A Multiparadigm Approach to Integrate Gestures and Sound in the Modeling Framework

Vasco Amaral[1], Antonio Cicchetti[2], Romuald Deshayes[3]

[1] Universidade Nova de Lisboa, Portugal vasco.amaral@fct.unl.pt
[2] Malardalen Research and Technology Centre (MRTC), Vasteras, Sweden
antonio.cicchetti@mdh.se
[3] Software Engineering Lab, Université de Mons-Hainaut, Belgium
romuald.deshayes@umons.ac.be

Abstract. One of the essential means of supporting Human-Machine Interaction is a (software) language, exploited to input commands and receive corresponding outputs in a well-defined manner. In the past, language creation and customization used to be accessible to software developers only. But today, as software applications gain more ubiquity, these features tend to be more accessible to application users themselves. However, current language development techniques are still based on traditional concepts of human-machine interaction, i.e. manipulating text and/or diagrams by means of more or less sophisticated keypads (e.g. mouse and keyboard).
In this paper we propose to enhance the typical approach for dealing with language intensive applications by widening available human-machine interactions to multiple modalities, including sounds, gestures, and their combination. In particular, we adopt a Multi-Paradigm Modelling approach in which the forms of interaction can be specified by means of appropriate modelling techniques. The aim is to provide a more advanced human-machine interaction support for language intensive applications.

1 Introduction

The mean of supporting Human-Machine interaction are languages: a well-defined set of concepts that can be exploited by the user to compose more or less complex commands to be input to the computing device. Given the dramatic growth of software applications and their utilization in more and more complex scenarios, there has been a contemporary need to improve the form of interaction in order to reduce users' effort. Notably, in software development, it has been introduced different programming language generations, programming paradigms, and modelling techniques aiming at raising the level of abstraction at which the problem is faced. In other words, abstraction layers have been added to close the gap between machine language and domain-specific concepts, keeping them interconnected through automated mechanisms.

While the level of abstraction of domain concepts has remarkably evolved, the forms of interaction with the language itself indubitably did not. In particular, language expressions are mainly text-based or a combination of text and

diagrams. The underlying motivation is that, historically, keyboard and mouse have been exploited as standard input devices. In this respect, other forms of interaction like gestures and sound have been scarcely considered. In this paper we discuss the motivations underlying the need of enhanced forms of interaction and propose a solution to integrate gestures and sound in modeling frameworks. In particular, we define *language intensive* applicative domains the cases where either the language evolves rapidly, or language customizations are part of the core features of the application itself.

In order to better grasp the previously mentioned problem, we consider a sample case study in the Home Automation Domain, where a language has to be provided as supporting different automation facilities for a house, ranging from everyday life operations to maintenance and security. This is a typical language intensive scenario since there is a need to customize the language depending on the customer's building characteristics. Even more important, the language has to provide setting features enabling a customer to create users' profiles: notably, children may command TV and lights but they shall not access kitchen equipment. Likewise, the cleaning operator might have access to a limited amount of rooms and/or shall not be able to deactivate the alarm.

In the previous and other language intensive applicative domains it is inconceivable to force users to exploit the "usual" forms of interaction for at least two reasons: i) if they have to digit a command to switch on the lights they could use the light switch instead and hence would not invest money in these type of systems. Moreover, some users could be unable to exploit such interaction techniques, notably disabled, children, and so forth; ii) the home automation language should be easily customizable, without requiring programming skills. It is worth noting that language customization becomes a user's feature in our application domain, rather than a pure developer's facility. If we widen our reasoning to the general case, the arguments mentioned so far can be referred to as the need of facing accidental complexity. Whenever a new technology is proposed, it is of paramount importance to ensure that it introduces new features and/or enhances existing ones, without making it more complex to use, otherwise it would not be worth to be exploited.

Our solution is based on Multi-Paradigm Modelling (MPM) principles, i.e. every aspect of the system has to be appropriately modeled and specified, combining different points of view of the system being then possible to derive the concrete application. In this respect, we propose to precisely specify both the actions and the forms of language interactions, in particular gestures and sound, by means of models. In this way, flexible interaction modes with a language are possible as well as languages accepting new input modalities such as sounds and gestures.

