Software and Systems Modeling
https://doi.org/10.1007/s10270-021-00904-y

EXPERT VOICE

Towards a model-driven approach for multiexperience AI-based user interfaces

Elena Planas[1] · Gwendal Daniel[1] · Marco Brambilla[2] · Jordi Cabot[3]

Received: 2 June 2021 / Accepted: 14 June 2021

Abstract
Software systems start to include other types of interfaces beyond the "traditional" Graphical-User Interfaces (GUIs). In particular, Conversational User Interfaces (CUIs) such as chat and voice are becoming more and more popular. These new types of interfaces embed smart natural language processing components to understand user requests and respond to them. To provide an integrated user experience all the user interfaces in the system should be aware of each other and be able to collaborate. This is what is known as a *multiexperience* User Interface. Despite their many benefits, multiexperience UIs are challenging to build. So far CUIs are created as standalone components using a platform-dependent set of libraries and technologies. This raises significant integration, evolution and maintenance issues. This paper explores the application of model-driven techniques to the development of software applications embedding a multiexperience User Interface. We will discuss how raising the abstraction level at which these interfaces are defined enables a faster development and a better deployment and integration of each interface with the rest of the software system and the other interfaces with whom it may need to collaborate. In particular, we propose a new Domain Specific Language (DSL) for specifying several types of CUIs and show how this DSL can be part of an integrated modeling environment able to describe the interactions between the modeled CUIs and the other models of the system (including the models of the GUI). We will use the standard Interaction Flow Modeling Language (IFML) as an example "host" language.

Keywords Multiexperience development platform (MXDP) · Model-driven development (MDD) · bots · Conversational user interface (CUI)

1 Introduction

The specification and the implementation of the User Interface (UI) of a system is a key aspect in any software development project. In most cases, this UI takes the form of a Graphical User Interface (GUIs) that encompasses a number of visual components [1] [19] to offer rich interactions between the user and the system. But nowadays, a new generation of UIs which integrate more interaction modalities (such as chat, voice and gesture) is gaining popularity [27]. Moreover, many of these new UIs are becoming complex software artifacts themselves, for instance, through AI-enhanced software components that enable even more natural interactions, including the possibility to use Natural Language Processing (NLP) via chatbots or voicebots. These NLP-based interfaces are commonly referred as Conversational User Interfaces (CUIs).

Even more, many times several types of UIs are combined as part of the same application (e.g., a chatbot in a web page), what it is known as *Multiexperience User Inter-*

Communicated by Iris Reinhartz-Berger.

✉ Elena Planas
eplanash@uoc.edu

Gwendal Daniel
gdaniel@uoc.edu

Marco Brambilla
marco.brambilla@polimi.it

Jordi Cabot
jordi.cabot@icrea.cat

[1] Universitat Oberta de Catalunya, 08018 Barcelona, Spain

[2] Politecnico di Milano, 20133 Milano, Italy

[3] ICREA - Universitat Oberta de Catalunya, 08010 Barcelona, Spain

[1] www.usability.gov/how-to-and-tools/methods/user-interface-elements.html.

Published online: 02 August 2021

face. These multiexperience UIs may be built together in the development environment provided by a Multiexperience Development Platform (MXDP). According to Gartner[2], *MXDPs serve to centralize life cycle activities—designing, developing, testing, distributing, managing and analyzing — for a portfolio of multiexperience apps. Multiexperience refers to the various permutations of modalities (e.g., touch, voice and gesture), devices and apps that users interact with on their digital journey across the various touchpoints. Multiexperience development involves creating fit-for-purpose apps based on touchpoint-specific modalities, while at the same time ensuring a consistent user experience across web, mobile, wearable, conversational and immersive touchpoints*. According to Gartner, by 2023, more than 25% of the mobile apps, progressive web apps, and conversational apps at large enterprises will be built and/or run through a multiexperience platform.

Despite the increasing popularity of these new types of user interfaces, and the rich possibilities they offer when combining among them or with preexisting GUIs, we are still missing of dedicated software development methods and techniques that facilitate their definition and implementation. The fragmented ecosystem of languages, libraries and APIs for building them (many times, proprietary and vendor-specific) add a new challenge to their development. Besides, their development is often completely separated from that of the rest of the system, as an ad hoc extension. This raises significant integration, evolution and maintenance challenges. Developers need to handle the coordination of the cognitive services to build multiexperience UIs, integrate them with external services, and worry about extensibility, scalability, and maintenance.

We believe a model-driven approach for MXDP could be an important first-step towards facilitating the specification of rich UIs able to coordinate and collaborate to provide the best experience for end-users. Indeed, most non-trivial systems adhere to some kind of model-based philosophy [8] where software design models (including GUI models) are transformed into the production code the system executes at run-time. This transformation can be (semi)automated in some cases. In this sense, the contribution of this paper is twofold:

- We propose to raise the abstraction level used in the definition of this new breed of conversational and smart interfaces, facilitating their specification and implementation on a variety of platforms.
- We show how these interface models can be used in conjunction with GUI models to combine the benefits of all these different types of interfaces.

[2] www.gartner.com.

The rest of the paper is structured as follows: Section 2 provides the background and introduces some preliminary concepts used through the paper; Section 3, which is focused on Conversational User Interfaces (CUIs), presents a new Domain Specific Language (DSL) for specifying these interfaces; Section 4 shows an extension of the standard Interaction Flow Modeling Language (IFML) to describe the links between the multiexperience UIs and the other software components; Section 5 reviews the related work; Section 6 outlines several lines for the further work; and finally Sect. 7 draws the conclusions.

2 Background and preliminary concepts

Before presenting our model-driven approach for multiexperience applications, we review in this section some basic concepts on User Interfaces (UIs) and their evolution, with a special focus on Conversational User Interfaces.

2.1 Evolution of user interfaces

The history of computer User Interfaces is that of the improvement of the user experience. At the beginning, text-based User Interfaces, such as Command-Line Interfaces (CLIs) which evolved from batch computing, allowed to process commands in the form of plain text. The next relevant improvement was the introduction of Graphical User Interfaces (GUIs), which allow interacting through graphical elements such as icons, cursors, and buttons. In the last decades, GUIs evolved to a new paradigm known as Natural User Interfaces (NUIs), which integrate more interaction modalities such as touch, voice or gestures to make the user interactions with the software feel as natural as possible.

This *naturalization* process often implies embedding some AI components in the UIs (for instance, for voice and gesture recognition), what result in the Intelligent User Interfaces (IUI) [45]. A clear example of Intelligent UIs are the smart Conversational User Interfaces (CUIs)[3] [28], which allow users to interact with computer-based applications via natural language.

Note that an application may include more than one User Interface as part of an integrated *multiexperience* User Interface enabling the user to interact with the application in multiple ways depending on the more convenient way to accomplish a certain task in a given context. Even in parallel. The development of this new type of multiple and complex

[3] Not all CUIs rely on neural networks or other AI techniques for understanding the user (e.g., you could use more grammar-like approaches to parse the input text) but nowadays most CUIs do rely on an AI-based NLP engine for better accuracy.

