trainingSentence("Hello")
```

Figure 8. Part of definition of training phrases for intent greeting.

To integrate the ability to respond to user messages not found in the intents of the conversational agent, deep learning was implemented by implementing the GPT-3 autoregressive language model. OpenAI's GPT-3 library theokanning.openai-gpt3-java API allows the integration and automatic generation of text to respond to the bot user. The libraries enabling communication with the natural language processing providers Dialogflow and NLP.js were then imported. Finally, the ability to run the agent was integrated into WhatsApp using Twilio.

Entities were also created to increase the responsiveness of the conversational agent, giving it the ability to know whether the user's message was requesting information about a particular course. This made it possible to personalize the PCI bot's response and not overwhelm the user with information that was not requested. The entities allow information to be extracted from the user's message. The Twilio platform and TwilioEventProvider were then installed so that the conversational agent could receive input (messages) from the user and perform the actions of sending a text or multimedia message in response. It should be noted that the source code for defining intents, states, transitions and actions was automatically generated from the model shown in Figure 4.

Table 3. Example of suggestions.

Example of Suggestions
Ask the user for an option that may interest them from the list of options based on their message in the conversation, in the same way as in the example. You can only respond with examples from the list of options and must make a suggestion related to the user's message. If you cannot offer a suitable or useful recommendation from the list of options, respond with "Sorry, I don't have a suggestion to show".
Examples:
User: Does Professor Carlos teach classes? Suggestion: If you're interested, you could try writing: Information about the teachers User: Time Suggestion: If you're interested, you could try writing: What is the starting date of activities? User: But I don't have money Suggestion: If you're interested, you could try writing: Information about scholarships User: I don't understand what I have to do Suggestion: If you're interested, you could try writing: What do I have to do to enter the postgraduate program? User: Is it too far from me? Suggestion: If you're interested, you could try writing: Where are they located? User: Is it difficult to speak Japanese? Suggestion: Sorry, I don't have a suggestion to show. User: Is it expensive? Suggestion: If you're interested, you could try writing: Information about costs User: Information about dogs Suggestion: Sorry, I don't have a suggestion to show. User: Is it too far from my house? Suggestion: If you're interested, you could try writing: Where are they located? User: I want to get a transcript Suggestion: If you're interested, you could try writing: Contact details User: What are the requirements to enter the master's degree? Suggestion: If you're interested, you could try writing: What are the requirements to enter the Master's program? User: I love you Suggestion: Sorry, I don't have a suggestion to show. User: I'm bored Suggestion: If you're interested, you could try writing: Information about mobility User: I don't know what to do Suggestion: If you're interested, you could try writing: General information User: What if I can't take the exam? Suggestion: If you're interested, you could try writing: Information about the mode of the exams User: I want to listen to music Suggestion: Sorry, I don't have a suggestion to show.

In order to continue with the implementation of the conversational agent, designated PCIBot, it was deemed necessary to define the states that the aforementioned bot can be in, as well as the transitions that are required to move from one state to another. This is due to the fact that chatbots in the Xatkit Chat Platform operate as state machines, enabling them to respond to any kind of event, such as the detection of an intent from the user's message.

Each of the steps explained above allowed us to define the responses for each defined query that could be sent to the PCIBOt. A total of 42 states were programmed with their actions and transitions.

In the particular case where the conversational agent's NLP provider has no response registered in its intents, there is a special state called defaultFallback. Its purpose is to allow

the developer to program a pre-defined response to indicate to the user that the request could not be resolved but will provide an appropriate recommendation. For this case study, the implementation of this functionality was as follows: a string variable or text string called promptVar was defined to initialize the OpenAi service. Once this is accomplished, an array of response options is defined. The defaultFallback will then retrieve the last message from the user using the getMatchedInput() function. The message is appended to the end of promptVar and the string 'Suggestion:' is concatenated to continue with the desired format as described in the prompt instructions and examples.

The subsequent phase involved utilizing the service to generate text that would be sent back to the user as a response, which would then be documented in a file. It is necessary to filter the text in advance as there are occasions when the model will be unable to generate a suitable recommendation. In this case study of PCIBot implementation, text strings were filtered to a maximum length of 25 characters.

The final step in the process was the implementation of the capabilities of the conversational agent, which allowed the agent's behavior to be fully defined. Subsequently, the final configuration adjustments were made. These included the delineation of the platforms to be utilized by the bot, the specification of the natural language processing service responsible for interpreting the user's intentions, and the authentication credentials for the services to be employed.

