Assessing the Support Provided by a Toolkit for Rapid Prototyping of Multimodal Systems

Fredy Cuenca, Davy Vanacken, Karin Coninx, Kris Luyten
Hasselt University - tUL - iMinds
Expertise Centre for Digital Media, Diepenbeek, Belgium
{fredy.cuencalucero,davy.vanacken,karin.coninx,kris.luyten}@uhasselt.be

ABSTRACT
Choosing an appropriate toolkit for creating a multimodal interface is a cumbersome task. Several specialized toolkits include fusion and fission engines that allow developers to combine and decompose modalities to capture multimodal input and provide multimodal output. Unfortunately, the extent to which these toolkits can facilitate the creation of a multimodal interface is hard or impossible to estimate, due to the absence of a scale where the toolkit's capabilities can be measured on. In this paper, we propose a measurement scale, which allows the assessment of specialized toolkits without need for time-consuming testing or source code analysis. This scale is used to measure and compare the capabilities of three toolkits: CoGenIVE, HephaisTK and ICon.

Author Keywords
Multimodal systems; User interface toolkits; Visual languages; Domain specific languages;

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
For a traditional WIMP system, the detection of a single user event is enough to identify the user's intent. For instance, a click on a button *Accept* or *Cancel* of a GUI is enough to realize whether a user wants to process or close a form respectively. For the case of a multimodal system, its users are allowed to dissociate a command so that it can be conveyed through multiple modalities. For example, the users can simultaneously utilize speech and pointing to issue commands, such that the action to be executed on an object is indicated by the speech input whereas the object itself is pointed out. Thus, the identification of the user's intent is not that simple since it requires the evaluation of multiple events in order to decode what the user is requesting. A multimodal system is a computer system capable of collecting the information provided by a user through multiple input modes, integrating these inputs in order to interpret the user's intent, and responding to him/her via multiple outputs. Some input modes that can be used to enter information into a multimodal system are speech, touch, hand gestures, handwriting or sketching. Output modes can include images, audio, synthesized voice, video or haptics.
The development of a multimodal system is time-consuming, and therefore expensive. It involves the creation and iterative adaptation of prototypes. Therefore, the development phase of a multimodal system can be shortened by facilitating the creation and modification of prototypes, which is precisely the purpose of the toolkits under study. In the remainder of this work, these toolkits will be referred to as toolkits for rapid prototyping of multimodal systems.
Some existing toolkits for rapid prototyping of multimodal systems are ICon [5], Squidy [13], CoGenIVE [4], HephaisTK [6] and PetShop [9]. They are rather different one from another, since they provide different features, target different domains, use different programming paradigms and/or expect different skills from their users. Some aspects of these toolkits have already been assessed. De Boeck et al. [3] evaluated the abstraction, difuseness, role-expressiveness, viscosity and premature commitment of the visual languages of two toolkits. Later, Dumas et al. [7] used the architecture traits, reusability easiness and other characteristics as criteria for assessing a set of toolkits. Even though the results of these evaluations deepen our understanding of rapid prototyping toolkits, their practical application is not always obvious.
From a pragmatic viewpoint, the evaluation of a toolkit for rapid prototyping of multimodal systems leads us to the concrete question 'To what extent is the use of this toolkit going to facilitate the implementation of a multimodal prototype?'. Unfortunately, the absence of a scale for measuring the functionalities incorporated in a specialized toolkit prevents us from accurate answers. Such measurement scales will be proposed in this work and use to evaluate the support provided by CoGenIVE, HephaisTK, and ICon for the implementation of prototypes.

ARCHITECTURE OF A MULTIMODAL SYSTEM
The parts that comprise a multimodal system, and their interrelations are shown in Figure 1.
In this architecture, user inputs are recognized by a group of specialized software components called recognizers. Each recognizer is continuously sensing and decoding the infor-