The remaining of the paper is organized as follows: sec. 2 discusses the state-of-the-art; sec. 3 presents the communication between a human and a machine through a case study related to home automation; sec. 4 introduces an interaction metamodel and discusses how it can be used to generate advanced concrete syntaxes relying on multiple modalities; finally, sec. 5 concludes.

2 State of the Art

This section describes basic concepts and state-of-the-art techniques typically exploited in sound/speech and gesture recognition together with their combinations created to provide advanced forms of interaction. Our aim is to illustrate the set of concepts usually faced in this domain and hence to elicit the requirements for the interaction language we will introduce in Section 4.

Sound and speech recognition

Sound recognition is usually used for command-like actions; a word has to be recognized before the corresponding action can be triggered. With the Vocal Joystick [1] it is possible to use acoustic phonetic parameters to continuously control tasks. For example, in a WIMP (Windows, Icons, Menus, Pointing device) application, the type of vowel can be used to give a direction to the mouse cursor, and the loudness can be used to control its velocity.

In the context of environmental sounds, [2, 3] proposed different techniques based on Support Vector Machines and Hidden Markov Models to detect and classify acoustic events such as foot steps, a moving chair or human cough. Detecting these different events help to better understand the human and social activities in smart-room environments. Moreover, an early detection of non-speech sounds can help to improve the robustness of automatic speech recognition algorithms.

Gesture recognition

Typically, gesture recognition systems resort to various hardware devices such as data glove or markers [4], but more recent hardware such as the Kinect or other 3D sensors enable unconstrained gestural interaction [5]. According to a survey of gestural interaction [6], gestures can be of 3 types :

- hand and arm gestures: recognition of hand poses or signs (such as recognition of sign language);
- head and face gestures: shaking head, direction of eye gaze, opening the mouth to speak, happiness, fear, etc;
- body gestures: tracking movements of two people interacting, analyzing movement of a dancer, or body poses for athletic training.

The most widely used techniques for dynamic gestural recognition usually involve hidden Markov models [7], particle filtering [8] or finite state machines [9].

Multimodal systems

The "Put-that-there" [10] system, developed in the 80's, is considered to be the origin of human-computer interaction regarding the use of voice and sound. Vocal commands such as "delete this elements" while pointing at one object

displayed on the screen can be correctly processed by the system. According to [11], using multiple modalities, such as sound and gestures, helps to make the system more robust and maintainable. They also proposed to split audio sounds in two categories: human speech and environmental sounds.

Multimodal interfaces involving speech and gestures have been widely used for text input, where gestures are usually used to choose between multiple possible utterances or correct recognition errors [12, 13]. Some other techniques propose to use gestures on a touchscreen device in addition to speech recognition to correct recognition errors [14]. Both modalities can also be used in an asynchronous way to disambiguate between the possible utterances [15].

More recently, the SpeeG system [16] has been proposed. It is a multimodal interface for text input and is based on the Kinect sensor, a speech recognizer and the Dasher [17] user interface. The contribution lies in the fact that, unlike the aforementioned techniques, the user can perform speech correction in real-time, while speaking, instead of doing it in a post processing fashion.

Human-computer interaction modeling

In the literature, many modeling techniques have been used to represent interaction with traditional WIMP user interfaces. For example, statecharts have been dedicated to the specification and design of new interaction objects or widgets [18].

Targetting virtual reality environment, Flownets [19] is a modeling tool, relying on high-level Petri nets, based on a combination of discrete and continuous behavior to specify the interaction with virtual environments. Also based on high-level Petri nets for dynamic aspects and an object oriented framework, the Interactive Cooperative Objects (ICO) formalism [20] has been used to model WIMP interfaces as well as multimodal interactions in virtual environments. In [4], a virtual chess game was developed in which the user can use a data glove to manipulate virtual chess pieces. In [21], ICO has been used to create a framework for describing gestural interaction with 3D objects has been proposed.

