Defining CARE Properties through Temporal Input Models

Lucio Davide Spano
Department of Mathematics and Computer Science, University of Cagliari
Via Ospedale 72 Cagliari, Italy
davide.spano@unica.it

ABSTRACT

In this paper we show how it is possible to represent the CARE properties (complementarity, assignment, redundancy, equivalence) modelling the temporal relationships among inputs provided through different modalities. For this purpose we extended GestIT, which provides a declarative and compositional model for gestures, in order to support other modalities. The generic models for the CARE properties can be used for the input model design, but also for an analysis of the relationships between the different modalities included into an existing input model.

Author Keywords

Gesture models, input models, multimodality, CARE properties

ACM Classification Keywords

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

INTRODUCTION

The characteristics of gestural input, such as its long temporal extension, or the possibility to support parallel interaction even for single-users, set different challenges that are shared with multimodal interaction. In particular, different modelling techniques and formalisms are shared between gestural and multimodal interaction research. As we discuss better in the related work section, the considered modalities in literature are usually graphical and vocal. However, the wide availability of input devices allowing gestural interaction and the progresses in voice recognition technologies require a general approach for combining modalities, focusing more on User Interface (UI) design rather than technology integration.

In this paper, we enlarge the spectrum of GestIT [12, 13], a declarative and compositional meta-model with an associated code library for describing gestures, generalizing the modelling technique to modalities different from gestural. We show that providing such temporal modelling for multimodal input allows to i) represent the CARE [4] (*Complementarity*, *Assignment*, *Redundancy* and *Equivalence*) properties in the UI code and ii) to identify the CARE properties into existing UI code and/or models. The provided definitions allow checking whether an interaction model is compliant with the intended modality allocation or not, providing the theoretical background for an automatic tool support.

EGMI 2014, 1st International Workshop on Engineering Gestures for Multimodal Interfaces, June 17 2014, Rome, Italy.

http://dx.doi.org/xxx.

RELATED WORK

In this paper, we exploit a formal notation for defining the input sequences that can be recognized by the UI. The idea of describing different types of input through a formal notation has been widely investigated in literature, using different formalisms. For instance, Finite State Machines (FSM) have been integrated into widely adopted window toolkits, such as Java Swing, by Appert et al. [3]. The authors integrate FSMs inside the definition of the UI classes, in order to define in a single place the interaction code. Different FSMs can work together at the same time, in order to avoid the state explosion problem. One of the motivating examples was the drag and drop interaction technique, which cannot be modelled through a single event.

The combination of different interaction modalities needed a formalism able to integrate different concurrent information sources. In [2], Accot et al. used Petri Nets for modelling low-level graphical interaction events. In addition, they created multimodal models starting from single-modalities, composing them into one Petri Net. They exemplified the composition technique defining a bimanual interaction model for a direct-manipulation interface. A similar approach for modelling bimanual interaction has been proposed in the same years in [6].

The research in multimodality has been focused mainly in combining vocal and graphical inputs. Different environments and model based techniques have been proposed in this regard. For instance, Damask [9] is an environment allowing designers to create interfaces through different layers associated to particular devices and modalities. It is possible to specify which elements are common to all devices and modalities and which UI parts are related only to a particular interaction platform. MARIA [10] describes user interfaces through different levels of abstractions. It contains a multimodal platform, which allows generating applications for exploiting web services through both the graphical and the vocal modality. In order to combine them, the language allows specifying

CARE properties [4] for each interactor in the interface, which has both a graphical and a vocal part in its definition.

This paper starts from a compositional and declarative model for gestural interaction and extends it in order to support input from different modalities. We consider the GestIT model in [12, 13], but a similar approach has been adopted in Proton++ [8, 7]. Both of them allow reusing the same gesture models for different applications or in different parts of the user interface, separating the gesture definition from the application behaviour. A comparison between the two approaches can be found in [13], where the authors demonstrate that GestIT is more expressive than Proton++. A library support for the model is publicly available in [1]. Our research aims at combining different modalities, extending the approach for gesture modelling to a more general input model.

INPUT ABSTRACTION

In this section we extend the definition of the declarative and compositional gesture model in [12, 13] for including input modalities different from gestural. We show that having an explicit dialog model in the application definition allows checking the CARE properties [4] on the input temporal sequence.