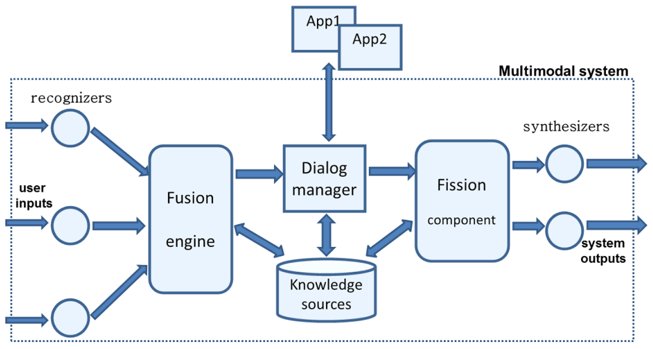

Figure 1. Architecture of a multimodal system

mation provided by the user via the modality it is intended to sense. Some examples of these components are gesture, handwriting and voice recognizers.
Whenever a recognizer has interpreted a stream of user inputs, it informs the fusion engine, which is in charge of merging the information provided by all the recognizers in order to interpret the user's request.
Once the dialog manager is notified of the user's request, it must decide how to handle it. Since the same input may result in different responses, depending on the context, the dialog manager must track the status of the human-machine dialog so that user requests can be addressed correctly.
After the dialog manager has decided on the response to be sent, it delegates this task to the fission component. This must then choose the synthesizers (computer programs that control rendering devices) that are best suited for the situation. The generation and coordination of multimedia output is the responsability of the fission component.
Finally, the response to a user command may depend on the user profile (e.g. gender, age, preferences, etc.), on the domain of the problem, or on the history of the human-machine dialog. All the relevant information needed by the system is available in data storages called knowledge sources.
When using a toolkit for rapid prototyping of multimodal systems, its users do not have to implement all the aforementioned functionalities from scratch. Rather, they can use its visual language to invoke some functions that are pre-programmed in its framework, as shown below.

TOOLKIT FOR RAPID PROTOTYPING OF MULTIMODAL SYSTEMS

A toolkit for rapid prototyping of multimodal systems includes a framework and a graphical editor. It aims to enhance an external application, herein called client application, with multimodal capabilities. On the one hand, the client application is developed by means of a textual programming language and with no support from the toolkit. It must implement the particular functionalities of the intended prototype. On the other hand, the graphical editor allows the depiction of visual models that will be interpreted and executed by the framework. These visual models specify the tasks the prototype must perform during its interaction with the end user. Some of these tasks are present in a wide variety of multimodal systems (e.g. speech recognition or tracking of system state) and are already pre-programmed in the framework. Other tasks are application-specific and have to be carried out by the subroutines of the client application.

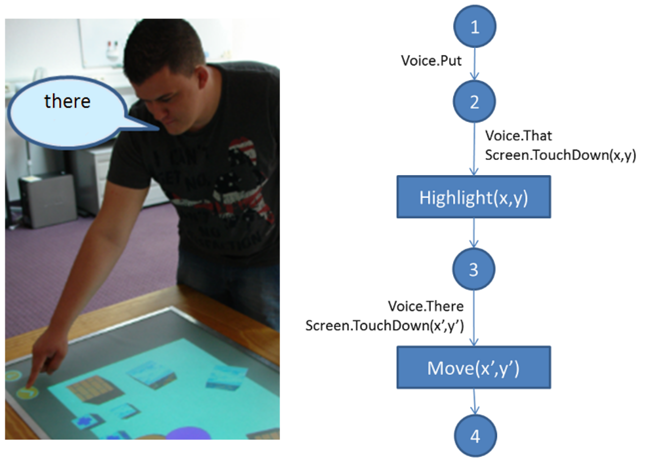

Figure 2. Left. End user interacting with a multimodal system. Right. Visual model used for specifying human-machine interaction.

Consider a multimodal prototype that supports the *put-that-there* interaction technique [1]. This prototype displays a series of objects on a touch-sensitive screen, and its user can move any of these objects by using speech and pointing (Figure 2). The user must utter the sentence 'put that there' to move an object from its original position to a new one. In order for the system to correctly interpret the meanings of the utterances 'that' and 'there', the user must point out an object and any arbitrary position while pronouncing these words respectively.
The layout of the GUI and the algorithms for highlighting and moving an object must be implemented in a client application. This application does not need to detect voice commands or pointing events. Nor does it have to verify the temporal co-occurrence of the speech input 'that' (or 'there') and the touch on the screen. These functionalities can be delegated to the framework through a visual model like the one shown on the right side of Figure 2. It specifies that the occurrence of the speech input 'put' followed by the co-occurrence of the speech input 'that' and a touch on the screen will cause the execution of the subroutine *Highlight*. Afterwards, the co-ocurrence of the speech input 'there' and a touch on the screen will trigger the execution of the subroutine *Move*, which will have to change the position of the currently selected (highlighted) object. *Highlight* and *Move* have to be programmed in the client application.