Providing a UML based generic framework for modeling interaction modalities such as speech or gestures enables software engineers to easily integrate multimodal HCI in their applications. That's the point defended by [22]. In their work, the authors propose a metamodel which focuses on the aspects of an abstract modality. They distinguish between simple and complex modalities. The first one represents a primitive form of interaction while the second integrates other modalities and uses them simultaneously. With the reference point being the computer, input and output modalities are defined as a specification of simple modality. Input modalities can be event-based (e.g. performing a gesture or sending a vocal command) or streaming based (e.g. drawing a circle or inputting text using speech recognition) and output modalities are used to provide static (e.g. a picture) or dynamic (e.g. speech) information to the user.

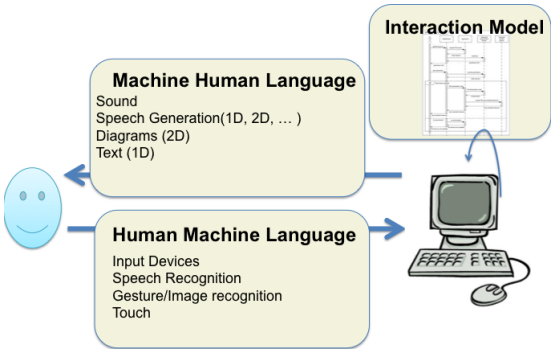

Fig. 1. A more advanced support of Human-Machine Interaction

3 Human-Machine Communication

As discussed so far, different techniques supporting more advanced forms of interaction between humans and machines have already been proposed. Nonetheless, their exploitation in current software languages has been noticeably limited. This work proposes to widen the modalities of human-machine interaction as depicted in Fig. 1. In general, a human could input information by means of speech, gestures, texts, drawings, and so forth. The admitted ways of interaction are defined in an interaction model, which also maps human inputs into corresponding machine readable formats. Once the machine has completed its work, it outputs the results to the user through sounds, diagrams, texts, etc.; also in this case the interaction model prescribes how performed computations should be rendered to a human comprehensible format.

A software language supports the communication between humans and machines by providing a set of well-defined concepts that typically abstract real-life concepts. Hence, software language engineering involves the definition of three main aspects: i) the internal representation of the selected concepts, understandable by the machine and typically referred to as *abstract syntax*; ii) how the concepts are rendered to the users in order to close the gap with the applicative domain, called *concrete syntax*; iii) how the concepts can be interpreted to get/provide domain-specific information, referred to as *semantics*. Practically, a metamodel serves as a base for defining the structural arrangement of concepts and their relationships (abstract syntax), the concrete syntax is "hooked" on appropriate groups of its elements, and the semantics is generically defined as computations over elements.

In order to better understand the role of these three aspects, Fig. 2 and 1 illustrate excerpts of the abstract and concrete syntaxes, respectively, of a sample Home Automation DSL. In particular, a home automation system manages a `House` (see Fig. 2 right-hand side) that can have several rooms under domotic control (i.e. `Room` and `DomoticControl` elements in the metamodel, respectively).

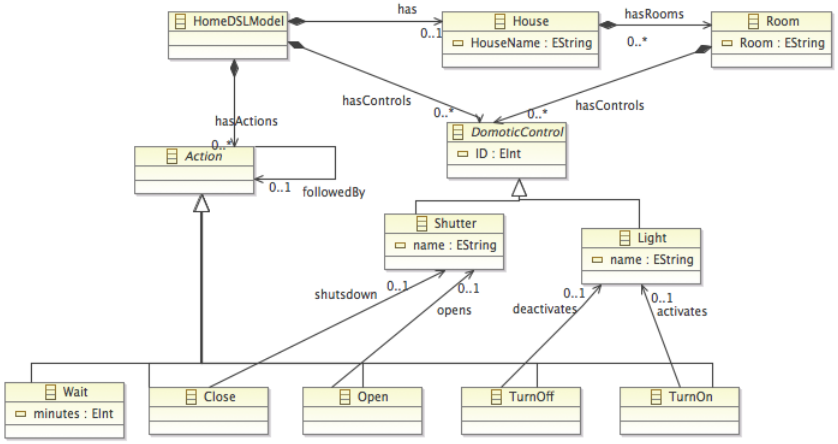

Fig. 2. Metamodel (abstract syntax) of a simple illustration DSL for Home Automation

The control is composed by several devices and actions that can be performed with them. Notably, a `Light` can be turned on and off, while a `Shutter` can be opened and closed.

