A Framework to Create Conversational Agents for the Development of Video Games by End-Users

Rubén Baena-Perez(✉), Iván Ruiz-Rube, Juan Manuel Dodero, and Miguel Angel Bolivar

University of Cadiz, Cádiz, Spain
{ruben.baena,ivan.ruiz,juanma.dodero,mabolivar.perez}@uca.es

Abstract. Video game development is still a difficult task today, requiring strong programming skills and knowledge of multiple technologies. To tackle this problem, some visual tools such as Unity or Unreal have appeared. These tools are effective and easy to use, but they are not entirely aimed at end-users with little knowledge of software engineering. Currently, there is a resurgence in the use of chatbots thanks to the recent advances in fields such as artificial intelligence or language processing. However, there is no evidence about the use of conversational agents for developing video games with domain-specific languages (DSLs). This work states the following two hypotheses: (i) Conversational agents based on natural language can be used to work with DSL for the creation of video games; (ii) these conversational agents can be automatically created by extracting the concepts, properties and relationships from their abstract syntax. To demonstrate the hypotheses, we propose and detail the implementation of a framework to work with DSLs through a chatbot, its implementation details and a systematic method to automate its construction. This approach could be also suitable for other disciplines, in addition to video games development.

Keywords: Video games · End-User Development · Conversational agents · Model-Driven Engineering · Domain Specific Languages · Chatbots

1 Introduction

The video game industry has reached almost 138 million dollars in sales in 2018, becoming one of the most important and profitable sectors in the world, surpassing in growth the music and film industry [29]. This growth has been boosted since the appearance of smartphones, where video games are one of the most popular applications. Although video games have traditionally been seen as a leisure-focused activity, they are increasingly becoming a popular alternative in other fields such as academia [7,15], where teachers are limited by time and skills

B. Dorronsoro et al. (Eds.): OLA 2020, CCIS 1173, pp. 216–226, 2020.
https://doi.org/10.1007/978-3-030-41913-4_18

for its development and use in class, or in the field of health, where they can offer an attractive alternative to traditional modes of exercise [6].

Nowadays, the development of a videogame requires the participation of several professionals, each of them focused on one aspect of the videogame, forming a multidisciplinary team where there are not only programmers. This means that non-expert users are limited when it comes to creating their own video games. This circumstance also affects both the personal and professional context, increasing the need for users to create their own software artifacts in any of the fields. As a result, there is a growing interest in End-User Development (EUD) tools [18], thanks to which non-programmers can create software, participating in different stages of the creation process, without having to know professional programming languages [22]. Although these tools can help users, they still do not seem to adapt to complex software engineering problems.

End-user development tools are usually based on special languages adapted to the needs of the target users. Domain-specific languages (DSLs) are hence specialised in modeling or solving a specific set of problems. There are several cases of success in terms of end-user development tools using DSLs, as is the case of spreadsheets [10] or business process management [11], among others. Domain experts are increasingly participating in the design of these languages, contributing with their experience [20].

New devices such as mobile phones, tablets or smartwatches have made desktop computers no longer the only representatives of end-user computing [14]. The appearance of new methods of interaction, such as voice or gesture recognition, has led to a rethinking of the interaction between human and machine, with the main objective of achieving natural communication with the end-user [14]. In the field of end-user development, there is little work on these new paradigms of user interfaces, suggesting that even EUD researchers are not significantly contributing to the development of such tools [19].

Traditionally, textual or visual user interfaces are the most commonly used interfaces for working with DSL. Could new interaction methods be applied to DSL and its supporting tools? In order to develop the hypotheses previously raised, a framework has been developed in this research to automate the creation of conversational agents for DSLs. The framework allows users to interact with their DSL through a text-based or voice-based chatbot.

The rest of the paper is structured as follows. Section 2 presents the research background and related works. Section 3 describes the conversational modeling framework. A case study consisting on the use of a conversational agent for the Martin Fowler's State Machine language [13] is included in Sect. 4. Conclusions and future lines of work are stated in the last section.