5.8. Implementing the Chatbot

Upon completion of the coding for the conversational agent, the implementation phase was started. The preliminary phase of the process involved the installation of the Xatkit chat platform on a Linux Ubuntu operating system. Subsequently, as determined during the design phase, a WhatsApp Business account was established through the use of the DigitalOcean hosting service. Ultimately, a telephone number was assigned to the Twilio service, thus facilitating communication between users and the chatbot. The model depicted in Figure 7 was subsequently transformed into source code using the M2T (Model-to-Text) transformation, following the process detailed in Figure 2. An excerpt of the resulting Java bot source code is presented in Figure 9.

```
/*
 * Define the intents the bot will react to.
 * Note that we recommend the usage of Lombok's val when using the
 * Xatkit DSL: the fluent API defines many interfaces that are not
 * useful for bot designers. If you don't want to use val you can use
 * our own interface IntentVar instead.
 */
val greeting = intent("greeting")
        .trainingSentence("Hello")
        .trainingSentence("Good morning")
        .trainingSentence("Good afternoon")
        .trainingSentence("What's up")
        .trainingSentence("How's it going");

val whoAreYou = intent("whoAreYou")
        .trainingSentence("Who are you?")
        .trainingSentence("Who is it?")
        .trainingSentence("Tell me about yourself")
        .trainingSentence("Are you?");
```

Figure 9. Snippet of chatbot source code generated.

5.9. Evaluating the Chatbot

This section describes the evaluation of the PCIBot conversational agent, which mainly consisted of analyzing the quality of the conversational agent's answers and the GPT-3 recommendations. In addition, we investigated how users interacted with the bot when it was unable to solve their query.

In order to achieve the above objectives, the evaluation first focused on collecting data from users through a survey. The conversational agent, PCIBot, was created to provide information from the Information Sciences Graduate Program of the Universidad Autónoma de Sinaloa. It was implemented on the world's most widely used instant messaging platform, WhatsApp. To initiate the case study, the PCIBot WhatsApp phone number was shared with 100 users who belong to or graduated from four universities: Universidad Autónoma de Sinaloa (UAS), Universidad Politécnica de Sinaloa (UPSIN), Universidad Autónoma de Occidente (UAdeO), and Instituto Tecnológico de Mazatlán (ITMAZ). These institutions have in common the study of computer science and computer systems at the undergraduate level. Figure 10 shows the operation of the chatbot interacting with a user through natural language in the WhatsApp application installed on their mobile phone. The PCIBot is able to respond to the query through text and images. Then a satisfaction survey was shared with the conversational agent users. The survey was distributed through the Google Forms tool [41]; once conducted, by analyzing the questions related to the conversations between the user and the bot, the objectives of this case study were evaluated.

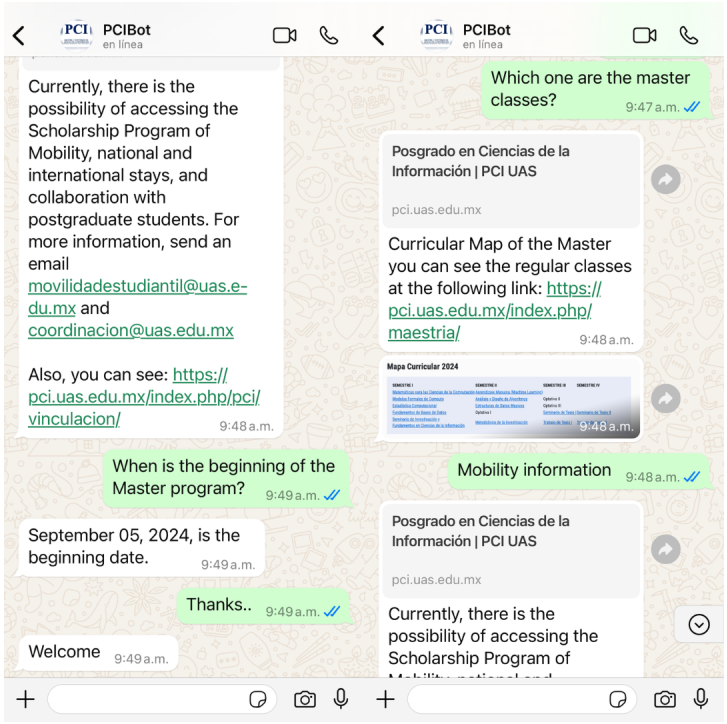


Figure 10. PCIBot in action.