It is easy to notice that the abstract syntax representation would not be user-friendly in general, hence a corresponding concrete syntax can be defined in order to provide the user with easy ways of interaction. In particular, Prog. 1 shows an excerpt of the concrete syntax definition for the home automation DSL using the Eugenia tool[4]. The script prescribes to depict a `House` element as a graph node showing the rooms defined for the house taken into account.

Prog. 1 Concrete Syntax mapping of the DSL for Home Automation with Eugenia

```
@gmf.node(label="HouseName", color="255,150,150", style="dash")
class House {
  @gmf.compartment(foo="bar")
  val Room[*] hasRooms;
  attr String HouseName;
}
```

Despite the remarkable improvements in language usability thanks to the addition of a concrete syntax, the malleability of interaction modalities provided by Eugenia and other tools (e.g. GMF[5]) is limited to the standard typing and/or

[4] http://www.eclipse.org/epsilon/doc/eugenia/
[5] http://www.eclipse.org/modeling/gmp/

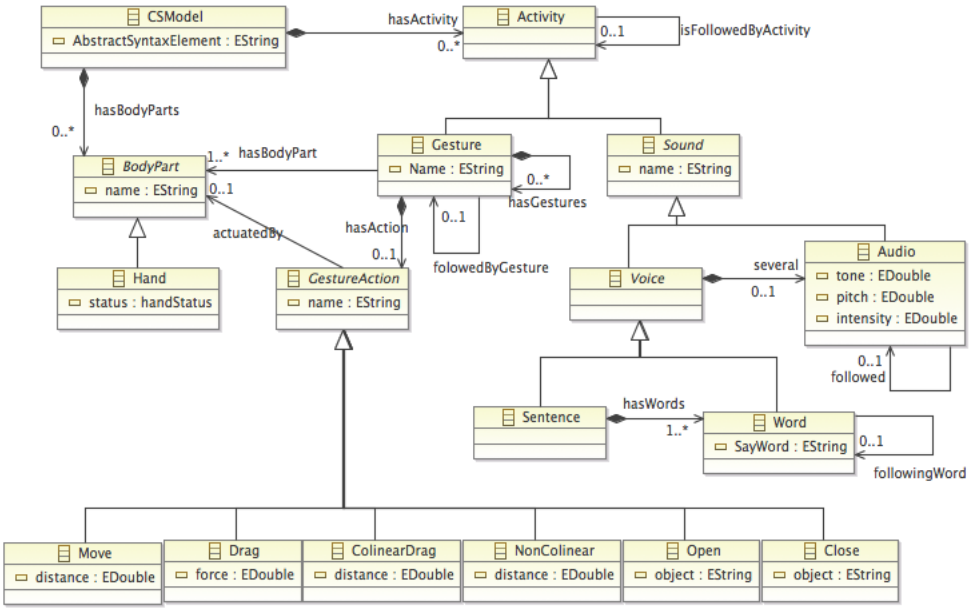

Fig. 3. A possible language to define enhanced concrete syntaxes for DSL

drawing graphs. Such a limitation becomes evident when needing to provide users with extended ways of interaction, notably defining concrete syntaxes as sounds and/or gestures. For instance, a desirable concrete syntax for the home automation metamodel depicted in Fig. 2 would define voice commands for turning lights on and off, or alternatively prescribe certain gestures to do the same operations.

By embracing the MPM vision, which prescribes to define any aspect of the modeling activity as a model, next Section introduces a language for defining advanced human-machine interactions. In turn, such a language can be combined with abstract syntax specifications to provide DSLs with enhanced concrete syntaxes.