Ground terms

GestIT models the input through an expression defining the input temporal sequence. The ground term expressions represent the basic building blocks for such description. Each of them describes an atomic event, that is a notification that cannot be further decomposed. In general, they are associated to a value change of a *feature*, which is a single data tracked by an input device.

For instance, we can define a ground term representing the current mouse pointer position, or a key press on a keyboard. In the gestural modality, we associate a ground term to each point tracked by the recognition device, such as the skeleton joints for MS Kinect, or the touch points for a multitouch screen. In the vocal modality, we can associate a ground term to the recognition of a word (or a phrase) pronounced by the user.

The notification can be optionally associated to a condition, which can be exploited receiving only a subset of all possible state changes associated to a ground term. This is useful in the gestural modality for receiving movements that follow a specific trajectory (e.g. linear), or in order to calculate differential characteristics of the gesture performance (e.g. the speed). In the vocal modality, we can associate a word or a grammar to an utterance recognition.

In this paper, we associate a symbol to each considered ground term. For instance, we indicate with H_r the position of the right hand, with S_l the position of the left shoulder etc.. We represent a generic ground term with the symbol G_t. For the vocal modality, we use the symbol V for indicating the recognition of a word.

A boolean predicate can be associated to a ground term for filtering the notification. We represent it through its name in square brackets, immediately after the ground term name: $G_t[p]$ represents a generic ground term G_t associated to a predicate p. For instance $V[quit]$ expresses the recognition of the word "quit", where V is the vocal ground term and *quit* is the predicates that restrict the recognition only to the considered word. A predicate p can be associated to more than one ground term in both the formal notation and in the library supporting the GestIT model.

In GestIT [12], a generic ground term is represented by an abstract class (*SimpleExpr*), which can be extended for including different sources of input. A predicate can be associated with a ground term providing the implementation of the boolean method *accept*, through a delegate pattern [5].

Composition operators

Starting from ground terms, it is possible to define the input temporal sequence composing the different expression through the following set of temporal operators:

- **Iterative Operator**, represented by the $*$ symbol, repeats an input expression an indefinite number of times.
- **Sequence Operator**, represented by the $\gg$ symbol, connects two (or more) expressions to be executed in sequence, from left to right.
- **Parallel Operator**, represented by the symbol $\|$, connects two (or more) expressions that can be recognized at the same time.
- **Choice Operator**, represented by the symbol $[\,]$, allows to select one among the connected components in order to complete the entire expression.
- **Disabling Operator**, represented by the $[>$ symbol, defines that an expression stops the recognition of another one, typically used for stopping iteration loops.
- **Order Independence**, represented by the $|=|$ symbol, defines that the connected sub-expression can be performed in any order.

The choice, parallel and order independence operators are commutative. The sequence, parallel, choice and disabling operator are binary and associative. The order independence operator is a n-ary operator, it can be defined through a choice of sequences, as we detail better while describing equation 7.

In GestIT, a composed expression is represented by the class *ComplexExpr*, where it is possible to include the operands (either simple or composite) and to specify the temporal operator. This class is independent from the modality and it can be used for connecting ground terms from different input devices.

Auxiliary functions

We define two auxiliary functions for defining the CARE properties on input expressions. The first function answers the following question: which modalities are needed

for completing the considered input expression? Obviously, since it may be possible to provide the same information through different modalities, it is possible to exploit different sets of modalities for completing the same expression.

We formally model this association through the Mod function, which maps an expression to a family of sets. The family contains all the possible sets of modalities that the user can exploit for completing the considered input expression. We provide an inductive definition for this function in equation 1, starting from a ground term and showing how to build the family of sets for two generic expressions connected through a composition operator. M is the set of all possible input modalities.

$$
\begin{aligned}
&Mod : Exp \mapsto \{S_1 ... S_n\} \\
&\quad \text{where } S_i \subseteq M, i \in [1, n] \\
&\\
&Mod(G_t) = \{m\},\ m \in M \\
&Mod(Exp^*) = Mod(Exp) \\
&Mod(Exp_1 [\,] Exp_2) = \{S_i | S_i \in Mod(Exp_1) \vee \\
&\quad S_i \in Mod(Exp_2)\} \\
&Mod(Exp_1\ op\ Exp_2) = \{S_{i,j} | S_{i,j} = S_i \cup S_j, \\
&\quad S_i \in Mod(Exp_1), S_j \in Mod(Exp_2)\} \\
&\quad op \in \{ \gg, [>, |=| \, , \|\}
\end{aligned}
\tag{1}
$$

A ground term G_t is assigned to a single modality, therefore the function Mod maps it to a single set with a single element. The iterative operator does not add any element to the family of sets.