2 Background and Related Works

The objective of this research is to demonstrate that it is possible to use conversational agents to work with DSLs under the Model-Driven Engineering (MDE) approach in video game development. The following is a description of the most

commonly used game engines, the concepts of DSL and MDE as well as their applications to video game development and, finally, the foundations and the role of conversational agents for modeling and programming purposes.

2.1 Game Development Tools

Thanks to the great impact it has had on the commercial software industry, the video game industry has evolved by offering different frameworks for the creation of video games [26]. These frameworks, also known as game engines, provide users with the programming experience, the set of functionalities necessary to design and create their own video games, through the definition of game elements and their behavior, quickly and efficiently [4].

Unity3D and Unreal Engine are two of the most used tools for videogame development. Both provide an engine that can be used to create multiplatform 2D and 3D video games. With its editor, developers can modify the physics, behavior of objects, interactions between them and also import 3D objects created with 3D modeling tools. Unreal Engine allows you to configure the behavior of objects in video games, with a complexity and graphic quality superior to Unity3D, but unfortunately, the learning curve of Unreal is steep, which is a disadvantage for novice users.

With the aim of bringing non-expert users closer to video game development, some EUD tools have emerged. For example, GameMaker [3] is a video game engine that can be used to create 2D video games by novice creators. With its editor, developers can configure the interactions and relations of the objects through drag-and-drop, thus simplifying and minimizing code writing. In addition, the tool has its own programming language, which provides expert users with a higher level of customization of object actions.

2.2 MDE and DSLs for Video Game Development

MDE focuses on the systematic use of software models to improve productivity and some other aspects of software quality. This discipline has demonstrated its potential to master the arbitrary complexity of software by providing a higher level of abstraction as well as raising the level of automation [8,25].

Domain-specific languages are systems of communication for design or development purposes, usually small in size, that are adapted to the needs of a given domain to solve a specific set of problems [13]. This adaptation is done in semantic terms, but also in notation and syntax, allowing an abstraction close to the one used in the domain [13].

A DSL is composed of three elements [8,20]: (i) the abstract syntax (or metamodel) that defines language concepts, the relationships between them and the rules that ensure when a model is well-formed; (ii) the concrete syntax that establishes notation; and (iii) the semantics that are normally defined through the translation to concepts of another target language with known semantics.

MDE approach is suitable to develop DSLs, as evidenced by the number of MDE frameworks for creating both graphical languages, such as Sirius or DSL

Tools, and textual ones, such as EMFText or Xtext. The visual syntax defines a mapping between the abstract elements of the language and a set of figures or shapes. On the contrary, in textual syntaxes the mapping is done between the language elements and grammatical rules. Once both abstract and concrete syntax are defined, it is necessary to provide a support tool that allows users to work with the DSL.

The MDE approach and DSLs can be used to support the video game development. For example, Gonzalez-Garcia et al. [15] propose a DSL for the development of multiplatform educational video games focused on non-expert users. In this same line is also the proposal of Solis-Martinez et al. [26], which present a specific notation based on the BPMN specification to define several key characteristics of video game logic. Furthermore, Nuñez-Valdez et al. [21] present a graphical editor intended for non-expert users to develop their own video games with a DSL.

2.3 Chatbots in Modeling and Programming Tasks

Although the term chatbot has traditionally been used only with text-only applications, due to the advances in Automatic Speech Recognition (ASR) and Natural Language Processing (NLP), the concept has become nowadays broader [9]. The objective of NLP is to facilitate the interaction between human and machine. To achieve this, it learns the syntax and meaning of user language, processes it and gives the result to the user [16]. In the case of the ASR systems, the aim is to translate a speech signal into a sequence of words. The purpose of this conversion is text-based communication or for the control of devices [12]. This renewed interest in chatbots has led large technology companies to create their own development frameworks. Some examples are Google Dialogflow, Amazon Lex and Microsoft Bot Framework, among others.