Analysis

The survey data and the associated questionnaire are available in FigShare via the link provided in [42]. The analysis of the results of the survey administered to PCIBot users is presented below.

Regarding overall user satisfaction, while PCIBot received generally positive feedback (66% satisfaction rate), instances of user dissatisfaction were observed. Analysis of recorded user–PCIBot interactions revealed that dissatisfied users commonly sought functionalities not yet implemented within the agent. This led to requests beyond the bot's current capabilities, underscoring the necessity of a post-deployment maintenance phase for chatbot development, enabling iterative updates based on user needs.

Regarding perceived ease of use, a significant majority (71%) of users reported ease in formulating requests and expressed overall satisfaction with the usability and utility

of the agent. Analysis of interaction logs indicated that users frequently resolved their queries using recommendations provided by the bot. However, instances where the agent struggled to interpret user intent were also identified, highlighting a key area for future development. Improved training sequences are proposed to mitigate these occurrences and enhance the effectiveness of interaction.

The integration of PCIBot within the WhatsApp platform was met with strong user acceptance (99%). User feedback emphasized the convenience and accessibility of 24/7 availability from their mobile devices within a frequently used application. Participants further noted the utility of a conversational agent for accessing PIS information, learning more about the graduate program's field of study, and gaining rapid access to information outside of regular business hours. In particular, a substantial majority (96%) preferred WhatsApp over other instant messaging platforms such as Facebook, demonstrating the value of tools that facilitate WhatsApp integration, such as the one presented in this investigation.

Assessment of the efficacy of the GPT-3 autoregressive language model recommendation revealed general user satisfaction. However, a discrepancy emerged between reported satisfaction and actual feature usage, indicating that the DialogFlow natural language processing service was not required, as the bot successfully classified all queries. Analysis of user–PCIBot dialogues revealed that users used the recommendation feature to rephrase queries or obtain additional information. Fifty-five percent of the respondents found the recommendations useful, while 36% did not utilize the feature because they received direct responses to their queries. Only 8% reported unhelpful recommendations. Although recommendations proved beneficial when the agent was unable to directly answer a question, users primarily used this functionality to refine their queries for improved bot comprehension or explore related information. Both survey responses and data analysis revealed instances of unsatisfactory recommendations, particularly when requests deviated significantly from the academic context of the program, leading to confusing or irrelevant suggestions. While the GPT-3 recommendation system generally responded appropriately with "Sorry, I don't have any suggestions to show" for out-of-scope requests, some instances of contextually irrelevant suggestions were observed. Despite these limitations, 95% of the users rated the quality of PCIBot's recommendations as useful. User feedback highlighted the value of the suggestion function for clarifying doubts and exploring related information, the utility of guidance on query reformulation, and the benefit of continued conversational flow following a suggestion, contrasting with other agents that abruptly terminate interactions.

6. Conclusions

To address the limitations of existing chatbot development frameworks, this research proposes an extension to the Xatkit framework, enabling seamless integration with WhatsApp via the Twilio communication platform. This proposal not only extends the capabilities of Xatkit, but also positions it as a versatile tool for creating conversational agents that can be deployed on WhatsApp, one of the most popular messaging platforms in the world. This integration addresses a critical gap, significantly broadening the applicability of the framework and establishing it as a resource for developers seeking to deliver sophisticated and accessible chatbot solutions. The increasing prevalence of conversational agents reflects the user's preference for text- and voice-based interaction over traditional phone calls. As demonstrated in [6], research in this area has intensified, underscoring its current relevance within the scientific community. This research has largely focused on developing new technological proposals for creating more comprehensive tools that facilitate and accelerate conversational agent development, reducing costs and time for developers and companies. The proliferation of voice-activated smart devices, such as Amazon Alexa [43],

Google Assistant [44], and Apple Siri [45,46], has further driven the demand for 24/7 service availability via conversational agents, offering companies a cost-effective means of automating interactions related to product information, pricing, and availability.