MEASURING THE SUPPORT OF A TOOLKIT TO THE IMPLEMENTATION OF MULTIMODAL PROTOTYPES

Through a visual model, users can delegate some tasks to the framework of a toolkit, as illustrated in the previous section. We now want to identify and classify these tasks in accordance to the software components (recognizers, fusion engine, etc.) that are in charge of their execution.
The study of several toolkits shows that they all incorporate software for detecting the inputs coming from a myriad of

hardware devices. Since this is the responsability of the recognizers, it can be claimed that the use of a toolkit can release its users from implementing the recognizers of a multimodal prototype. Another point in common is that none of the studied toolkits supports the implementation of synthesizers or knowledge sources. Thus, their users have to include software for synthesis of modalities in their client applications, and to create and fill the data storages containing the information needed by the prototype. However, the support offered for the implementation of the fusion engine, dialog manager and fission component varies with each toolkit.

Scale for measuring toolkit's support

We propose to map the support provided by a toolkit to the set of components whose implementation can be facilitated through its use. For instance, the support of a toolkit T will be $\{recognizers, fusion\ engine\}$ if the functionality in charge of both components can be delegated to the framework of T through the use of its visual language. Then, the set of all the possible combinations of components is the scale of measurement we are proposing. For the sake of formality, let C be the set of components shown in Figure 1, the scale on which the support of a toolkit will be measured on is the power set 2^C. Even though the nature of this scale is qualitative, it will still lead to more precise assessments of a toolkit's capabilities, which is of interest for its potential users.

The use of the proposed measurement scale requires finding out whether some functionalities of the fusion engine, dialog manager or fission component are incorporated in a toolkit, and available to be invoked by its users through the depiction of visual models. Indications to create such awareness are given below.

The detection of a user's request, which is a task of the ***fusion engine****, can be delegated to the framework of a toolkit if its visual language allows the specification of composite events.* A composite event is a set of events and the temporal constraints among them. It occurs whenever its constituent events are detected in a predefined order. By including composite events in the specification of a human-machine dialog, the user exploits the framework's capacity to evaluate streams of events, seeking for those meaningful patterns that are of interest for the client application to handle.

Managing context-dependent human-machine dialogs entails identifying the current state of the prototype throughout its interaction with the end user. *The management of context-dependent human-machine dialogs, which is the responsability of the* ***dialog manager****, can be supported by the framework of a toolkit if its visual language allows representing the states the prototype may ever be in.* Without using a toolkit, tracking the state of a multimodal prototype would imply the maintenance of global variables across different event handlers. Furthermore, choosing the subroutines that will handle a user's request would imply the implementation of complex convoluted logic, i.e. sets of nested if-else statements, involving the aforementioned global variables. By using an appropiate toolkit, users can release their client applications from this spaghetti code, entailing the creation of easy-to-maintain client applications.

The generation and coordination of multiple outputs, which is a task of the ***fission component****, can be delegated to the framework of a toolkit if its visual language offers constructs for concurrency and synchronization.* Concurrency is required to convey the returning message through multiple outputs, and synchronization is required to keep these outputs coordinated at every moment. For instance, a multimodal system displaying an animated character capable of talking must concurrently activate a display manager and a speech synthesizer. Additionally, in order to display the lips of the animated character such that they can always be in accordance with its speech [12], both outputs have to be constantly synchronized.

Scale for measuring toolkit's fusion, dialog management and fission capabilities

The preceding subsection proposed measuring the support provided by a toolkit in terms of the components of a multimodal system. In addition, it gave us indications to realize whether the fusion of inputs, the human-machine dialog management or the fission of a returning message can be handled by the framework of a toolkit. This subsection proposes additional metrics to increase the precision of toolkit assessment.