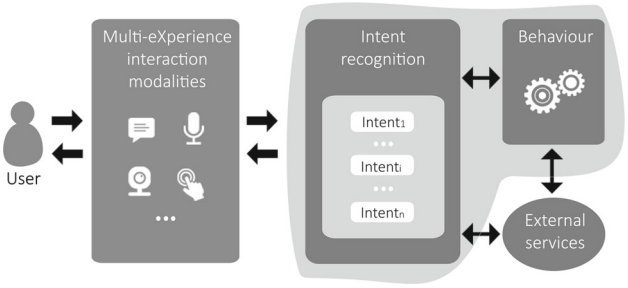


Fig. 1 Bot working schema

UIs is precisely the goal of the new emerging Multiexperience Development Platforms (MXDP).

2.2 Conversational user interfaces

CUIs are becoming more and more popular every day. The most relevant example is the rise of bots [27], which are being increasingly adopted in various domains such as e-commerce or customer service, as a direct communication channel between companies and end-users. A bot wraps a CUI as key component but complements it with a behavior specification that defines how the bot should react to a given user message.

Bots are classified in different types depending on the channel employed to communicate with the user. For instance, in *chat*bots the user interaction is through textual messages, in *voice*bots is through speech, while in *gesture*bots is through interactive images. Note that in all cases bots are the mechanism to implement a conversation, it just changes the medium where this conversation takes place. As such, bots always follow the similar working schema depicted in Fig. 1.

As you can see in the Figure, the conversation capabilities of a bot are usually designed as a set of *intents*, where each intent represents a possible user's goal when interacting with the bot. The bot then waits for its CUI front-end to detect a match of the user's input text (called *utterance*) with one of the intents the bot implements. The matching phase may rely on external Intent Recognition Providers (e.g., DialogFlow, Amazon Lex, IBM Watson,...). When there is a match, the bot back-end executes the required behavior, optionally calling external services for more complex responses; and finally, the bot produces a response that it is returned to the user (via text, voice or images, depending on the type of CUI).

As an example, we show in Fig. 2 a bot that gives the weather forecast for any city in the world. Following the working schema sketched in Fig. 1, this weather bot defines several intents such as *asking for weather forecast*. When a user writes (considering a chatbot) or says (considering a voicebot) *"What is the weather today in Barcelona?"* or *"What is the forecast for today in Barcelona?"*, the intent

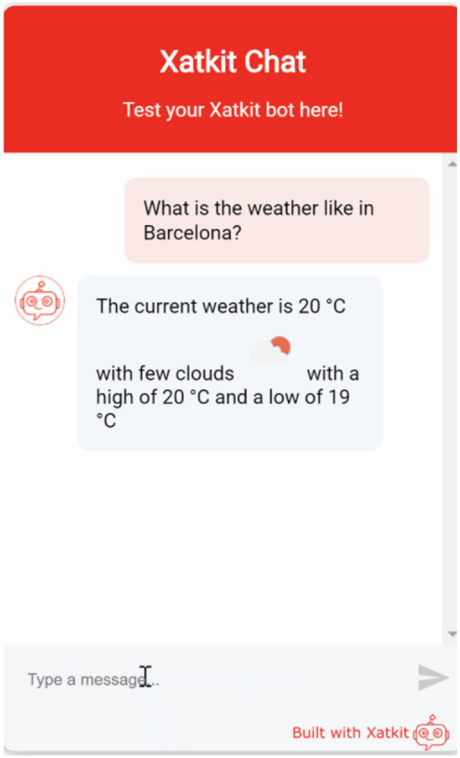


Fig. 2 Screenshot of our example chatbot created with Xatkit [17]

asking for weather forecast is matched and "Barcelona" is recognized as a city parameter (also called "entity") to be used when building the response. Then, the bot calls an external service (in this case, the REST API of OpenWeather[4]) to look up this information and give it back to the user.

3 Modeling smart conversational user interfaces

In this section we describe how the Model-Driven Development approach may be applied to specify bots comprising some type of Conversational User Interface (e.g., chatbots). To this end, we present a new Domain Specific Language (DSL), that generalizes the one discussed in [17] to cover all types of CUIs and not just chatbots and follows a state-machine semantics to facilitate the definition of more complex flows and behaviors in the interaction with the users.

A DSL is defined through two main components [26]: (i) an *abstract syntax* (metamodel) which specifies the language concepts and their relationships (in this context, generalizing the primitives provided by the major intent recognition platforms used by bots such as [3,21,24]), and (ii) a *concrete syntax* which provides a specific (textual or graphical)

[4] https://openweathermap.org.

representation to specify models conforming to the abstract syntax.

In the next subsections we provide the abstract syntax of our language split into three packages in order to facilitate its readability: the *intent package* metamodel (see Sect. 3.1), the *behavioral package* metamodel (see Sect. 3.2), and the *runtime package* metamodel (see Sect. 3.3). Besides, we provide a possible concrete syntax, to show how the concepts of the metamodel are instantiated in our running example (see Sect. 3.4).

3.1 Intent package

The *Intent Package* metamodel (see Fig. 3) describes the set of concepts used for modeling the intent definitions at design time.

It defines a top-level *Library* class containing a collection of *Event*s, representing *occurrences that have some consequence for the system* [32]. *Event*s, which are identified by its name, are classified in a hierarchy according to the UML [32] and the IFML [31] metamodels. There are two types of *Event*s: *ThrowingEvent*s (events that are generated by the UI) and *CatchingEvent*s (events that are captured in the UI and that trigger a subsequent change). In its turn, we classify *CatchingEvent*s into two sub-categories: *UserEvent*s (events produced by the user) and *SystemEvent*s (events produced by the system itself, usually provided by *ExternalService*s, which represent a high-level abstraction of external platforms and providers).

*Intent*s, which in the context of CUIs are the most relevant type of *UserEvent*s, represent the user intentions or requests that bots have to address. For each *Intent* we need to store a set of training sentences (*Text*) in a specific language, which are input examples used to detect the user intention underlying a textual message. The *Text* of a training sentence is split into several *TextFragment*s representing input training sentence parts—typically words—to match. Note that we assume voicebots (or other bots which are not dealing with text) have an internal pre-processing which translates all the interactions to text in order to be able to automatize its treatment, as typically done in industrial practice. As a consequence, in this package we can address all *Intents* directly as text, just like it occurs in chatbots. Besides, *Intent*s can optionally have *Parameter*s which have a *name* and an *entity*, and allow to define specific characteristics of an *Intent*. We define the *EntityType* enumeration to save the values that a parameter may acquire. Note that, this enumeration includes basic types (String, Integer, Boolean, etc) as well as other more specific types (Date, Time, Location, City, etc) which are supported by most natural language understanding platforms such as `dialogFlow`[5].

[5] https://cloud.google.com/dialogflow.