Not all chatbots are aimed to get information from the users, but they can also be used to create new content or to perform actions. These chatbots can also be related to computing, more specifically to modeling, as by demonstrated Perez-Soler et al. [23]. In that work, the authors detail how, through a messaging system such as Telegram, it is possible to create domain models. Furthermore, Jolak et al. [17] propose a collaborative software design environment that allows users to create UML models with multiple modes of interaction including, among others, voice commands.

Besides, Rosenblatt [24] proposes a development environment for programming via voice recognition. Vaziri et al. [27] present a compiler that takes the specifications of a web API written in Swagger and automatically generates a chatbot that allows the end-user to make calls to that API. Additionally, it is possible to work with ontologies through a chatbot, as proposed by Al-Zubaide et al. [5]. Their tool maps ontologies and knowledge in a relational database so that they can be used in a chatbot.

In the reviewed literature, there are some works about the use of chatbots or voice commands to deal with some specific languages. However, none of these

works deal with video game development. In addition, these works are not generalizable enough to support different domains and there is no evidence of methods or tools to automate the development of conversational agents for working with DSLs.

3 A Framework for Using and Creating Conversational Agents for DSLs

This section describes an approach for working with DSLs through a chatbot, its implementation details and a systematic method to automate their construction.

3.1 Approach

DSL support tools enhanced with conversational bots are equipped with a messaging window to manipulate the running model or program via natural language. In this way, the chatbot enables the user to introduce voice or textual commands that will be processed and transformed into specific actions on the artefact under development.

Regardless of the concrete syntax developed for the DSL, the use of conversational agents with a DSL previously requires a mapping between its abstract syntax and the elements of every conversational agent, namely entities and intentions. First, all the concepts, properties and relationships defined in the abstract syntax are mapped to entities of the conversational agent. These entities are grouped into three categories, namely *Element*, *Attribute*, and *Relation*. Then, a set of specific intents for Creating, Reading, Updating or Deleting (CRUD) model elements must be provided. Figure 1 shows a classification of the intents according to whether the actions they perform depends on the focused model element (i.e., focus intents) or not focused (i.e., general intents). All the intents are pre-trained with a set of specific expressions to deal with them. For example, the *create element* general intention includes expressions in natural language, such as *create the [element] with the [attribute] [value]*.

In addition to the intents to manipulate the model elements, there are other intents aimed at improving the user experience during the conversation. For example, contextual awareness, i.e., the bot remembers the last action applied to the model and, depending on the focused element, provides hints to tell the user what to do next.

3.2 Implementation

The above-described approach has been developed by using the Dialogflow [1] conversational agent engine because it provides us with a complete Java API and training features to improve its performance by incorporating interaction logs. The current implementation requires that the abstract syntax of the DSLs must be written with Ecore, the de facto standard for meta-modeling of the Eclipse Modeling Framework (EMF) [2]. The conversational agent's user interface was