While the demand for chatbots is increasing, so is the need for specialized technical expertise in their development. Although several proposals offer specialized techniques, libraries, and frameworks to facilitate conversational agent development (as detailed in Section 3), these often neglect widely used messaging platforms such as WhatsApp and Telegram, limiting their applicability and potentially introducing programming complexities due to required intermediary layers. Furthermore, the potential of autoregressive language models, such as GPT-3, remains underexplored in this context [47]. This research addresses these limitations by proposing an improved approach for generating conversational agents within the Xatkit Chat Platform framework for WhatsApp integration. Key contributions include the following: (1) establishing connectivity between the Xatkit platform and WhatsApp via the Twilio cloud communication platform, enabling personalized agent interaction; (2) defining a metamodel (XatkitMM) using Ecore, specifying the core components of a Xatkit chatbot and enabling model-based agent design, obviating the need for direct source code manipulation; (3) implementing a Model-to-Text (M2T) transformation using Acceleo to automatically generate bot Java source code from the defined model; (4) integrating the GPT-3 autoregressive language model via API calls to provide user suggestions and address queries beyond the agent's predefined intents; and (5) conducting a case study analysis demonstrating the necessity of implementing advanced query resolution techniques to ensure satisfactory user experiences.

Future Work

Future research will focus on several key areas to enhance conversational agent development within the Xatkit platform. These include the following: (1) developing a direct integration with the WhatsApp Business API to eliminate intermediaries like Twilio, reducing costs and streamlining development for multiple platforms (e.g., Telegram); (2) enhancing automation of model-based chatbot generation to minimize manual coding, addressing the current lack of comprehensive MDD frameworks; (3) integrating prompt engineering and design techniques, including modifiers for generative models, particularly in text-to-image systems, and developing a WYSIWYG interface using tools like Eclipse Sirius or MetaEdit+ to democratize chatbot model creation; (4) exploring domain-specific training, language adaptation, and alternative autoregressive models (e.g., GPT-Neo, BERTIN-GPT-J-6B) for dynamic response generation; and (5) investigating fine-tuning techniques to optimize pretrained models for specific tasks, aiming to improve performance, reduce token usage, and lower operational costs.

Author Contributions: Formal analysis, Á.M.-G.; investigation, Á.M.-G. and J.A.A.-C.; methodology, C.T.-B.; software, Á.M.-G. and J.A.A.-C.; validation, J.A.A.-C.; writing—original draft, C.T.-B.; writing—review and editing, A.Z.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are openly available in FigShare at https://doi.org/10.6084/m9.figshare.27287673.v1.

Acknowledgments: The authors appreciate the assistance provided by members of the Research Group Tecnología Educativa I+D+i (UAS-CA-303) from Universidad Autónoma de Sinaloa (UAS), México.

Conflicts of Interest: Author Ángel Martínez-Gárate was employed by the company Tata Consultancy. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Rarhi, K.; Bhattacharya, A.; Mishra, A.; Mandal, K. Automated medical chatbot. *SSRN Electron. J.* **2017**, *1*, 1–15. [CrossRef]
2. Ranoliya, B.R.; Raghuwanshi, N.; Singh, S. Chatbot for university related faqs. In Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 1525–1530. [CrossRef]
3. Perez-Soler, S.; Juarez-Puerta, S.; Guerra, E.; de Lara, J. Choosing a chatbot development tool. *IEEE Softw.* **2021**, *38*, 94–103. [CrossRef]
4. Abdellatif, A.; Costa, D.; Badran, K.; Abdalkareem, R.; Shihab, E. Challenges in Chatbot Development: A Study of Stack Overflow Posts. In Proceedings of the 17th International Conference on Mining Software Repositories, Seoul Republic of Korea, 25–26 May 2020; pp. 174–185.
5. Aguilar, J.A.; Garrigós, I.; Mazón, J.-N.; Trujillo, J. An MDA Approach for Goal-oriented Requirement Analysis in Web Engineering. *J. Univers. Comput. Sci.* **2010**, *16*, 2475–2494. [CrossRef]
6. Martinez-Garate, A.; Aguilar-Calderon, J.A.; Tripp-Barba, C.; Zaldivar-Colado, A. Model-driven approaches for conversational agents development: A systematic mapping study. *IEEE Access* **2023**, *11*, 73088–73103. [CrossRef]
7. Daniel, G.; Cabot, J.; Deruelle, L.; Derras, M. Xatkit: A multi-modal low-code chatbot development framework. *IEEE Access* **2020**, *8*, 15332–15346. [CrossRef]
8. Statista. *WhatsApp: Number of Users 2013–2020*; Statista: Hamburg, Germany, 2024. Available online: https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/ (accessed on 23 November 2024).