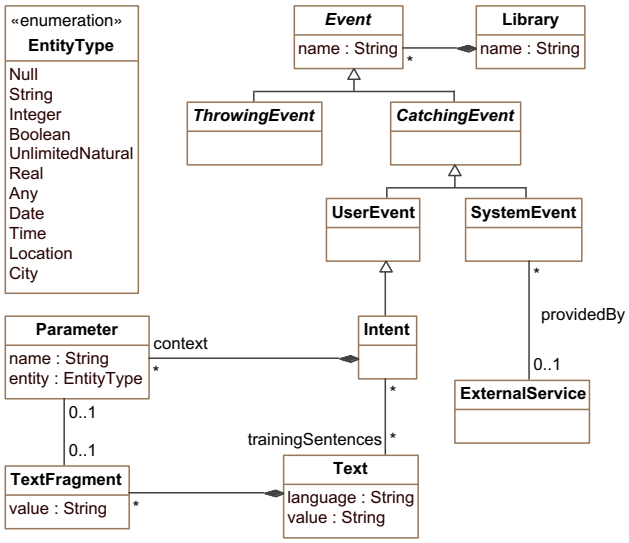


Fig. 3 Intent package metamodel

3.2 Behavioral package

The *Behavioral package* metamodel (see Fig. 4) defines the set of concepts used for modeling the execution logic of the *Intent package*. For this aim, we reuse the UML state-machine formalism [32]. Roughly, a state-machine is a graph where vertices represent the *State*s and arcs represent the *Transition*s. State machine execution is triggered by appropriate *Event* occurrences. In the context of this paper, this representation allows to express the valid (conversational or event-driven) interaction flows between a user and a bot.

A *State* models a situation in the execution of the *Behavior* during which some invariant condition holds. A particular state is the *InitialState*, which is, as its name indicates, the initial state where the bot is when starting its execution. Each *State* defines a *body*, which is the *Behavior* executed when entering the state. All these *Behavior*s consist of a set of *Action*s which specify, in a concrete *language*, the specific functions performed. These *Action*s may be provided by an *ExternalService*, such as platforms and providers.

*State*s are connected through *Transitions*. A *Transition* represents a single directed arc from a single *source State* and terminating on a single *target State*. We distinguish two types of *Transition*s: *AutomaticTransitions* (which are triggered automatically) and *GuardedTransitions* (which are triggered when a specific guard holds). A *GuardedTransition* may be triggered by several *Event*s (in our context, usually an *Intent*). When multiple triggers are defined for a *Transition*, they are logically disjunctive, that is, if any of them are enabled, the *Transition* will be triggered. Besides, a *GuardedTransition* may have an associated *guardedConstraint*, which allows a fine-grained control over the firing of the *Transition*. The constraint is evaluated when an *Event* occurrence is dispatched

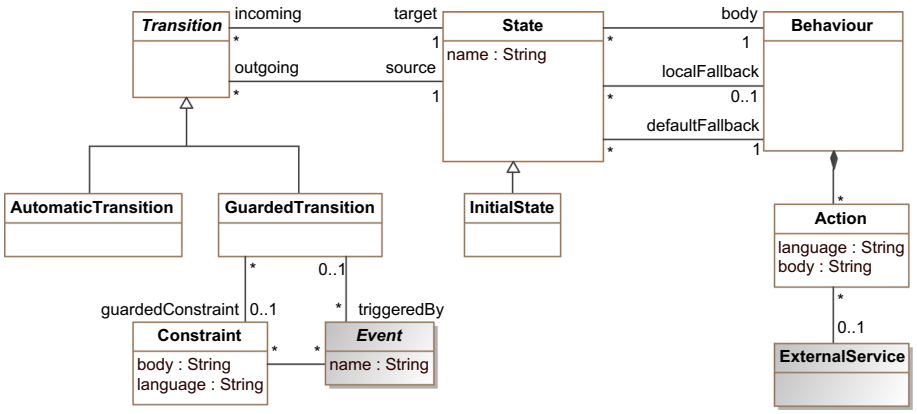


Fig. 4 Behavioral package metamodel

by the state-machine. If the constraint is true at that time, the *Transition* may be enabled, otherwise, it is disabled.

When none of the output Transitions hold, the bot executes the general *defaultFallback* behavior, or, if defined in the state where the bot is on, the *localFallback* behavior to try to recover from this unexpected situation (e.g., a matched intent in a state where that intent was not supposed to be matched).

3.3 Runtime package

The *Runtime package* metamodel (see Fig. 5) defines some of the classes used during the runtime execution of the deployed bot. This is useful to illustrate how the bot works but also to describe some of the internal structures required to build a bot engine that can be run CUIs as the ones described before. We focus on the classes related with the *Intent package* and we avoid the classes from the *Behavioral package*, since we consider the former is the most relevant for the topic of this paper and the later are similar to any proposal for enacting state machines.

In this package, the *UserInput* metaclass represents the user utterances. Depending on the type of UI we are dealing with, the user input may be in the form of a *SpokenInput* (in voicebots), a *TextualInput* (in chatbots) or a *GestualInput* (in gestual interfaces). Each of these inputs represent *Data* in form of *Speech* (for voicebots), *Text* (for chatbots) or *Video* (for gestual interfaces). In all cases, these data is translated into text (see the associations *speechToText* and *videoToText*). To simplify, in this paper, we do not model the external AI libraries that could be used to perform this translation, but in practice, all these different input types are translated into text (and later on converted back if necessary) for an homogeneous treatment by the CUI.

A *UserInput* may match with one or more *Intents* defined in the *Intent Package*[6]. We assume there exist an intent recognition provider (out of the scope of this paper) used to classify *UserInput*s into one or more *Intent*s. Each recognized intent is represented by the *MatchedIntent* class, which stores the level of recognition confidence provided by the intent recognition provider.

Finally, for each *Parameter* of an *Intent*, the corresponding *MatchedIntent*s keep its *value* in the association class *ParameterValue*. Each of these *ParameterValue*s correspond to a specific *TextFragment* of the analogous *UserInput* which has derived the *MatchedInput*.

3.4 Concrete syntax

In order to complete the definition of our DSL, in this section we provide an example of textual concrete syntax based on Xatkit [17]. The syntax is presented through examples using the running case study introduced in Sect. 2.2. The complete syntax of this language and its instantiation can be seen in GitHub[7]. Alternative syntaxes (including graphical ones) can be easily mapped to the abstract syntax presented so far[8].

The first step is defining the asking for weather forecast intent (see Listing 1). Note how as part of the intent, the bot collects the name of the desired city.

Listing 1 Intent definition for the weather bot.

```
val howIsTheWeather = intent("HowIsTheWeather")
  .trainingSentence("How is the weather today in CITY?")
  .trainingSentence("What is the forecast for today in CITY?")
  .parameter("cityName").fromFragment("CITY").entity(city());
```

Then, the bot waits in an initial state of the state-machine (*awaitingInput*) for the intent to match (see Listing 2).

Listing 2 Intent awaiting for the weather bot.

```
awaitingInput
  .next()
  .when(intentIs(howIsTheWeather)).moveTo(printWeather);
```

[6] Classes from another packages are shown shaded.

[7] https://github.com/xatkit-bot-platform/xatkit-examples/tree/master/WeatherBot.

[8] Xatkit itself has drastically changed its concrete syntax moving from an External DSL to a Java-based Fluent API while keeping the same internal metamodel.