For a toolkit supporting the implementation of a fusion engine, we can identify the type of fusion it can support. According to the **CASE classification space** [10], a system can fuse data that is conveyed sequentially or simultaneously. The fusion of sequential (simultaneous) data allows identifying multimodal commands issued through consecutive (parallel) user actions. The CASE space can also be used to obtain a more precise gauge of the toolkit's fission capabilities. Indeed, the fission ability of a toolkit can be measured in terms of whether the toolkit can render a returning message through consecutive and/or parallel outputs.

For a toolkit capable of handling context-dependent human-machine dialogs, it is pertinent to detail whether these dialogs can involve complementary, assigned, redundant or equivalent modalities. Formally speaking, we can use the **CARE properties** [2] for providing more precise assessments of the toolkit's dialog management capabilities.

Unlike the previous subsection, we cannot give generic indications about how to infer the CASE and CARE properties of a toolkit from its visual language. The reason is that the language constructs required to exploit these properties vary from toolkit to toolkit. The identification of the type of fusion, dialog management and fission provided by a toolkit has to be done on an ad hoc basis.

ASSESSING COGENIVE, HEPHAISTK AND ICON

The diagrams depicted with the graphical editor of a toolkit for rapid prototyping of multimodal systems are variations of some well-known model. For the studied toolkits, these models are state diagrams and block diagrams.

State diagrams

State diagrams are graphs that can be utilized to model the interaction between a system and its end user. In this case, the nodes represent the states the system may ever be in, and the arcs represent its state transitions. Every arc of a state diagram can hold two annotations. One annotation is intended to indicate the event that makes the system changes its state, and the other to specify the subroutine(s) the system must execute during this transition. The models created with the editors of CoGenIVE and HephaisTK are variants of state diagrams.

CoGenIVE

The model shown on the left side of Figure 3 was depicted with the graphical editor of CoGenIVE [11, 4]. It specifies the behavior of a prototype implementing the *put-that-there* interaction technique described above.

This model shows that the prototype can be in four states, which are depicted as circles and labeled *Start*, *Put*, *Put-That* and *Put-That-There*. Once up and running, the prototype is in the state *Start*. Thereupon, the user commands will make it change its state. For instance, when the prototype is in the state *Put*, it will ignore every command except for the selection of the object to be moved, which is characterized by the co-occurrence of the speech input 'that' and a mouse click, i.e. by the occurrence of the composite event *Voice.That & Mouse.ButtonPressed*. As a response to this user command, the prototype will invoke some subroutines (represented as the rectangles labelled *CollisionWithPointer* and *SelectObject*) that will lead to the identification of the selected object. After performing these actions, the system will change its state to a new one labelled *Put-That*.

The possibility to include composite events in a CoGenIVE model permit us to delegate the **fusion of inputs** to the CoGenIVE's framework. In CoGenIVE, a composite event is a label annotated in an arrow of a visual model. It specifies those events whose co-occurrence must be detected by CoGenIVE's framework. This detection indicates that the end user has requested a service to the prototype.

CoGenIVE's editor allows us to place circles to represent each potential state of the prototype. Thus, the **management of context-dependent human-machine dialogs** is more easily implemented when using CoGenIVE. Its users must only program those subroutines implementing the particular functionalities of the intended prototype. Then, the CoGenIVE's framework will decide which of these subroutines must be executed and when this is the case.

As to the **fission component**, its functionality cannot be delegated to CoGenIVE's framework. This is due to the impossibility of specifying concurrency with the CoGenIVE's visual language, i.e. CoGenIVE's framework can only execute one subroutine per time. This limitation stems from the fact that state diagrams only experience one transition at a time and thus, only one subroutine can be executed in a given moment. Of course, one subroutine can be programmed so that it can handle concurrent computation but this would put all the burden on the programmer instead of on the framework.

At a more detailed level, we observe that CoGenIVE can fuse information coming from modalities used in parallel as seen in Figure 3. A sequence of events can also be interpreted as a multimodal command [4]. Finally, CoGenIVE's framework can handle human-machine dialogs exhibiting the four CARE properties. In view of space limitation, we cannot elaborate upon this point, but interested readers can refer to [11].