The choice operator allows selecting between one of the sub-expressions. From the modality point of view, it is possible to choose only one of the sets provided either by the left or the right operand. Therefore, the resulting family of sets for the composed expression is obtained considering all sets from both operands.

For the other composition operators, the user needs to complete both operands, which means that she can select a set of modalities for completing the left operand and another set for completing the right operand. Therefore, in order to obtain the resulting family of sets, we first calculate all possible pairs of sets, selecting one set from left and one set from the right operand families. After that, we calculate for each pair the union of the two sets and we obtain the resulting family for the composed expression.

The second function associates an input expression with a semantic label identifying which input data provides to the application. Considering two input expressions Exp_1 and Exp_2, the value returned by $Input$ function is the same if they provide an equivalent input for the application. The possible values for this function are related to the specific application. The function is defined in equation 2: $Data_{in}$ is a set of semantic labels associated to all the possible user's input needed by the considered application, and d is a particular label.

$$
\begin{aligned}
&Input : Exp \mapsto d \\
&\quad d \in Data_{in}
\end{aligned}
\tag{2}
$$

This association between an expression and its semantic label represents the abstraction of the "message" that the system needs from the user for completing an operation. The same input can be provided through different modalities. For instance, we can consider an application allowing the user to enter a phone number either dialling it through a virtual keyboard or to pronouncing it vocally, as modelled in equation 3. In this case, we consider two different input expressions: one defining the screen taps (Tap) and one for the vocal interaction (Num). The first one is simply a sequence of a touch start ($Start_1$) and a touch end (End_1). The second expression allows the user to dictate the number digit by digit. Such expressions provide the same data to the application logic (represented by the $phoneNumber$ label).

$$
\begin{aligned}
&Tap\ [\,]\ Num^* \\
&Tap = Start_1 \gg End_1 \\
&Num = V[zero]\ [\,]\ V[one]\ [\,]\ \ldots\ [\,]\ V[nine] \\
&\\
&Input(Tap) = Input(Num) = phoneNumber
\end{aligned}
\tag{3}
$$

In this paper, we assume that the $Input$ function has been defined for each expression in the input model. The straightforward way for obtaining this mapping is to require its specification from designers, but this may limit the acceptance of the modelling technique.

We think that it is possible to provide an automated labelling procedure based on the reverse engineering of code. If we consider the Tap and Num expressions, the behaviour associated to their completion should contain a call to the same application logic, e.g. through a command pattern [5]. We aim to investigate the identification of such patterns in future work.

MODELLING CARE PROPERTIES

The CARE properties [4] are a simple way for characterizing how different modalities relate to each other for supporting the interaction. In this section, we show how it is possible to model such properties through the composition operators defined in GestIT, starting from their definitions in [4]. The care properties are four (*Complementarity*, *Assignment*, *Redundancy* and *Equivalence*). In the following discussion we include their definition (reported from [4]) and their modelling through the proposed notation.

In all definitions, we suppose that the UI is currently in a state s_t and that the user wants to change this state to s_{t+1}. Exp is the input expression defining all the possible UI options for reaching s_{t+1} from s_t. In addition, we consider two modalities in the definitions, but they can be easily extended to the general case.

Assignment

Definition. The modality m is assigned from state s_t to reach s_{t+1} if no other modality can be used for moving from s_t to s_{t+1}.

Model. We can define this property on the expression Exp simply checking that all ground terms belong to the modality m, as defined in equation 4.

$$Assignment(Exp, m) \Leftrightarrow Mod(Exp) = \{m\} \quad (4)$$

Equivalence

Definition. Two (or more) modalities $m1$ and $m2$ are equivalent for changing the state of an application from s_t to s_{t+1} if it is necessary and sufficient to use any one of the modalities for changing the state. This means that the user is free to select exactly one modality among the ones supported by the interface for completing one action.