9. Eclipse Foundation. Eclipse Modeling Project. Available online: https://projects.eclipse.org/projects/modeling (accessed on 10 January 2025).
10. Eclipse Foundation. Acceleo. Available online: https://projects.eclipse.org/projects/modeling.acceleo (accessed on 10 January 2025).
11. Brambilla, M.; Cabot, J.; Wimmer, W. *Model-Driven Software Engineering in Practice*, 2nd ed.; Morgan & Claypool Publishers: San Rafael, CA, USA, 2017.
12. Panach, J.I.; Pastor, O.; Juristo, N. A Family of Experiments to Compare Two Model-Driven Development Tools vs. a Traditional Development Method. *IEEE Trans. Softw. Eng.* **2022**, *48*, 4802–4817. [CrossRef]
13. Zafar, A.; Azam, F.; Latif, A.; Anwar, M.W.; Safdar, A. Exploring the Effectiveness and Trends of Domain-Specific Model Driven Engineering: A Systematic Literature Review (SLR). *IEEE Access* **2024**, *1*, 99–120. [CrossRef]
14. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
15. WhatsApp Inc. WhatsApp Messenger. Available online: https://business.whatsapp.com/developers/developer-hub?lang=es_LA (accessed on 10 January 2025).
16. Twilio. Twilio | Communication APIs for Businesses. Available online: https://www.twilio.com/docs/iam/organizations (accessed on 10 January 2025).
17. Raibulet, C.; Fontana, F.A.; Zanoni, M. Model-driven reverse engineering approaches: A systematic literature review. *IEEE Access* **2017**, *5*, 14516–14542. [CrossRef]
18. Aguilar-Calderón, J.-A.; Tripp-Barba, C.; Zaldívar-Colado, A.; Aguilar-Calderón, P.A. Requirements Engineering for Internet of Things (loT) Software Systems Development: A Systematic Mapping Study. *Appl. Sci.* **2022**, *12*, 7582. [CrossRef]
19. Object Management Group Official Site. 2024. Available online: https://omg.org (accessed on 10 october 2024).
20. Xatkit Wiki. Available online: https://xatkit.com/ (accessed on 10 June 2024).
21. Daniel, G.; Cabot, J.; Deruelle, L.; Derras, M. Multi-platform Chatbot Modeling and Deployment with the Jarvis Framework. In *Advanced Information Systems Engineering*; Giorgini, P., Weber, B., Eds.; Springer: Cham, Switzerland, 2019; Volume 11483, pp. 177–193.
22. Pérez-Soler, S.; Guerra, E.; de Lara, J. Creating and Migrating Chatbots with Conga. In Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Madrid, Spain, 25–28 May 2021; pp. 37–40. [CrossRef]
23. Parada, W.V.; Giraldo, F.A.; Londoño, M.I. MDD based case tool for Automatic Generation of ChatBot. In Proceedings of the XLV Latin American Computing Conference (CLEI), Panama, Panama, 30 September–4 October 2019; p. 111737. [CrossRef]
24. Baena-Perez, R.; Ruiz-Rube, I.; Dodero, J.M.; Bolivar, M.A. A framework to create conversational agents for the development of video games by end-users. In *Optimization and Learning*; Dorronsoro, B., Ruiz, P., de la Torre, J.C., Urda, D., Talbi, E.-G., Eds.; Springer: Cham, Switzerland, 2020; Volume 1173, pp. 216–226.

25. Ed-douibi, H.; Izquierdo, J.L.C.; Daniel, G.; Cabot, J. A model-based chatbot generation approach to converse with open data sources. In *Web Engineering*; Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I., Eds.; Springer: Cham, Switzerland, 2021; Volume 12706, pp. 440–455.
26. Hogan, K.; Baer, A.; Purtilo, J. Diplomat: A conversational agent framework for goal-oriented group discussion. In *Contemporary Issues in Group Decision and Negotiation*; Morais, D.C., Fang, L., Horita, M., Eds.; Springer International Publishing: Cham, Switzerland, 2021; Volume 420, pp. 143–154.
27. Planas, E.; Daniel, G.; Brambilla, M.; Cabot, J. Towards a model-driven approach for multiexperience AI-based user interfaces. *Softw. Syst. Model.* **2021**, *20*, 997–1009. [CrossRef]
28. Pérez-Soler, S.; Daniel, G.; Cabot, J.; Guerra, E.; de Lara, J. Towards automating the synthesis of chatbots for conversational model query. In *Enterprise, Business-Process and Information Systems Modeling*; Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J., Eds.; Springer International Publishing: Cham, Switzerland, 2020; Volume 387, pp. 257–265.