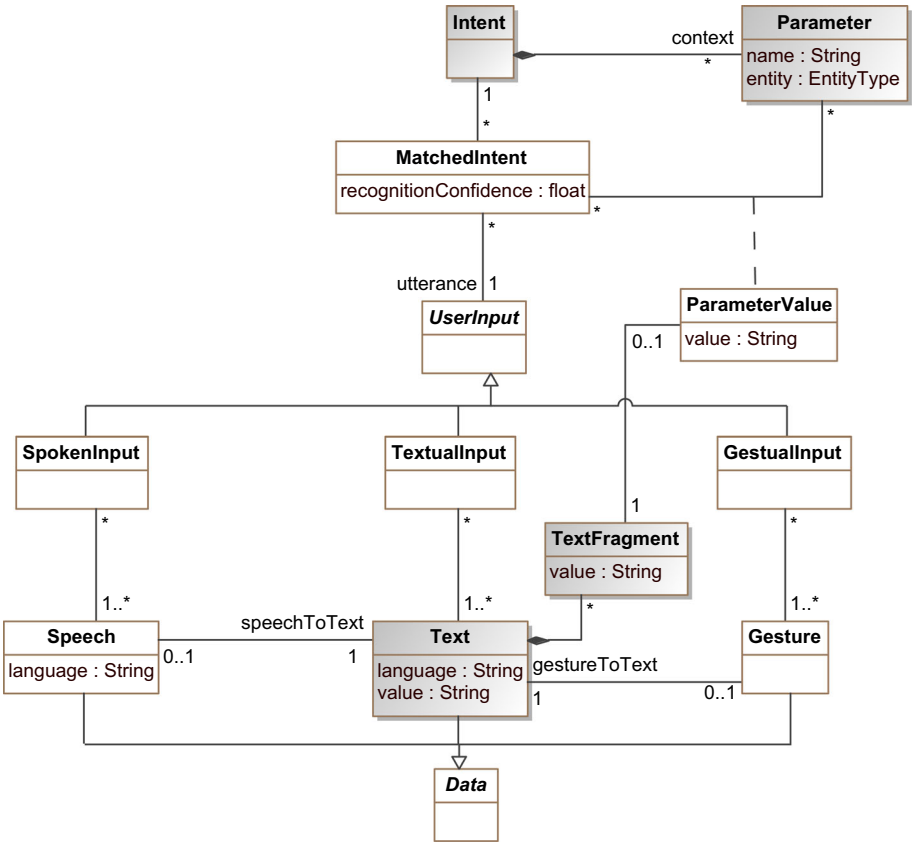


Fig. 5 Runtime package metamodel

Once the intent is matched, the transition to the *printWeather* state is executed to actually process the request. With this purpose, the bot calls an external service (in this example, the REST API OpenWeather) passing the requested city as parameter (see Listing 3). To simplify, we have omitted the code corresponding to the API call.

Listing 3 Intent resolution for the weather bot.

```
printWeather
  .body(context -> {
    String cityName = (String) context.getIntent().getValue("cityName");
    // Call the OpenWeather REST API providing the cityName as a
    // parameter and return the proper message
    ...
  })
  .next()
  .moveTo(awaitingInput);
```

4 Modeling multiexperience applications

A key element of multiexperience UIs is that every type of interface communicates and collaborates with each other, across different devices and applications, thus giving the user the perception of a unified and shared experience across the different modalities and interfaces. The expectation of users when aiming at a given objective is that he can interact with a variety of applications in different contexts in a seamless way to reach the target.

For instance, Fig. 6 shows a typical scenario of multiexperience user interaction. Suppose that a customer on a Sunday morning wants to buy a new technical product (a cell phone or a home theater system). He first interacts with his home assistant (like Alexa or Google assistant) to ask it to find the best nearby tech store open on Sunday. With this information in mind, he looks at the store web site on his PC and, being satisfied with the kind of store, he asks the web site chatbot to find the type of products he is looking for. After browsing the various alternatives, he finds one item he likes, and sets the place and the product as preferences on his mobile phone. He reads the details of the product on the phone while walking to his car. When he reaches the car, he transfers the information about the place to the car navigation system and drives there. Finally, in the stores he looks around, tries various items, reads the reviews about them on a dedicated mobile app, and finally picks up the product and pays for it.

This kind of dynamic and seamless interaction demands a variety of complex design and implementation mechanisms to be put in place. In the previous section, we have discussed our approach for the specification of CUIs. In this section, we will see how they can be integrated in other, more general, languages to become part of a multiexperience development project that are able to satisfy requirements like the ones described in the above scenario.

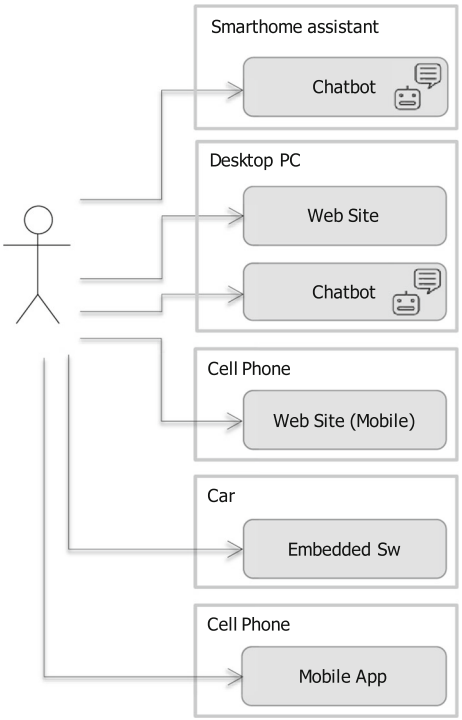


Fig. 6 Use case diagram representing a typical scenario of multiexperience user interaction

As before, given the cost and complexity of developing comprehensive multiexperience UIs, MDD can play an important role in the specification and implementation of the user interaction paradigms and of the respective UIs realizations.

In particular, for describing the integration of the DSL presented in Sect. 3 within a broader-scope user interaction design at the modeling level, we will make use of the Interaction Flow Modeling Language (IFML) [31], one of several existing modeling languages for UI design. We describe various levels of integration and we discuss the advantages of a model-based MDXP approach in our context.

IFML is an international standard defined by the OMG that aims at the platform-independent description of graphical user interfaces for applications accessed or deployed on a variety of systems including computers, mobile phones, tablets, and smart devices. It supports event-driven specification of the interactions and it integrates seamlessly with other languages for the specification of the business logic, including UML and BPMN.

The focus of IFML is on the structure and behavior of the application front-end as perceived by the end user, i.e., the view part of the application. Therefore, modeling the UI and interaction with IFML amounts to addressing the following aspects: The *view structure and components* that define the interface shapes, and the contents visible in the UI; The *events specification*, which consists of the definition of events (coming from user's interaction, application logic, or external agents) that may affect the state of the UI; The reference to *actions* triggered by the user's events; The *events* triggered in the user interaction and their effects in terms of action execution and on the state of the UI; And the *parameter binding* between the elements of the UI and the triggered actions.