Model. We can define the equivalence between two different modalities connecting the expressions for $m1$ and $m2$ with a choice operator as shown in equation 5 (Exp_1 is assigned to $m1$ and Exp_2 is assigned to $m2$). However, this is not sufficient for being compliant with the definition. For ensuring that s_{t+1} is the same independently from which expression the user selects, we must ensure also the same type of input is provided by the user through both expressions. This is modelled by the condition $Input(Exp_1) = Input(Exp_2)$.

$$\begin{aligned} &Equivalence(Exp, \{m1, m2\}) \\ &\quad \Leftrightarrow Exp = Exp_1 \,[\,]\, Exp_2 \\ &\quad \wedge Mod(Exp_1) = \{m1\} \\ &\quad \wedge Mod(Exp_2) = \{m2\} \\ &\quad \wedge Input(Exp_1) = Input(Exp_2) \end{aligned} \quad (5)$$

Redundancy

Definition. Two (or more) modalities are used redundantly to reach state s_{t+1} from state s_t, if they have the same expressive power (e.g. the user provides the same information to the UI through different channels or vice versa).

Model. As discussed in [4], the redundancy may occur following two different temporal relationships: sequence and parallelism. In the first case, before completing the state transition, the user selects the first modality and completes the correspondent actions, then she selects the second modality and completes the interaction, providing again the same input. In general, if the number of modalities is more than two, it is possible to select the modalities in any order, but the user has to complete the actions for all of them. In the second case (parallelism), the user can perform actions belonging to different modalities at the same time. Therefore, we can define in equation 6 two variants for the redundancy property.

$$\begin{aligned} &SeqRedundancy(Exp, \{m1, m2\}) \\ &\quad \Leftrightarrow Exp = Exp_1 \,|\!=\!|\, Exp_2 \\ &\quad \wedge Mod(Exp_1) = \{m1\} \\ &\quad \wedge Mod(Exp_2) = \{m2\} \\ &\quad \wedge Input(Exp_1) = Input(Exp_2) \end{aligned} \quad (6)$$

$$\begin{aligned} &ParRedundancy(Exp, \{m1, m2\}) \\ &\quad \Leftrightarrow Exp = Exp_1 \,||\, Exp_2 \\ &\quad \wedge Mod(Exp_1) = \{m1\} \\ &\quad \wedge Mod(Exp_2) = \{m2\} \\ &\quad \wedge Input(Exp_1) = Input(Exp_2) \end{aligned}$$

The two versions model the property in a similar way. The only difference is the temporal operator connecting the two expressions Exp_1 and Exp_2, which are assigned to only one modality (respectively $m1$ and $m2$).

The first definition, $SeqRedundancy$, connects the two expressions through the order independence operator, which guarantees that the two input sub-expression must be both completed in order to reach the state s_{t+1}. The user is free to select which modality she wants to use first. Indeed the order independence is by definition a choice between all possible sequences for executing the input actions, as shown in equation 7.

$$Exp_1|\!=\!|Exp_2 = (Exp_1 \gg Exp_2)[\,](Exp_2 \gg Exp_1) \quad (7)$$

In the second definition, Exp_1 and Exp_2 are connected through the parallel operator, which allows the execution of the input actions assigned to the different modalities at the same time. It is worth pointing out that the sequences of actions recognized by expressions satisfying $SeqRedundancy$ can be also recognized by those that satisfy $ParRedundancy$. However, as also remarked in [4], when two modalities compete for the same human resources (e.g. the same buffers in sensory memory) the designer should avoid a parallel redundancy and restrict it to the sequential version.

In both cases, the command for changing the UI state is associated to the completion of Exp, which ensures that the UI changes its state only when the input from both modalities has been provided.

Complementarity

Definition. Two (or more) modalities are used for reaching the state s_{t+1} from s_t, all of them must be used for changing the state, but no one is able to complete the change individually.

Model. In this case, there is no need to enforce a particular temporal sequence for the input actions in Exp. The only requirement is that $Mod(Exp)$ contains a single set, whose elements are all the considered modalities. Considering the definition we provided for the function Mod, this guarantees that the user completes the input expression using all modalities at least once. The formal definition is shown in 8.

$$Complementarity(Exp, \{m1, m2\}) \Leftrightarrow Mod(Exp) = \{m1, m2\} \quad (8)$$

SAMPLE APPLICATION

In this section we redesign the interaction applying the CARE properties to the touchless recipe browser presented in [11]. We show that it is possible to check the constraints on the input expression automatically, applying the definitions discussed in the previous sections.