4 Interaction Modeling

The proposed language tailored to enhanced human-machine interaction concrete syntax definition is shown in Fig. 3. In particular, it depicts the metamodel to define advanced concrete syntaxes, encompassing sound and gestures, while the usual texts writing and diagrams drawing are treated as particular forms of gestures. Going deeper, elements of the abstract syntax can be linked to (sequences of) activities (see `Activity` on the bottom-left part of Fig. 3). An activity, in turn, can be classified as a `Gesture` or a `Sound`.

Fig. 4. A possible instance model defining the Concrete Syntax of a concept

Regarding gestures, the language supports the definition of different types of primitive actions typically exploited in gesture recognition applications. A `Gesture` is linked to the `BodyPart` that is expected to perform the gesture. This way we can distinguish between performed actions, for example, by a hand or a head. `Move` is the simplest action a body part can perform, it is triggered for each displacement of the body part. Dragging (`Drag`) can only be triggered by a hand a represents a displacement of a closed hand. `ColinearDrag` and `NonColinearDrag` represent a movement of both hands going in either the same or opposite directions while being colinear or not colinear, respectively. `Open` and `Close` are triggered when the user opens or closes the hand.

Sounds can be separated in two categories: i) `Voice` represents human speech, which is composed of `Sentences` and/or `Words`. ii) `Audio` that relates to all non speech sounds that can be encountered, such as knocking on a door, a guitar chord or even someone screaming. As it is very generic, it is characterized by fundamental aspects as `tone`, `pitch`, and `intensity`.

Complex activities can be created by combining multiple utterances of sound and gestures. For example one could define an activity to close a shutter by closing the left hand and dragging it from top to bottom while pointing at the shutter and saying "close shutters".

In order to better understand the usage of the proposed language, Fig. 4 shows a simple example defining the available concrete syntaxes for specifying the turning the lights on command. In particular, it is possible to wave the right hand, first left and then right to switch on light 1 (see the upper part of the picture). Alternatively, it is possible to give a voice command made up of the sound sequence "Turn Light 1 On", as depicted in the bottom part of the figure.

It is worth noting that, given the purpose of the provided interaction forms, the system can be taught to recognize particular patterns as sounds, speeches,

gestures, or their combinations. In this respect, the technical problems related to recognition can be alleviated. Even more important, the teaching process discloses the possibility to extend the concrete syntax of the language itself, since additional multimodal commands can be introduced as alternative ways of interaction.

5 Conclusions

The ubiquity of software applications is widening the modeling possibilities of end users who may need to define their own languages. In this paper, we defined these contexts as language-intensive applications given the evolutionary pressure the languages are subject to. In this respect, we illustrated the needs of having enhanced ways of supporting human-machine interactions and demonstrated them by means of a small home automation example. We noticed that in general concrete syntaxes usually refer to traditional texts writing and diagrams drawing, while more complex forms of interaction are largely neglected. Therefore, we proposed to extend the current concrete syntax definition approaches by adding sounds and gestures, but also possibilities to compose them with traditional interaction modalities. In this respect, by adhering to the MPM methodology we defined an appropriate modeling language for illustrating concrete syntaxes that can be exploited later on to generate corresponding support for implementing the specified interaction modalities.

As next steps we plan to extend the Eugenia concrete syntax engine in order to be able to automatically generate the support for the extended human-machine interactions declared through the proposed language. This phase will also help in the validation of the proposed concrete syntax metamodel shown in Fig. 3. In particular, we aim at verifying the adequacy of the expressive power provided by the language and extend it with additional interaction means.

This work constitutes the base to build-up advanced modeling tools relying on enhanced forms of interaction. Such improvements could be remarkably important to widen tools accessibility to disabled developers as well as to reduce the accidental complexity of dealing with big models.

References

1. J. Bilmes, X. Li, J. Malkin, K. Kilanski, R. Wright, K. Kirchhoff, A. Subramanya, S. Harada, J. Landay, P. Dowden, and H. Chizeck, "The vocal joystick: A voice-based human-computer interface for individuals with motor impairments," in *HLT/EMNLP*. The Association for Computational Linguistics, 2005.
2. A. Temko, R. Malkin, C. Zieger, D. Macho, and C. Nadeu, "Acoustic event detection and classification in smart-room environments: Evaluation of chil project systems," *Cough*, vol. 65, p. 6, 2006.
3. S. Nakamura, K. Hiyane, F. Asano, T. Nishiura, and T. Yamada, "Acoustical sound database in real environments for sound scene understanding and hands-free speech recognition," in *LREC*. European Language Resources Association, 2000.