Furthermore, IFML can be complemented with external models for connecting to any kind of content model (representing databases, ontologies, file systems or other resources) and any kind of dynamic model (describing the business logic behind the application front end). Various implementation exist for this modeling languages, including *WebRatio*[9] [1] and *IFMLedit.org* [10] [6], which provide modeling, validation, and code generation capabilities.

For granting a multiexperience UI featuring a new CUI integration together with the GUI option already predefined in IFML, we focus on extending the IFML language with appropriate metamodel-level concepts, according to the extensibility rules provided within the language itself. We start from defining the different levels of integration that might be of interest for a multiexperience UI, based on increasing levels of complexity:

1. The first integration level consists in supporting **the positioning of the chatbot** within a specific page, window, or screen of the GUI, so that it becomes visible and accessible from it, alongside the other interface elements;
2. The second integration level is that of **data sharing** between the bot and the rest of the interface, thanks to various data sharing mechanisms;
3. The third level of integration is that of **interactive integration**, where the chatbot and the other elements of the UI are connected in a bi-directional way through interactions on one side which can trigger effects on the other and vice versa.
4. the fourth level of integration enables also **event-triggered side effects** through execution of actions on both sides.

Thanks to the modular and appropriate design of both the IFML metamodel and the chatbot metamodel, all the integration levels can be achieved with a very simple extension strategy, namely the addition of just two concepts in the IFML language, as described in the IFML metamodel excerpts reported in Figs. 7 and 8. Figure 7 shows the integration of the Chatbot metaclass, which defines the presence of the chatbot component. More precisely, Chatbot is defined as a specification of the ViewComponent IFML metaclass, exactly as any other typical component we usually add to interfaces, including Lists, Forms, Details panels, and so

[9] http://www.webratio.com.

[10] https://ifmledit.org/.

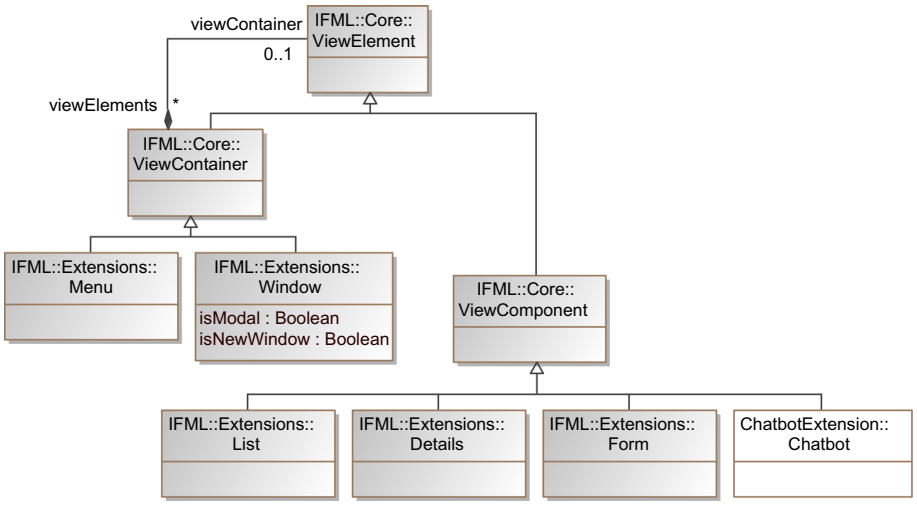


Fig. 7 IFML metamodel extension for the Chatbot, allowing positioning and interaction among components in the UI

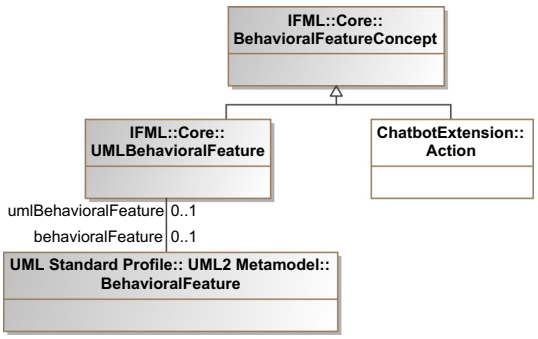


Fig. 8 IFML metamodel extension for the Chatbot Action, i.e., the corresponding IFML concept of the Action metaclass defined in the Chatbot metamodel, allowing invocation of actions in the chatbot from any UI components

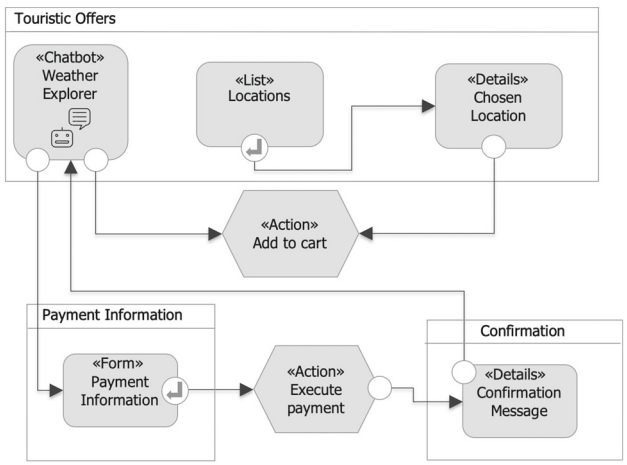


Fig. 9 IFML model example with chatbot integrated in the user navigation of a tourism web site

on. This allows the Chatbot to be positioned within ViewContainers such as Windows, as per the *composite* design pattern specified in the metamodel. Thanks to the general semantics of IFML, this also allows full support of the two additional levels of integration, because the component can send or receive data to/from other components, and can trigger navigations of the interface. Figure 8 shows the IFML metamodel extension for the chatbot *Action* metaclass, corresponding to the Action metaclass defined in the Chatbot metamodel. This allows the invocation of actions in the chatbot from any UI component. And vice versa the bot behavior definition can easily reuse and call IFML behaviors by working at the BehavioralFeatureConcept level when needed.

To demonstrate the advantages of this integration, Fig. 9 shows an example IFML model featuring a chatbot integrated within the UI of a tourism web site. The site includes a page *Touristic Offers*, including a list of touristic Locations. The user can select a location and thus see the details of the *Chosen Location*. From there, the user can add the location to the cart, for future purchase. The page also features a chatbot component named *Weather Explorer*, which allows the user to ask about the weather in specific locations. Besides asking questions and interacting directly with the chatbot, the user can also ask to perform specific tasks, such as add a location to the cart, or start the checkout process. These two requests are performed by the chatbot through the outgoing arrows, which lead, respectively, to the *Add to cart* action and to the *Payment Information* page. Information is transferred between IFML components according to the semantics of the language. In particular, this can be achieved according to two strategies: (i) by transferring parameter values along the navigation flow, i.e., the arrows connecting the components, which therefore carry incoming and outgoing parameters; (ii) by defining a shared Context environment, which is available as a global data space to all the components. These behaviors automatically extend to the chatbot component too.