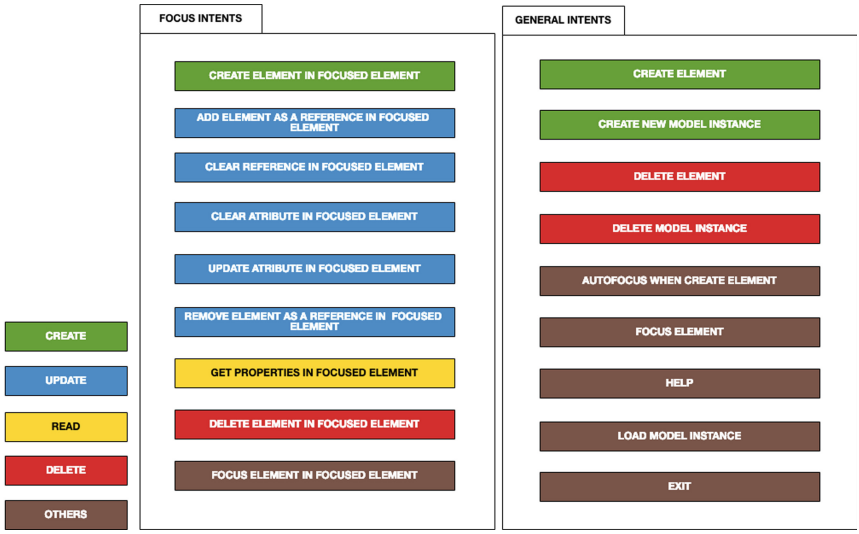

Fig. 1. Classification of intents for the conversational agent

developed as an Eclipse plug-in, fully integrated with the rest of the Eclipse IDE ecosystem.

Through the chatbot interface, the user sends a text or voice input to Dialogflow. The content of this input is processed, determining what intent it belongs to and what relevant information (entities) it contains. Afterwards, Dialogflow generates a response in JSON format, which includes the context of the current conversation, the action that the user wants to perform and the entities involved. Once the necessary data are extracted from Dialogflow's response, they are sent to an interpreter who determines what set of EMF operations (e.g: *EcoreUtil.delete*, *EcoreUtil.create*, etc.) should be applied on the active model. Once the operation is executed, the chatbot will provide the user with a message about the result of the operation. Figure 2 depicts the interactions among the above components.

3.3 Automated Generation of Conversational Agents

A particular application of MDE is MDD, which aims to automatically generate code by developing and transforming software models and, hence, reduces complexity and development efforts [28]. By following the MDD paradigm, a systematic method to (semi-)automatically generate chatbots for DSLs is defined below (see Fig. 3).

First, the DSL developer must create the language's abstract syntax (i.e., EMF domain model) and create an empty Dialogflow project. The latter action cannot be automated because Dialogflow does not permit to create projects programmatically, so that it is necessary to register the new conversational agent