4. D. Navarre, P. Palanque, R. Bastide, A. Schyn, M. Winckler, L. Nedel, and C. Freitas, "A formal description of multimodal interaction techniques for immersive virtual reality applications," in *INTERACT*, ser. Lecture Notes in Computer Science, M. F. Costabile and F. Paternò, Eds., vol. 3585. Springer, 2005, pp. 170–183.
5. Z. Ren, J. Meng, J. Yuan, and Z. Zhang, "Robust hand gesture recognition with kinect sensor," in *ACM Multimedia*, 2011, pp. 759–760.
6. S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Trans. on Systems, Man and Cybernetics - part C*, vol. 37, no. 3, pp. 311–324, 2007.
7. J. Yamato, J. Ohya, and K. Ishii, "Recognizing human action in time-sequential images using hidden Markov model," in *Proceed. IEEE Conf. Computer Vision and Pattern Recognition*, 1992, pp. 379–385.
8. M. Isard and A. Blake, "Condensation - conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.
9. P. Hong, T. S. Huang, and M. Turk, "Gesture modeling and recognition using finite state machines," in *FG*. IEEE Computer Society, 2000, pp. 410–415.
10. R. Bolt, "Put-that-there: Voice and gesture at the graphics interface," in *Proceed. 7th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '80. New York, NY, USA: ACM, 1980, pp. 262–270.
11. B. Demiroz, I. Ar, A. Ronzhin, A. Coban, H. Yalcn, A. Karpov, and L. Akarun, "Multimodal assisted living environment," in *eNTERFACE 2011, The Summer Workshop on Multimodal Interfaces*, 2011.
12. N. Osawa and Y. Y. Sugimoto, "Multimodal text input in an immersive environment," in *ICAT 2002, 12th International Conference on Articial Reality and Telexistence*, 2002, pp. 85–92.
13. K. Vertanen, "Efficient computer interfaces using continuous gestures, language models, and speech," http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-627.pdf, Computer Laboratory, University of Cambridge, Tech. Rep. UCAM-CL-TR-627, 2005.
14. D. Huggins-Daines and A. I. Rudnicky, "Interactive asr error correction for touchscreen devices," in *ACL (Demo Papers)*. The Association for Computer Linguistics, 2008, pp. 17–19.
15. P. O. Kristensson and K. Vertanen, "Asynchronous multimodal text entry using speech and gesture keyboards," in *INTERSPEECH*. ISCA, 2011, pp. 581–584.
16. L. Hoste, B. Dumas, and B. Signer, "Speeg: a multimodal speech- and gesture-based text input solution," in *AVI*, G. Tortora, S. Levialdi, and M. Tucci, Eds. ACM, 2012, pp. 156–163.
17. D. J. Ward, A. F. Blackwell, and D. J. C. MacKay, "Dasher - a data entry interface using continuous gestures and language models," in *UIST*, 2000, pp. 129–137.
18. D. A. Carr, "Specification of interface interaction objects," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '94. New York, NY, USA: ACM, 1994, pp. 372–378. [Online]. Available: http://doi.acm.org/10.1145/191666.191793
19. S. Smith and D. Duke, "Virtual environments as hybrid systems," in *Proceedings of Eurographics UK 17th Annual Conference (EG-UK99)*, E. U. K. Chapter, Ed., United Kingdom, 1999.
20. P. A. Palanque and R. Bastide, "Petri net based design of user-driven interfaces using the interactive cooperative objects formalism," in *DSV-IS*, 1994, pp. 383–400.
21. R. Deshayes, T. Mens, and P. Palanque, "A generic framework for executable gestural interaction models," in *Proc. VL/HCC*, 2013.
22. Z. Obrenovic and D. Starcevic, "Modeling multimodal human-computer interaction," *IEEE Computer*, vol. 37, no. 9, pp. 65–72, 2004.