Finally, Fig. 10 shows a deeper level of integration between the pieces of the user interface, where the chatbot is

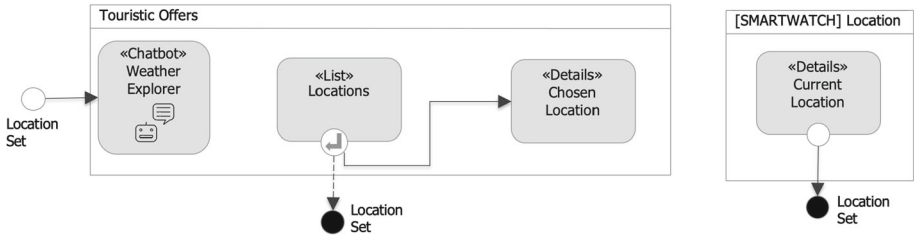


Fig. 10 IFML model example: the chatbot receives a *Location Set* event, which triggers a change of state in the internal bot state machine, independently on where the user triggered the event

directly solicited through event notifications. In IFML, any component, possibly from multiple devices and interfaces, can generate events (event generation is represented by black circles in the models). Such events may be captured by other components, which in turn may change their state or trigger some action according to the event meaning (event capturing is represented by a white circle). In the example, a *Location Set* event may be generated by the user selecting a location in the Web interface, or by a relocation of a smart watch or another device, which generates a GPS positioning event. The chatbot can capture these events and trigger a state change in the internal state machine that determines the evolution of the dialogue of the bot.

Thanks to this wide variety of integration options, a seamless multiexperience integration can be achieved. Indeed, the flexibility of the proposed metamodel allows to integrate blend interactions with the chatbot, with interactions happening on other components of a Web page, and with interactions happening with other interfaces (mobile apps, embedded systems, wearables, environmental sensors, and other devices), provided that they are able to send and receive events according to a classical event-driven architecture. The event-driven approach allows a very general and light-weight integration practice, which is widely supported. At the same time, it allows for direct integration at the user interface level (by sending around front-end events that trigger UI reactions), and at the back-end level (where the events can directly trigger changes in the state of the components or even on the data layer).

5 Related work

In this section, we summarize a large corpus of previous works that address how the model-driven development approach has been applied to model user interfaces, mainly focusing on Conversational User Interfaces and their integration with other types of interfaces in a multiexperience application.

Popularity of GUIs have resulted in a large number of GUI definition languages and run-time libraries for any imaginable programming language. For instance, some model-based proposals focusing on Web interfaces are W2000 (HDM) [4], OO-HDM [40], WebDSL [22], OOH-Method [20], WebML [14], RUX-Model [42] and HERA [44]) while others are focused on multi-device UI modeling (IFML [9], TERESA [7], MARIA [34] , MBUE [29], UsiXML [43] and UCP [38]. Among them, it is worth highlighting the OMG standard Interaction Flow Modeling Language (IFML) [9]. IFML has also been extended to cover other domains [10,23,33]. Our own IFML extension for CUIs has been presented in Sect. 4.

Instead, there is much less work focusing on the current emerging set of UIs, aiming at establishing more natural and conversational experiences, supported by the AI. These new type of UIs, which are hard to specify [37], test, verify and debug [39] and require a different and specialized skillset [25] could benefit as well from a model-based perspective. Right now, most of the initiatives focus on CUIs and, in particular, chatbots, mainly lead by commercial companies offering a kind of low-code/no-code [12] front-end to their specific chatbot platform. Some examples are: `Tock`[11], `Engati`[12] or `FlowXO`[13]. With a more research perspective, we have Xatkit [17] (formerly known as Jarvis [16]), a fully modular and extensible platform-independent chatbot modeling language which provides a meta-model and a textual DSL for defining all types of bots (which we have been used as basis for the generalized metamodel in Sect. 3); CONGA (ChatbOt modeliNg lanGuAge) [36] providing a unifying DSL for specifying some types of chatbots that can then be implemented on top of a couple of chatbot platforms (and even migrated from one to the other); and Baudat et al. [5] proposing wcs-OCaml, a new multi-tier chat-bot generator library designed for use with the reactive language ReactiveML. Some other works focus on voicebots. For example, `tortu`[14] which supports the visual creation of conversation flows and `VoiceFlow`[15] which facilitates a graphical DSL to create voice-based conversation flows that can be deployed on Google home or Alexa.

Nevertheless, all these initiatives focus on specific types of CUIs and treat them as standalone components and not as part of the more general systems in the context of MXDP. Some

[11] https://doc.tock.ai/tock/.

[12] https://www.engati.com.

[13] https://flowxo.com.

[14] https://tortu.io/.

[15] https://www.voiceflow.com/.

well-known low-code platforms like Mendix[16], GeneXus[17] or OutSystems[18] start to include chatbots as part of their system specification. But this support is rather limited and mostly consisting in either simple chatbot templates or in helping you to connect your application with an external AI component defined with a separate tool (e.g., one of the above).

To sum up, although there exist plenty of tools that provide model-based environments to define CUIs they are limited to specific types or bots or target only concrete technologies. Moreover, none of the tools we have analyzed support the MXDP approach as there is no way to link the CUI models with those of other CUIs or with the software models to provide richer multiexperience interactions.

6 Roadmap

We believe our proposal is a good first step towards this model-based MXDP vision. But there is still plenty of work to be done to advance in the model-driven development of powerful smart Conversational User Interfaces, especially as part of an integrated systems modeling process. In this section, we discuss a few of them.

6.1 Modeling the training process of CUIs

AI-based CUIs must be trained before deploying them. In some cases, the process is simple and fast but in others (e.g., CUIs for very specific domains, like the medical context [41]) we may need to reuse pre-trained language libraries and/or perform specific curation tasks on the input training sentences of the bot.

In order to able to completely define the end-to-end process of specifying and generating a CUI, we should provide new modeling primitives for specifying these tasks as part of our CUI metamodel, potentially looking at existing languages for workflow modeling.

This would be very useful also as a way to trace and explain the bot behavior. For instance, a voicebot can present accessibility issues by only properly recognizing voice messages from a certain set of English Speakers. Examining the training process could help as pinpoint the origin of the issue.

6.2 A repository of MXDP models and best practices

Many CUIs need to deal with a set of repetitive interaction patterns. Some examples would be greeting users and other chit-chat conversation flows, providing contact information, or detecting and stopping trolls. It would be ideal to have a community-driven repository of partial models for these repetitive tasks that we could easily import in new projects.

Similarly, we could also use this repository to share best practices around CUI design. There is plenty of literature around good design patterns for Graphical User Interfaces but not so much for CUIs. Same applies to patterns and best practices for combining several UIs part of a MXDP approach.

6.3 Modeling extensions

Our proposal covers the core aspects of modeling multiexperience interfaces. But there are several other dimensions of bots that could be added to the metamodel shown in Sect. 3. An example is role-based access control. We may need to define bots that hide some intents from some users (e.g., non-authenticated ones) or that respond differently to them. Integrating a RBAC metamodel (or any of its derivatives [30]) in our proposal would allow for a joint definition of the access-control policies together with the bot behavioral aspects.