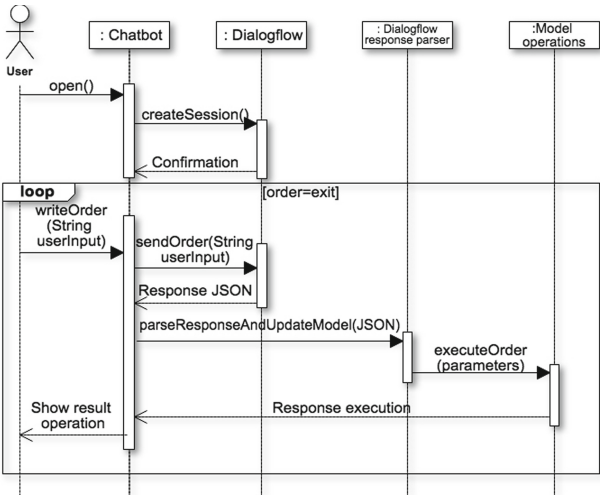

Fig. 2. Sequence diagram of the Eclipse chatbot plug-in

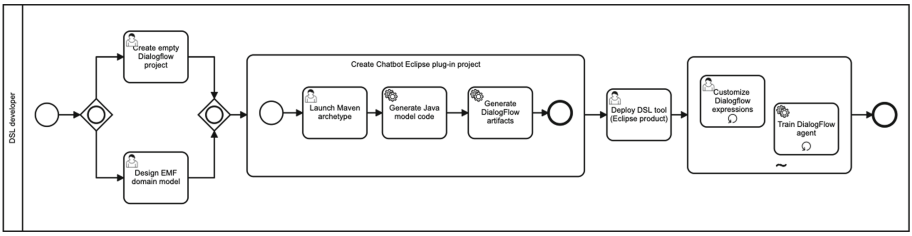

Fig. 3. Chatbot plug-in generation

and acquire the developer API keys through the Dialogflow's management web tool. Second, the Eclipse plug-in project must be developed to interact with the chatbot. For that, a dedicated Maven archetype to automate this activity is available to the DSL developer. The archetype asks the user for the project name, the ecore file with the abstract syntax and the Dialogflow's access key. The archetype consists of a parameterized template of an Eclipse project containing the libraries required to interact with the conversational agent engine and the EMF-based models. Two processes are triggered by this archetype: (i) a Model to Text (M2T) transformation process to generate, from the language meta-model, a series of code artifacts required to manage Java classes and to perform EMF operations over the running models; and (ii) another M2T process to populate the Dialogflow project with the required entities, intents and expressions via a REST API. Third, the generated plug-in can be later integrated with other plug-ins to build a complete Eclipse product and distribute it. Finally, DSL developers can customize the set of expressions suitable for each Dialogflow project's intent. Additionally, thanks to the DialogFlow's built-in machine learning algorithms,

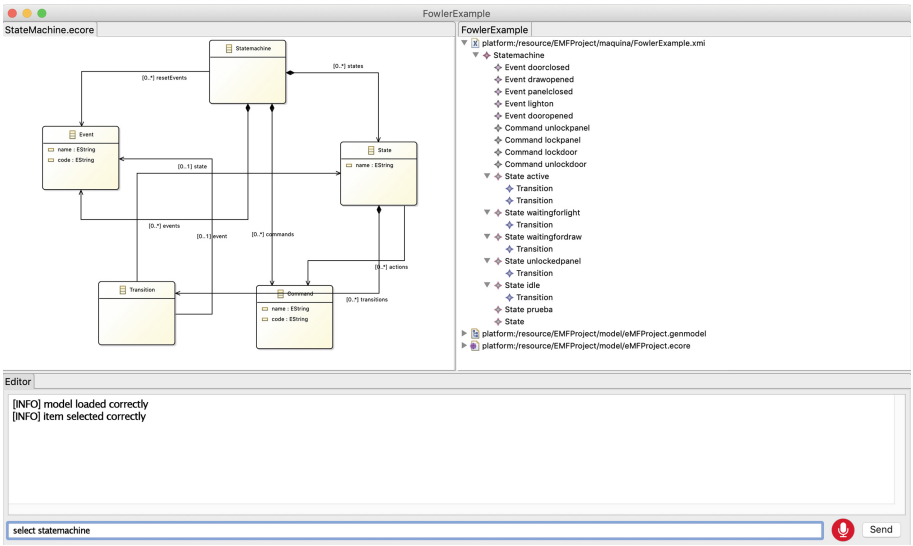

Fig. 4. Eclipse product including a chatbot interface

the agent's precision in matching user inputs can improve over time after the software is released.

4 Case Study

Since the nineties, state machines have been one of the most widely used resources in video game development [30]. The state machines are used to generalize all possible situations and reactions that occur by the action of the human player. Therefore, to verify the validity of the approach proposed in this research, a chatbot is presented to work with models of state machines, specifically the one defined by Martin Fowler in his book *Domain-Specific Languages* [13].

The Eclipse plug-in to work with that DSL through a chatbot interface was generated using the method and tools previously described. Afterwards, a desktop Eclipse application was built, comprising: (i) a read-only Ecore diagram viewer to show the visual representation of the language meta-model; (ii) the XMI tree viewer of the running model; and (iii) the chatbot interface to interact with the conversation agent using voice or text inputs. Figure 4 depicts the user interface of the product whilst working with the Miss Grant's secret compartment model [13], commonly used as an initial example for a lot of modeling tools.

With this tool, the user can edit models by adding states, events and commands, as well as editing their properties and relationships. Firstly, the DSL user must create a project by using the *create a new project* option. In a pop-up menu, the user must enter the project name and the name desired for the state