HephaisTK

The concise model of the right side of Figure 3, depicted with the editor of HephaisTK [6], specifies the behavior of a prototype implementing the *put-that-there* interaction technique. The prototype will initially be at state *Start* awaiting for the sequence of events whose detection will cause the movement of an object. This movement is executed by the subroutine $put\_that\_there\_action$ whereas the stream of events that will cause the execution of this subprogram is represented as a set of nested rectangles.

HephaisTK's notation allow four different types of rectangles to declare composite events [6]. Each type of rectangle indicates the temporal constraints among the events annotated within it. Thus, a wide variety of composite events can be defined by nesting these four types of containers. For instance, the yellow rectangles shown in the aforementioned figure mean simultaneous complementarity whereas the white one means sequential complementarity. Therefore, the model specifies that the sequential detection of the voice command 'put' followed by the co-occurrence of the voice command 'that' and a mouse click, and by the co-occurrence of the voice command 'there' and a mouse click will cause the execution of the subroutine $put\_that\_there\_action$ (implemented at the client side).

As mentioned above, the HephaisTK's editor allows representing composite events by nesting different types of rectangles. Thus, some functionalities of the **fusion engine** can be delegated to the HephaisTK's framework in benefit of its users. More specifically, users release their client applications from examining streams of events. Rather, it is the responsability of HephaisTK's framework to seek for those meaningful patterns of events specified as composite events.

Implementing the **management of context-dependent human-machine dialogs** can be facilitated when using HephaisTK. The states and subroutines to be called by HephaisTK's framework can be specified by means of a visual model. Each state is depicted as a circle, and the subroutines are annotated next to a zigzagged arrow. Models like the one shown in Figure 3, contain enough information so that HephaisTK's framework can always choose the subroutine that will correctly handle a multimodal command.

As regards the **fission component**, it has to be implemented at the client side with no support from HephaisTK. As mentioned in the previous subsection, this is due to the limitation of state diagrams to model concurrency.

At a more detailed level, HephaisTK can fuse information conveyed through sequential or simultaneous user actions. Finally, the appropiate use of the HephaisTK's visual language leads to the creation of prototypes supporting the four CARE properties[6].

Block diagrams

A block diagram is a set of blocks connected by directed arcs. It represents the transformations suffered by the data that flows within a system. Whereas the arcs can be seen as the channels through which the data flows, the blocks can be thought of as entities performing operations on the data that flows through them. The data enters into a multimodal prototype whenever its user issues some recognizable command. ICon models resemble block diagrams.

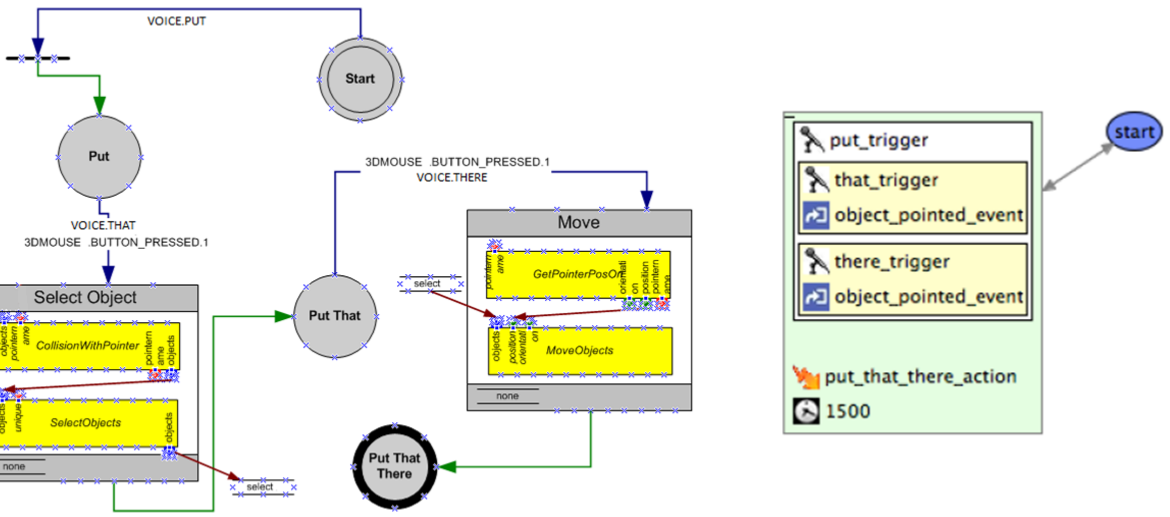

Figure 3. Specification of the put-that-there interaction technique in CoGenIVE (left) and HephaisTK (right) [8].