The touchless recipe browser allows the user to select among different recipes, groped in different categories (e.g. starters, first courses, second courses, desserts etc.). After selecting the dish, the application presents all the steps for cooking, through textual descriptions enhanced with a video. It is possible to watch the video entirely, or step by step. In the latter mode, the video stops at each intermediate action that the user has to complete for cooking the dish, and the playback continues when the application receives an explicit request by the user.

The application combines the gestural (g) and the vocal modality (v), since it may be difficult for a person to use a keyboard and/or a mouse while cooking (she may have wet or dirty hands). It consists of three different presentations: the first shows the recipe category, the second allows to select a dish from a list of a specific category and the last one shows the steps for preparing the selected dish.

All presentations respond to the input only if the user stands in front of the screen, in order to ignore movements or speech when they are not intended for interacting with the application (e.g. movements for chopping a carrot or talking with other persons in the kitchen). We can model this input filtering by recognizing the user pose in front of the screen. It is sufficient that the user's shoulder joints are contained in a plane roughly parallel to the screen. The interaction is modelled in equation 9.

$$\begin{aligned} AppIn &= Front \gg Interact^* \,[> NotFront \\ Front &= (S_l[p] \parallel S_r[p]) \\ NotFront &= (S_l[\bar{p}] \parallel S_r[\bar{p}]) \end{aligned} \quad (9)$$

The definition of the expressions $Front$ and $NotFront$ is similar, since they track the parallel movement of the left and right shoulder joints (represented by the S_l and S_r ground terms). They are complementary, since all movements recognized by $Front$ are not recognized by $NotFront$ and vice versa. This is modelled by the p predicate, which tests whether the two joints are parallel with respect to the screen plane. $NotFront$ uses the logical negation of the same predicate.

$Interact$ contains the expression defining the interactions supported by the application, which we refine later on. The sequence operator between $Front$ and $Interact$ guarantees that no input is processed while the user is not in front of the screen. The disabling operator between $Interact$ and $NotFront$ ensures that the input tracking finishes as soon as the user moves from the screen front position.

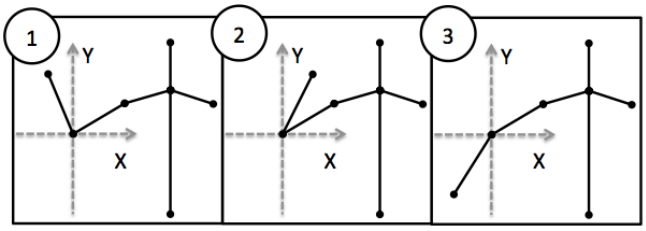

Figure 1. The wave gesture

Such design of the interaction, *assigns* the expression of the user's will to interact to the gestural modality, as demonstrated in equation 10.

$$\begin{aligned} Mod(Front) = Mod(NotFront) = \\ = Mod(S_l) \cup Mod(S_r) = g \end{aligned} \quad (10)$$

While interacting with all presentations, it is possible to quit the application or to go back to the previous presentation. In order to avoid unintended terminations, we ask a confirmation for the quit command, as usual in many interfaces. In the vocal modality, the user says the word "quit" for quitting the application, while in the gesture modality we associate this command to a wave gesture.

The wave gesture is depicted in figure 1. For convenience, we set the origin of our coordinate system on the elbow of the considered hand, obtained at each frame simply defining a translation of the original coordinate system, using the vector defined by the elbow position. The gesture starts when the hand point reaches the second quarter in our coordinate system, with a positive Y and a negative X value, depicted in figure 1. Then, the user has to move the hand in the first quarter of the coordinate system, with both values of X and Y positive, as shown in figure 1, part 2. At this point, there are two alternatives: either the user repeats the wave, returning to the situation in figure 1, part 1, or she can conclude the gesture moving the hand in the third quarter, as depicted in figure 1, part 3.