29. Matic, R.; Kabiljo, M.; Zivkovic, M.; Cabarkapa, M. Extensible chatbot architecture using metamodels of natural language understanding. *Electronics* **2021**, *10*, 2300. [CrossRef]
30. Pérez-Soler, S.; González-Jiménez, M.; Guerra, E.; Lara, J. Towards Conversational Syntax for Domain-Specific Languages using Chatbots. *J. Object Technol.* **2019**, *18*, 5. [CrossRef]
31. Neumann, A.T.; de Lange, P.; Klamma, R. Collaborative Creation and Training of Social Bots in Learning Communities. In Proceedings of the IEEE 5th International Conference on Collaboration and Internet Computing (CIC), Los Angeles, CA, USA, 12–14 December 2020; pp. 11–19. [CrossRef]
32. Sansonnet, J.-P.; Leray, D.; Martin, J.-C. Architecture of a framework for generic assisting conversational agents. In *Intelligent Virtual Agents*; Gratch, J., Young, M., Aylett, R., Ballin, D., Olivier, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4133, pp. 145–156.
33. Khalil, A.; Leiva, F.H.; Shonibare, A.; Arsenault, E.M.; Turner, L.; khalifa, S.; Tam-Seto, L.; Linden, B.; Wood, V.; Stuart, H.; et al. Model-Driven Chats: Enabling Chatbot Development for Non-technical Domain Experts Through Chat Flow Visualization and Auto-generation. In *Advances in Information and Communication*; Arai, K., Ed.; Springer: Cham, Switzerland, 2021; Volume 1363, pp. 1036–1050.
34. Scotti, V.; Tedesco, R.; Sbattella, L. A modular data-driven architecture for empathetic conversational agents. In Proceedings of the IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju Island, Republic of Korea, 17–20 January 2021; pp. 365–368. [CrossRef]
35. Harry, L. Huey: Intelligent Agents for Natural Human to Machine Communication. Ph.D. Dissertation, Harvard University, Cambridge, MA, USA, 2021.
36. Mahmood, R.; Joshi, A.; Lele, A.; Pennington, J. Dynamic natural language user interfaces using microservices. In Proceedings of the CEUR Workshop Proceedings, Cagliari, Italy, 17 March 2020.
37. Gronback, R.C. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*; Pearson Education: London, UK, 2009.
38. Agrawal, P.; Menon, T.; Kamel, A.; Naim, M.; Chouragade, C.; Singh, G.; Kulkarni, R.; Suri, A.; Katakam, S.; Pratik, V.; et al. QnAMaker: Data to Bot in 2 Minutes. In Proceedings of the Web Conference 2020 (WWW '20), Taipei, Taiwan, 20–24 April 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 131–134. [CrossRef]
39. Dale, R. GPT-3: What's it good for? *Nat. Lang. Eng.* **2021**, *27*, 113–118. [CrossRef]
40. Kanning, T. OpenAI-Java Library. Available online: https://github.com/TheoKanning/openai-java (accessed on 22 December 2024).
41. Google Forms. Available online: https://forms.google.com.mx (accessed on 10 October 2024).
42. Satisfaction Survey Data for PCIBot Builded with Xatkit Framework. Available online: https://doi.org/10.6084/m9.figshare.27287673 (accessed on 23 October 2024).
43. Amazon Alexa Official Site: What Is Alexa? Available online: https:developer.amazon.com/es-ES/alexa (accessed on 10 November 2024).
44. Google Assitant. Available online: https://assistant.google.com (accessed on 10 November 2024).
45. Apple Siri. Available online: https://www.apple.com/mx/siri/ (accessed on 10 November 2024).
46. Lokman, A.; Ameedeen, M. Modern chatbot systems: A technical review. In Proceedings of the Future Technologies Conference (FTC), Vancouver, BC, Canada, 15–16 November 2018.
47. Min, B.; Ross, H.; Sulem, E.; Veyseh, A.P.B.; Nguyen, T.H.; Sainz, O.; Agirre, E.; Heinz, I.; Roth, D. Recent advances in natural language processing via large pre-trained language models: A survey. *arXiv* **2021**, arXiv:2111.01243. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.