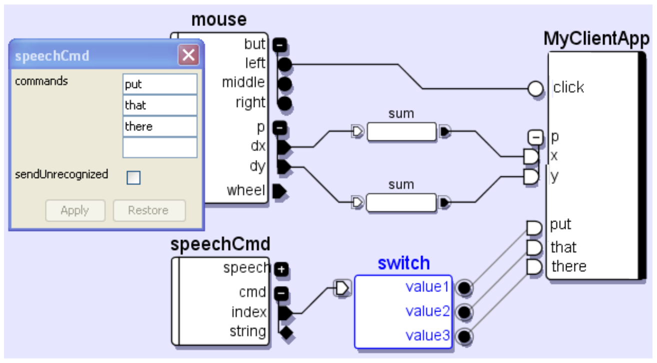

Figure 4. Specification of the put-that-there interaction technique in ICon.

ICon

Figure 4 shows how a client application called *MyClientApp* can be enhanced with speech recognition by means of a ICon's model. Instead of listening to system events, the client application expects to be notified, by the framework of ICon [5], about the user activities: clicking the mouse or uttering a voice command. The nodes labelled as *speechCmd* and *mouse* represent the recognizers that sense the voice and pointing commands issued by the end user. The data provided by these recognizers is then transformed by the nodes *switch* and *sum* before being sent to the client application. Nodes *sum* transform delta values (dx, dy) into cursor locations (x, y). The node *switch* activates one flag to identify the utterance -*put*, *that* or *there*- issued by the end user. ICon does not verify whether the speech and pointing inputs satisfy the temporal constraints expected during the put-that-there interaction technique. Rather, the client application must be programmed to check whether the sequence of events that will lead to the movement of an object has been detected or not. In short, ICon does not support the implementation of fusion of inputs [5].

As seen in Figure 4, ICon models do not contain symbols to represent the state of the prototype or to specify composite events. Moreover, the semantics of its visual language establishes that the data entering a block must be processed and inmediately reinjected to the net, thus preventing the users to model synchronization.
ICon successfully accomplishes its goal of providing its users an easy way to extend their applications so that they can support a wide set of heterogeneous devices. Its work consists of informing the client application about the events ocurring in the environment. Later, the client application will be responsible for interpreting the user's request from these events, tracking the state of the dialog, and responding the user. In other words, all the functionalities of the **fusion engine**, **dialog manager** and **fission component** have to be implemented without support from ICon.

DISCUSSION

This work is the first stage of a research intending to advance the state of the art of toolkits for rapid prototyping of multimodal systems. Such attainment would not be possible without a deep understanding of toolkits, which entails their precise assessment and objective comparison. The absence of metrics for measuring a toolkit's functionality, and our need for precise evaluations are the reasons that made us define the measurement scales presented in this paper.
We proposed to measure the support provided by a toolkit in terms of the components whose implementation it can facilitate. In order to foster the use of this scale, we described a heuristic method that helps us to uncover the capabilities of a toolkit from its visual modeling language. It consists of evaluating the visual language of a given toolkit, seeking for a series of features whose presence reveals the functionality incorporated in its framework. For instance, the presence of composite events in a visual model discloses the toolkit's capability to fuse multimodal inputs.
The support provided by three toolkits was measured on the aforementioned scale by using the proposed heuristics.

The results obtained (Table 1) show that the use of CoGenIVE and HephaisTK leads to a higher reduction of the programming code at the client side. More precisely, CoGenIVE, HephaisTK and ICon all allow their users to invoke the recognition capabilities incorporated in their frameworks, thus releasing them from implementing the recognizers of the intended multimodal prototype. But CoGenIVE and HephaisTK can also detect the multimodal commands issued by an end user (through a series of consecutive or parallel actions). Moreover, both also allow the specification of context-dependent human-machine dialogs (where the messages can be conveyed through complementary, assigned, redundant and equivalent modalities).