We model the gesture with the expression in equation 11. The position of the right hand is represented by H_r; x is true if the hand point has a positive value for the X coordinate, $\bar{x}$ is true otherwise; y is true if the hand point has a positive value for the Y coordinate, $\bar{y}$ is true otherwise.

$$\begin{aligned} &(H_r^{\{1,*\}}[\bar{x} \wedge y] \,[> \; H_r^{\{1,*\}}[x \wedge y])^{\{1,*\}} \,[> \\ &H_r^{\{1,*\}}[\bar{x} \wedge \bar{y}] \end{aligned} \quad (11)$$

In order to support the exit confirmation using different modalities, it is possible to follow different strategies. We can ask the user to provide the input sequentially or in parallel, since voice and gestures do not cause conflicts. The two situations are modelled in equation 12, respectively by the Seq_{quit} and by the Par_{quit} expressions.

Figure 2. Dish category selection

Figure 3. Dish preparation

$$
\begin{aligned}
&Seq_{quit} = V[quit] \,|=|\, Wave \\
&Par_{quit} = V[quit] \;\|\; Wave \\
&Mod(V[quit]) = \{v\} \\
&Mod(Wave) = \{g\} \\
&Input(V[quit]) = Input(Wave)
\end{aligned} \tag{12}
$$

Since the equivalence between the inputs entered through the expressions using the vocal and the gesture modality holds (considering the interaction semantics we associated them in this specific case) we can conclude that both $SeqRedundancy(Seq_{quit}, \{v, g\})$ and $ParRedundancy(Par_{quit}, \{v, g\})$ are true.

Another strategy is to allow only one modality for the first request (e.g. vocal) and then a confirmation through either the vocal or the gesture modality (the vice versa is also possible, with a symmetric modelling technique). In this case, the command received through the vocal command "enables" a selection between a gesture and the same vocal command for confirmation. The strategy is modelled in equation 13 by the $AssEquiv_{quit}$ expression. We have an *assignment* property for the first request and an *equivalence* property for the confirmation.

$$AssEquiv_{quit} = V[quit] \gg (Wave[\,]V[quit]) \tag{13}$$

We assign the back command to the vocal modality ($V[back]$). Therefore, the interaction with a generic presentation ($Interact$, see equation 9) consists of a presentation-specific ($PresIn$) expression in choice with the back and the quit command, as defined in equation 14, where Exp_{quit} is a place holder for either Seq_{quit}, Par_{quit} or $AssEquiv_{quit}$.

$$Interact = PresIn_i \,[\,]\, V[back] \,[\,]\, Exp_{quit} \tag{14}$$

In the first presentation, which is shown in figure 2, the user points a target on the screen and selects it closing the hand. The interaction is modelled in equation 15: the user moves the dominant hand an indefinite number of times (H_r^*), until she selects an item closing the hand (we indicate the feature for the hand opening with oH_r, and we restrict the recognition only to the *closed* state). In this case, we assigned the interaction to the gestural modality. The interaction for the dish selection presentation exploits the same gesture set.

$$
\begin{aligned}
&PresIn_1 = H_r^* \,[>\, oH_r[closed] \\
&Mod(PresIn) = g
\end{aligned} \tag{15}
$$

The presentation for preparing the selected dish is shown in figure 3. It allows reading the description of the steps that are needed for cooking a dish, through a text and a video. The video can be played continuously, in order to have an overall idea on the whole preparation process. Otherwise, the video can be played step-by-step, pausing the playback at the end of each step. In this case, the user requests to watch the next (or previous) step explicitly. In addition, she can jump randomly from one step to another, moving the video timeline knob. Finally it is possible to change the volume setting for the video description. The interface supports the interaction with the expression in equation 16. The change between the continuous and the step-by-step playback is supported through the "continuous" and the "step" vocal commands (respectively $V[cont]$ and $V[step]$.

In order to navigate the recipe steps, the user can select among the gestural and the vocal modality. The next step ($Next$) can be visualized through the "next" vocal command ($V[next]$) or (choice operator) through a swipe from left to right. The latter gesture can be modelled through an iterative movement of the dominant hand with a speed higher than a specific threshold ($H_r^*[sp_r]$), disabled by a movement slower than this threshold ($H_r[\bar{sp}_r]$). We use a symmetric modelling approach for visualizing the previous step.

The steps can be randomly navigated performing a grab gesture and dragging the timeline knob ($Random$): it consists of closing the dominant hand ($oH_r[closed]$), followed by its iterative movement (H_r^*), disabled by opening the hand ($oH_r[open]$). The same gesture can be used for changing the volume setting, after having pronounced the "volume" vocal command.