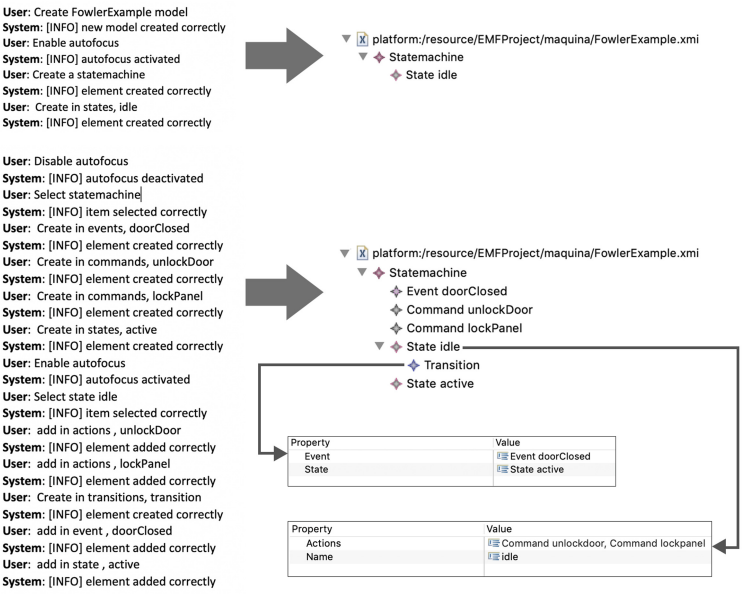

Fig. 5. Example of chatbot usage

machine to be created. Then, the user will be able to use the chatbot interface. Two snippets of the dialog between the user and the bot to model Fowler's example are shown in Fig. 5. The commands sent to the chatbot, the received answers and the impact on the running model can also be observed.

5 Conclusions

Video games are no longer only focused on leisure, but are increasingly used by different fields such as health or education. The growing interest in being able to create video games has led to the emergence of EUD tools, thanks to which non-programmers can create their own games using specific languages aimed at simplifying technological barriers.

In addition, new and improved methods of user-machine interaction are emerging. For example, thanks to advances in natural language processing and speech recognition, chat robots are being widely used in many applications. In this sense, this paper explores a new way of interacting with DSLs through conversational agents.

This document presents a framework for the use and creation of specific conversational interfaces for these domains focused on the creation of video games. The first hypothesis raised in this paper was tested by developing an Eclipse-based tool with a chatbot interface capable of working with Martin Fowler's DSL State Machine. However, the second hypothesis cannot be fully supported,

because the method and tool for automating the creation of conversational agents have been tested with a single language, so more applications are still required.

As future work, we consider two main lines. First, conduct a usability assessment to collect user feedback. We must verify that the intentions and the set of possible expressions that users may say match those intentions are appropriate for their needs and expectations. The training phrases initially provided will be expanded so that conversational agents achieve a higher success rate during recognition. Second, more quantitative studies will be conducted to measure the effort required to develop models using chatbot interfaces compared to the use of common notations (visual or textual).

We believe that the inclusion of chat robots during EUD processes through DSLs focused on creating video games provides users with a more adaptable way to use these languages. Currently, users of these DSLs must be accustomed to the concrete notation explicitly defined by the DSL designer. However, with the approach presented in this paper, DSL users will be able to work in a more flexible environment, using their own natural language expressions. In short, this approach can contribute to greatly reducing the learning curve of development languages.

References

1. DialogFlow. https://dialogflow.com/
2. Eclipse Modeling Framework (EMF). https://www.eclipse.org/modeling/emf/
3. GameMaker. https://www.yoyogames.com/gamemaker
4. Game engines how do they work? (2019). https://unity3d.com/what-is-a-game-engine
5. Al-Zubaide, H., Issa, A.A.Y.: OntBot: ontology based chatbot. In: International Symposium on Innovations in Information and Communications Technology, pp. 7–12 (2011)
6. Bock, B.C., et al.: Exercise videogames, physical activity, and health: Wii heart fitness: a randomized clinical trial. Am. J. Prev. Med. **56**, 501–511 (2019)
7. Bracq, M.S., et al.: Learning procedural skills with a virtual reality simulator: an acceptability study. Nurse Educ. Today **79**(May), 153–160 (2019)
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan Claypool, San Rafael (2012)
9. Brandtzaeg, P.B., Følstad, A.: Why people use chatbots. In: Kompatsiaris, I., et al. (eds.) INSCI 2017. LNCS, vol. 10673, pp. 377–392. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70284-1_30
10. Burnett, M.: What is end-user software engineering and why does it matter? In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf, V. (eds.) IS-EUD 2009. LNCS, vol. 5435, pp. 15–28. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00427-8_2
11. Dörner, C., Yetim, F., Pipek, V., Wulf, V.: Supporting business process experts in tailoring business processes. Interact. Comput. **23**(3), 226–238 (2011)
12. Errattahi, R., Hannani, A.E., Ouahmane, H.: Automatic speech recognition errors detection and correction: a review. Procedia Comput. Sci. **128**, 32–37 (2018)
13. Fowler, M.: Domain Specific Languages, 1st edn. Addison-Wesley Professional, Boston (2010)