6.4 Testing of MXDPs

There are some efforts for testing bots (e.g., [11]) but they focus on specific testings properties/techniques. Plenty of new research should be conducted to bring CUI testing to the same level of testing we have for regular interfaces.

This testing strategies should also cover system testing scenarios, where we test the collaboration scenarios between different CUIs part of a multiexperience application and not just each individual CUI in isolation. A roadmap for bot testing is also discussed in [13].

6.5 Automatic generation of MXDPs

For structured data sources, we can automatically derive a (limited but useful) behavioral/interface mode providing basic query and modification capabilities for the underlying data [2]. The same idea is implemented in many web application frameworks that offer scaffolding capabilities.

This same principle can be applied to CUIs. By looking at the data structure we could derive potential conversations users may want to have on top of that data. There are some initial works in this direction [15,18,35] but results are still preliminary. This generation can also take place at the MXDP level by combining generators for specific types of UIs to cover as well the glue code required to make them work as an integrated multiexperience application.

[16] https://www.mendix.com/.

[17] https://www.genexus.com/en/.

[18] https://www.outsystems.com/.

6.6 CUIs for modeling editors

Modeling editors could also benefit from the same advantages we get when adding CUIs to any other type of software systems. Modeling tools are known to be a barrier to entry for the adoption of modeling practices, CUIs could play a positive role here. For instance, a voicebot could help to create models as part of a collaborative design meeting. Or enable stakeholders to have a more direct participation in the modeling process even if they don't master the language notation.

7 Conclusions

The growing popularity of all kinds of intelligent Conversational User Interfaces is undeniable. And these interfaces are not isolated components. Instead, they are a core element of the software system that embeds them. As described by the MXDP initiative, CUIs must interact with the other interfaces of the system and have access to its functionality and resources.

The large number of libraries, platforms and technologies to develop CUIs complicates even more their development. To facilitate a global, integrated and platform-independent development process for CUIs, this paper has presented a model-based approach for CUIs covering both the design of each individual interface and the discussion of how such design could be combined with other software models for a complete software generation process.

This is a first step in this direction. As we discussed in the previous section, there are plenty of research challenges and potential improvements to advance towards this vision. We plan to tackle them next, starting with the chatbot testing and automatic generation aspects.

Acknowledgements This work has been partially funded by the Spanish government (PID2020-114615RBI00) and the AIDOaRt project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007350. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Austria, Czech Republic, Finland, France, Italy and Spain.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

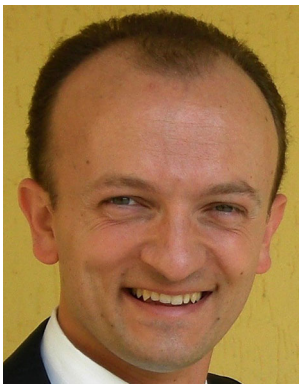
References

1. Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: Model-driven development based on omg's IFML with webratio web and mobile platform. In: Engineering the Web in the Big Data Era - 15th International Conference, ICWE Proceedings, pp. 605–608 (2015)
2. Albert, M., Cabot, J., Gómez, C., Pelechano, V.: Automatic generation of basic behavior schemas from UML class diagrams. Softw. Syst. Model. **9**(1), 47–67 (2010)
3. Amazon: Amazon Lex Website (2018). https://aws.amazon.com/lex/
4. Baresi, L., Garzotto, F., Paolini, P.: From web sites to web applications: New issues for conceptual modeling. In: Conceptual Modeling for E-Business and the Web, ER Workshops, *LNCS*, vol. 1921, pp. 89–100. Springer (2000)
5. Baudart, G., Hirzel, M., Mandel, L., Shinnar, A., Siméon, J.: Reactive chatbot programming. In: Proceedings of the 5th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems, REBLS@SPLASH, pp. 21–30. ACM (2018)
6. Bernaschina, C., Comai, S., Fraternali, P.: Ifmledit.org: model driven rapid prototyping of mobile apps. In: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, pp. 207–208. IEEE Press (2017)
7. Berti, S., Correani, F., Mori, G., Paternò, F., Santoro, C.: TERESA: a transformation-based environment for designing and developing multi-device interfaces. In: Extended abstracts of the 2004 Conference on Human Factors in Computing Systems, CHI, pp. 793–794. ACM (2004)
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on Software Engineering, 2nd edn. Morgan & Claypool Publishers (2017)
9. Brambilla, M., Fraternali, P.: Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML. Morgan Kaufmann (2014)
10. Brambilla, M., Mauri, A., Umuhoza, E.: Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end. In: Mobile Web Information Systems—11th International Conference, MobiWIS, *LNCS*, vol. 8640, pp. 176–191. Springer (2014)
11. Bravo-Santos, S., Guerra, E., de Lara, J.: Testing chatbots with charm. In: International Conference on the Quality of Information and Communications Technology, pp. 426–438. Springer (2020)
12. Cabot, J.: Positioning of the low-code movement within the field of model-driven engineering. In: MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, pp. 76:1–76:3. ACM (2020)
13. Cabot, J., Burgueño, L., Clarisó, R., Daniel, G., Perianez-Pascual, J., Rodríguez-Echeverría, R.: Testing nlp-intensive bots: challenges and roadmap. In: 3rd International Workshop on Bots in Software Engineering (BotSE'21), vol. to appear (2021)
14. Ceri, S., Matera, M., Rizzo, F., Demaldé, V.: Designing data-intensive web applications for content accessibility using web marts. Commun. ACM **50**(4), 55–61 (2007)
15. Chittò, P., Báez, M., Daniel, F., Benatallah, B.: Automatic generation of chatbots for conversational web browsing. In: Conceptual Modeling—39th International Conference, ER 2020, Vienna, Austria, November 3–6, 2020, Proceedings, *Lecture Notes in Computer Science*, vol. 12400, pp. 239–249. Springer (2020)
16. Daniel, G., Cabot, J., Deruelle, L., Derras, M.: Multi-platform chatbot modeling and deployment with the jarvis framework. In:

Advanced Information Systems Engineering—31st International Conference, CAiSE 2019 Proceedings, pp. 177–193 (2019)
17. Daniel, G., Cabot, J., Deruelle, L., Derras, M.: Xatkit: a multimodal low-code chatbot development framework. IEEE Access **8**, 15332–15346 (2020)
18. Ed-Douibi, H., Izquierdo, J.L.C., Daniel, G., Cabot, J.: A model-based chatbot generation approach to converse with open data sources. In: Proceedings of the 21st International Conference on Web Engineering, to appear (2021)
19. Garrett, J.J.: Elements of User Experience. User-Centered Design for the Web and Beyond. Pearson Education, The (2010)
20. Gómez, J., Cachero, C., Pastor, O.: Conceptual modeling of device-independent web applications. IEEE Multim. **8**(2), 26–39 (2001)
21. Google: DialogFlow Website (2018). https://dialogflow.com/
22. Groenewegen, D.M., Hemel, Z., Kats, L.C.L., Visser, E.: Webdsl: a domain-specific language for dynamic web applications. In: Companion to the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, pp. 779–780. ACM (2008)
23. Huang, A., Pan, M., Zhang, T., Li, X.: Static extraction of IFML models for android apps. In: Proceedings of the 21st ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS, pp. 53–54. ACM (2018)
24. IBM: Watson Assistant Website (2018). URL: https://www.ibm.com/watson/ai-assistant/
25. Kim, M., Zimmermann, T., DeLine, R., Begel, A.: Data scientists in software teams: state of the art and challenges. IEEE Trans. Software Eng. **44**(11), 1024–1038 (2018)
26. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education (2008)
27. Klopfenstein, L.C., Delpriori, S., Malatini, S., Bogliolo, A.: The rise of bots: A survey of conversational interfaces, patterns, and paradigms. In: Proceedings of the 2017 Conference on Designing Interactive Systems, DIS, pp. 555–565. ACM (2017)
28. McTear, M.F.: Spoken dialogue technology: enabling the conversational user interface. ACM Comput. Surv. **34**(1), 90–169 (2002)
29. Meixner, G., Seissler, M., Breiner, K.: Model-driven useware engineering. Model-Driven Develop Adv User Interfaces Stud Comput Intell **340**, 1–26 (2011)
30. Mouelhi, T., Fleurey, F., Baudry, B., Le Traon, Y.: A model-based framework for security policy specification, deployment and testing. In: International Conference on Model Driven Engineering Languages and Systems, pp. 537–552. Springer (2008)
31. OMG: Interaction Flow Modeling Language (IFML) specification. Version 1.0 (2015). https://www.omg.org/spec/IFML/About-IFML/
32. OMG: Unified Modeling Language (UML) specification. Version 2.5.1 (2017). https://www.omg.org/spec/UML/About-UML/
33. Pan, M., Lu, Y., Pei, Y., Zhang, T., Zhai, J., Li, X.: Effective testing of android apps using extended IFML models. J. Syst. Softw. **159**,(2020)
34. Paternò, F., Santoro, C., Spano, L.D.: MARIA:a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput. Hum. Interact **16**(4), 19:1–19:30 (2009)
35. 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 - 21st International Conference, BPMDS 2020, 25th International Conference, EMMSAD 2020, Held at CAiSE 2020, *Lecture Notes in Business Information Processing*, vol. 387, pp. 257–265. Springer (2020)
36. Pérez-Soler, S., Guerra, E., de Lara, J.: Model-driven chatbot development. In: 39th Int. Conf. on Conceptual Modeling, ER, *LNCS*, vol. 12400, pp. 207–222. Springer (2020)
37. Rahimi, M., Guo, J.L.C., Kokaly, S., Chechik, M.: Toward requirements specification for machine-learned components. In: 27th IEEE International Requirements Engineering Conference Workshops, RE, pp. 241–244. IEEE (2019)
38. Raneburger, D., Popp, R., Kavaldjian, S., Kaindl, H., Falb, J.: Optimized GUI generation for small screens. Model-Driven Develop Adv User Interfaces StudComput Intell **340**, 107–122 (2011)
39. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. **25**(6), 5193–5254 (2020)
40. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic hypermedia application design with OOHDM. In: The Seventh ACM Conference on Hypertext, pp. 116–128. ACM (1996)
41. Soysal, E., Wang, J., Jiang, M., Wu, Y., Pakhomov, S., Liu, H., Xu, H.: Clamp-a toolkit for efficiently building customized clinical natural language processing pipelines. J Am Med Inf Assoc **25**(3), 331–336 (2018)
42. Trigueros, M.L., Preciado, J.C., Sánchez-Figueroa, F.: A method for model based design of rich internet application interactive user interfaces. In: Web Engineering, 7th International Conference, ICWE, *LNCS*, vol. 4607, pp. 226–241. Springer (2007)
43. Vanderdonckt, J.: A MDA-compliant environment for developing user interfaces of information systems. In: Advanced Information Systems Engineering, 17th International Conference, CAiSE, *LNCS*, vol. 3520, pp. 16–31. Springer (2005)
44. Vdovjak, R., Frasincar, F., Houben, G., Barna, P.: Engineering semantic web information systems in hera. J. Web Eng. **2**(1–2), 3–26 (2003)
45. Völkel, S.T., Schneegass, C., Eiband, M., Buschek, D.: What is "intelligent" in intelligent user interfaces?: a meta-analysis of 25 years of IUI. In: 25th International Conference on Intelligent User Interfaces, pp. 477–487. ACM (2020)

Elena Planas received the B.Sc., M.Sc. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya–BarcelonaTech (UPC). Since 2007, she combines her activity as a lecturer on software engineering courses with the research activity at the Universitat Oberta de Catalunya (UOC). Her research interests mainly focus on the area of model-driven engineering (MDE) and its application in software development. Some of her current research topics include the extension of modeling languages to cover new types of multi-modal intelligent UIs and their integration and interaction with the rest of the system; the development of verification techniques to assess the quality of software models; and the empirical research on the use of modeling tools in e-learning environments.

Gwendal Daniel received the Ph.D. degree with the AtlanMod Team, Ecole des Mines de Nantes, France, in 2017. He is currently a Postdoctoral Fellow with the SOM Research Lab, Internet Interdisciplinary Institute (IN3), a Research Center of the Universitat Oberta de Catalunya (UOC). His research interests include model-driven engineering, model persistence, query, transformation techniques, domain-specific languages, and applying model-based techniques for large-scale data applications. He received the Best Thesis Award from the GDR-GPL and the INFORSID Association, in 2018. Gwendal is also the co-founder of Xatkit, a low code platform for building smart chatbots.



Marco Brambilla is a full professor at Politecnico di Milano. He is active in research and innovation, at both industrial and academic levels. His research interests include data science, software modeling languages and design patterns, crowdsourcing, social media monitoring, and big data analysis. He has been visiting researcher at CISCO, San Josè, and UCSD, and visiting professor at Dauphine University, Paris. He is co-founder of three startups. He co-authored over 250 papers and books. He was awarded various best paper prizes and gave keynotes and speeches at many conferences and organisations. He runs research projects on data science and industrial projects on data-driven innovation and big data. He is the main author of the OMG standard IFML. He is editor of Journal of Web Engineering, and Advances in Human-Computer Interactions.

Jordi Cabot received the B.Sc. and Ph.D. degrees in computer science from the Universitat Politècnica de Catalunya–BarcelonaTech (UPC). He was a Leader of an INRIA and LINA Research Group at Ecole des Mines de Nantes, France, a Post-Doctoral Fellow with the University of Toronto, a Senior Lecturer with the Open University of Catalonia, and a Visiting Scholar with the Politecnico di Milano. He is currently an ICREA Research Professor at Internet Interdisciplinary Institute. His research interests include software and systems modeling, formal verification and the role AI can play in software development (and vice versa). He has published over 200 peer-reviewed conference and journal papers on these topics. Apart from his scientific publications, he writes and blogs about all these topics in several sites like modeling-languages.com and livablesoftware.com. He is also the co-founder and CEO of Xatkit, an open-source chatbot development framework.

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com