The $Volume$ expression in equation 16 satisfies the *complementary* property, since $Mod(Volume)$ includes only one set containing both the vocal and the gesture modality. This means that the input expression cannot be

completed without using all the considered modalities. Instead, in *Next* we have an *equivalence* between the two sub-expressions since they provide the same input through different modalities (the same holds for *Previous*). The "continuous" and the "step" commands are *assigned* to the vocal modality.

$$\begin{aligned} PresIn_3 &= V[cont]\,[\,]\,V[step]\,[\,]\,Volume\,[\,] \\ &\quad Next\,[\,]\,Prev\,[\,]\,Random \\ Volume &= V[vol] \gg Drag \\ Next &= V[next]\,[\,]\,(H_r^*[sp_r]\,[>\ H_r[\bar{sp}_r]) \\ Prev &= V[prev]\,[\,]\,(H_r^*[sp_l]\,[>\ H_r[\bar{sp}_l]) \\ Random &= Drag \\ Drag &= oH_r[\bar{op}]\ \gg\ H_r^*\,[>\ oH_r[op] \\ Mod(Volume) &= \{g, v\} \\ Mod(V[step]) &= Mod(V[cont]) = \{v\} \\ Mod(Random) &= \{g\} \\ Input(V[next]) &= Input(H_r^*[sp_r]\,[>\ H_r[\bar{sp}_r]) \end{aligned} \tag{16}$$

CONCLUSIONS AND FUTURE WORK

In this paper we extended an existing declarative model for gesture definition in order to support other modalities. With this extension, we provided the definition of the CARE properties for managing multimodality, representing them through the temporal input modelling. This is useful for providing automatic model checking procedures, able to identify the properties inside existing models. This can support designers in the UI creation process or during the reverse engineering of existing applications. In future work, we aim to include the multimodal support in the next release of the GestIT library, together with tool support for the model creation and checking.

ACKNOWLEDGEMENTS

We gratefully acknowledge Sardinia Regional Government for the financial support (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1 "Avviso di chiamata per il finanziamento di Assegni di Ricerca"

REFERENCES

1. GestIT library `http://gestit.codeplex.com/`. Accessed: 2014-05-13.
2. Accot, J., Chatty, S., and Palanque, P. A. A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. In *DSV-IS*, F. Bodart and J. Vanderdonckt, Eds., Springer (1996), 92–104.
3. Appert, C., and Beaudouin-Lafon, M. SwingStates: adding state machines to the swing toolkit. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, ACM (New York, NY, USA, 2006), 319–322.
4. Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M. Four easy pieces for assessing the usability of multimodal interaction: the care properties. In *InterAct*, vol. 95 (1995), 115–120.
5. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994.
6. Hinckley, K., Czerwinski, M., and Sinclair, M. Interaction and modeling techniques for desktop two-handed input. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, UIST '98, ACM (New York, NY, USA, 1998), 49–58.
7. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton++ : A Customizable Declarative Multitouch Framework. In *Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST 2012)*, ACM Press (Berkeley, California, USA, 2012), 477–486.
8. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton: multitouch gestures as regular expressions. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (CHI 2012)*, ACM Press (Austin, Texas, USA, 2012), 2885–2894.
9. Lin, J., and Landay, J. A. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, ACM (New York, NY, USA, 2008), 1313–1322.
10. Manca, M., and Paternò, F. Supporting multimodality in service-oriented model-based development environments. In *Human-Centred Software Engineering*, R. Bernhaupt, P. Forbrig, J. Gulliksen, and M. Lrusdttir, Eds., vol. 6409 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, 135–148.
11. Spano, L. D. Developing Touchless Interfaces with GestIT. In *Ambient Intelligence*, F. Paternò, B. de Ruyter, P. Markopoulos, C. Santoro, E. van Loenen, and K. Luyten, Eds., vol. 7683 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2012, 433–438.
12. Spano, L. D., Cisternino, A., and Paternò, F. A Compositional Model for Gesture Definition. In *Proceedings of the 4th International Conference in Human-Centered Software Engineering (HCSE 2012)*, vol. 7623, LNCS, Springer (Tolouse, France, 2012), 34–52.
13. Spano, L. D., Cisternino, A., Paternò, F., and Fenu, G. A Declarative and Compositional Framework for Multiplatform Gesture Definition. In *EICS 2013, 5th Simposium on Engineering Interactive Computing Systems*, ACM Press (2013).