14. Gaouar, L., Benamar, A., Le Goaer, O., Biennier, F.: HCIDL: Human-computer interface description language for multi-target, multimodal, plastic user interfaces. Future Comput. Inform. J. **3**(1), 110–130 (2018)
15. González García, C., Núñez-Valdez, E.R., Moreno-Ger, P., González Crespo, R., Pelayo G-Bustelo, B.C., Cueva Lovelle, J.M.: Agile development of multiplatform educational video games using a domain-specific language. Univ. Access Inf. Soc. **18**(3), 599–614 (2019)
16. Jain, A., Kulkarni, G., Shah, V.: Natural language processing. Int. J. Comput. Sci. Eng. **6**, 7 (2018)
17. Jolak, R., Vesin, B., Chaudron, M.R.V.: Using voice commands for UML modelling support on interactive whiteboards: insights & experiences. In: CIbSE 2017 - XX Ibero-American Conference on Software Engineering, pp. 85–98 (2017)
18. Ko, A., et al.: The state of the art in end-user software engineering. ACM Comput. Surv. **43**(3), 1–44 (2011)
19. Maceli, M.G.: Tools of the trade: a survey of technologies in end-user development literature. In: Barbosa, S., Markopoulos, P., Paternò, F., Stumpf, S., Valtolina, S. (eds.) IS-EUD 2017. LNCS, vol. 10303, pp. 49–65. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58735-6_4
20. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
21. Núñez-Valdez, E.R., García-Díaz, V., Lovelle, J.M.C., Achaerandio, Y.S., González-Crespo, R.: A model-driven approach to generate and deploy videogames on multiple platforms. J. Ambient Intell. Humaniz. Comput. **8**(3), 435–447 (2017)
22. Pane, J., Myers, B.: More natural programming languages and environments. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development. Human-Computer Interaction Series, vol. 9, pp. 31–50. Springer, Dordrecht (2006). https://doi.org/10.1007/1-4020-5386-X_3
23. Pérez-Soler, S., Guerra, E., de Lara, J.: Collaborative modeling and group decision making using chatbots in social networks. IEEE Softw. **35**(6), 48–54 (2018)
24. Rosenblatt, L.: VocalIDE: an IDE for programming via speech recognition. In: Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 417–418. ACM (2017)
25. Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5), 19–25 (2003)
26. Solís-Martínez, J., Espada, J.P., García-Menéndez, N., Pelayo G-Bustelo, B.C., Cueva Lovelle, J.M.: VGPM: using business process modeling for videogame modeling and code generation in multiple platforms. Comput. Stand. Interfaces **42**, 42–52 (2015)
27. Vaziri, M., Mandel, L., Shinnar, A., Siméon, J., Hirzel, M.: Generating chat bots from web API specifications. In: Proceedings of the ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, vol. 14, pp. 44–57 (2017)
28. Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. IEEE Softw. **31**(3), 79–85 (2014)
29. Wijman, T.: Mobile revenues account for more than 50 of the global games market as it reaches 137.9 billion in 2018 (2018). https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/
30. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer, Heidelberg. https://doi.org/10.1007/978-3-319-63519-4. http://gameaibook.org