	ICon	CoGenIVE	HephaisTK
Recognizers	✓	✓	✓
Fusion Engine	✗	✓	✓
Dialog Manager	✗	✓	✓
Fission component	✗	✗	✗
Synthesizers	✗	✗	✗
Knowledge Source	✗	✗	✗

Table 1. Checkmarks are used to indicate the components whose implementation is supported by a toolkit.

Although the results summarized in Table 1 give us an overall idea of what can be expected from a toolkit during the implementation of multimodal prototypes, they are still coarse-grained and call for extending our measurement scale with additional criteria.
Finally, it seems to be a correlation between the gains provided by a toolkit and the formalism on which its visual language is based on, i.e. CoGenIVE and HephaisTK may exhibit similar functionalities because their visual models are based on state diagrams. Indeed, in an ongoing study we are looking into the existence of classes of toolkits. Such finding will facilitate the understanding, and permit an organized study of toolkits; both can hopefully lead to the design of simpler visual languages and/or more efficient toolkits.

CONCLUSIONS

The novelty of this work is the proposal of a scale for measuring the support provided by a toolkit for the implementation of multimodal prototypes. Such scale is not only a useful reference for the evaluation, but also for the comparison of toolkits. Since the use of this scale requires infering the functionalities that are pre-programmed in the framework of a toolkit, heuristic rules were provided to accomplish this task. We have discussed the results obtained from the evaluations of three toolkits with the proposed scale.

ACKNOWLEDGMENTS

We want to thank the BOF financing of Hasselt University for supporting this research, and our colleague, Jan Van den Bergh, for his valuable feedback.

REFERENCES

1. Bolt, R. Put-that-there: Voice and gesture at the graphics interface. In *SIGGRAPH' 80 Proc. of the 7th annual conference on computer graphics and interactive techniques*, ACM (1980).
2. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and R., Y. Four easy pieces for assessing the usability of multimodal interaction: The care properties. In *Proc. of INTERACT'95* (1995).
3. De Boeck, J., Raymaekers, C., and Coninx, K. Comparing nimmit and data-driven notations for describing multimodal interaction. In *TAMODIA' 06 Proc. of the fifth International Conference on Task Models and Diagrams for User Interaction Design*, Springer Verlag (2007).
4. De Boeck, J., Vanacken, D., Raymaekers, C., and Coninx, K. High level modeling of multimodal interaction techniques using NiMMiT. *Journal of Virtual Reality and Broadcasting 4*, 2 (2007).
5. Dragicevic, P., and Fekete, J. Support for input adaptability in the icon toolkit. In *ICMI'04 Proc. of the 6th International Conference on Multimodal Interfaces*, ACM (2004).
6. Dumas, B. *Frameworks, Description Languages and Fusion Engines for Multimodal Interactive Systems*. PhD thesis, University of Fribourg, 2010.
7. Dumas, B., Lalanne, D., and Oviatt, S. Multimodal interfaces: A survey of principles, models and frameworks. In *Human Machine Interaction*, Springer Verlag (2009).
8. Dumas, B., Signer, B., and Lalanne, D. A graphical uidl editor for multimodal interaction design based on smuiml. In *Proc. of the Workshop on Software Support for User Interface Description Language*, WISE publication (2011).
9. Navarre, D., Palanque, P., Ladry, J., and Barboni, E. ICOs: A Model-Based User Interface Description Technique dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. *ACM Transactions on Computer-Human Interaction 16*, 4 (2009).
10. Nigay, L., and Coutaz, J. A design space for multimodal systems: Concurrent processing and data fusion. In *Proc. of INTERACT'93*, ACM (1993).
11. Vanacken, D. *Touch-based interaction and collaboration in walk-up-and-use and multi-user environments*. PhD thesis, Universiteit Hasselt, 2012.
12. Wahlster, W., Reithinger, N., and Blocher, A. Smartkom: Multimodal communication with a life-like character. In *Proc. of the 7th European Conference on Speech Communication and Technology*, DKFI (2001).
13. Werner, K., Raedle, R., and Harald, R. Interactive Design of Multimodal User Interfaces - Reducing technical and visual complexity. *Journal on Multimodal User Interfaces 3*, 3 (2010).