Definition of Gesture Interactions Based on Temporal Relations

Dominik Rupprecht[1], Daniel Künkel[1], Rainer Blum[1] and Birgit Bomsdorf[1]
[1]*Department of Applied Computer Sciences, Fulda University of Applied Sciences, Leipzigerstraße 123, Fulda, Germany*
{dominik.rupprecht, daniel.kuenkel, rainer.blum, birgit.bomsdorf}@cs.hs-fulda.de

Keywords: 3D-Gesture Interaction, Touchless Interaction, Gesture Development, Gesture Notation, UI Feedback.

Abstract: Taxonomies reported in the literature and in technical instructions define terms of gestures and gesture interactions similarly, but hold differences in semantics that may lead to misunderstandings. However, in a heterogeneous development team a common understanding of concepts and notions is of utmost importance. In this paper, we present an approach to a more universal definition of gesture interactions and gesture types, respectively. We define a notation of gesture interactions using a specific combination of the temporal intervals of gesture execution, the user interface feedback and the system functionality to effectively consider all three perspectives. We do not introduce a completely different approach, but extend and combine existing work.

1 INTRODUCTION

In applications in which interactions are based on touchless gestures, movements of the human body are direct input to the system. A movement may be a hand or arm gesture, a head gesture or an upper body gesture, depending on which parts of the body should be involved in interactions. The intended gestures are communicated in different forms throughout the development process. Illustrations and demonstrations are useful in the communication with users, e.g. in the phase of conceptual design. Formal notations, e.g. XML-based languages, are practical for the gesture specification aiming at their implementation. The more different forms of describing gesture interactions are in use, the more important a common understanding of the term gesture is. The study of taxonomies reported in literature and technical instructions shows that such a common understanding does not exist yet. Although terms of gestures and gesture interactions are defined similarly, there are differences in semantics that may lead to misunderstandings during the design process. We experienced this several times in our own work on gesture-based applications.

For example, in the context of Microsoft's Visual Gesture Builder (Microsoft 2013) the difference between discrete and continuous gestures is based on the implementation of how a recognized gesture is reported to the application program logic. If a yes-no-indicator is used to inform the application about a performed gesture, this gesture is named discrete (because the indicator is discrete). A continuous gesture is reported by an indicator whose value is between 0 and 100, thus documenting the progress of the gesture execution, e.g. 60% done with the gesture. In contrary in literature, the difference between discrete and continuous gesture is based on system reaction (Wobbrock et al. 2009; Ruiz et al. 2011). If it starts right after the gesture, it is called discrete. In the case the feedback is performed simultaneously with the gesture, it is called continuous.

Particularly, different project participants like user interface designers, target users and programmers have different understandings of what constitutes a gesture and how different types could be classified. If they discuss gestures for a system to be implemented they might use the same terms but mean different types of gestures leading to misunderstandings. Therefore, the approach to gesture definition proposed in this paper is not only based on existing taxonomies but also on the different views taken by project members and aims to support and simplify the communication between these groups. The kernel of our concept are temporal intervals (inspired by Allen (1983)) of gestures and system reactions, and relations between them. A continuous gesture interaction, for example, is defined by temporal intervals starting and ending at the same time. This work does not introduce a complete different approach but extends and

combines existing work resulting in more universal terms of gestures and basic classification. While the different roles involved in the development process can keep their usual taxonomies, our common terms should raise awareness of existing differences in notions and concepts, and facilitate the communication between the groups.

In the next section, existing gesture taxonomies are presented. This is followed by an overview of the different views taken by developers and target users involved in the development process. Afterwards our approach of a gesture terminology is introduced. The paper ends with possible extensions and future work.

2 EXISTING GESTURE TAXONOMIES

Several different gesture taxonomies exist in the literature, intended to classify gestures for different types of user interfaces. Gestures are classified by means of specific criteria (also called *aspects* by Hummels and Stappers (1998); or *dimensions* by Ruiz et al. (2011) and by Wobbrock et al. (2009)). Such classification schemes are referred to in studies to support the design of gesture interactions. As taxonomies imply a general vocabulary, they facilitate a common understanding for all persons involved (for example, what a gesture is).

The differences between taxonomies are quite significant, in parts. Hummels and Stappers (1998) work in the context of product design of virtual 3D objects and classify gestures concerning spatial information (i.e., selecting one or several objects by pointing, changing the form of a 3D object by pushing or pulling, etc.). While this classification, dealing with a concrete use case, goes quite into detail, Karam and Schraefel (2005) report a more generic approach. They focus on aspects like the application domain (desktop, gaming, entertainment, etc.), enabling technologies (perceptual, i.e., vision, audio, remote sensing; non-perceptual, i.e., mouse, keyboard, tangible devices, etc.) and the system response (visual, audio, CPU commands, etc.). This constitutes a technically-oriented scheme that does not pay attention to the separate aspects of the actual interaction and their interrelations.

In the aforementioned and in further taxonomies (Obaid et al. 2014; Pavlovic et al. 1997; Ruiz et al. 2011; Wobbrock et al. 2009) additional central characteristics like the manner of execution of gestures and the feedback are included. Though the feedback of the user interface that is tied to a gesture is an important aspect in gesture interaction, the system-centric view should not be disregarded. However, the existing literature neglects the domain of system functionalities that are not directly visible to the user (though, of course, results become visible via the UI). An example of such a system functionality would be a database query or playing an audio CD. These taxonomies are therefore incomplete from the perspective of designers and programmers, that have to consider these system functionalities in combination with gestures.

Furthermore, the classification characteristics often rely on various pre-conditions. For example, if a user interface is not tangible, some of the published classification aspects from Wobbrock et al. (2009) cannot be used, e.g., the binding dimension that describes the relationship between touch gestures, the UI objects and the touchscreen. Alternatively, these classification aspects would need to be adapted to gesture studies that do not meet the respective pre-conditions.

In summary, a lack of generalizability or comparability, but a somewhat confusing proliferation can be identified in the published work. By contrast, our approach avoids these issues, as it is designed to be more universal and less technically-focused and avoids pre-conditions. Instead, it focuses on aspects like the temporal relations between gestures and UI feedback and system functionality.

3 USAGE AND SYSTEM VIEW

The development of interactive systems implies different views, i.e., the perspective of using the system and the perspective of implementing the system (Hix and Hartson 1993). The usage view is typically taken by designers as well as by users (see Figure 1).

Designers develop a conceptual model of the intended gesture execution and the according UI behavior. Users inevitably develop their personal mental model of how to utilize the interactive system und how to interact with it by means of gestures. The conceptual and the mental models must correspond as much as possible to avoid usage problems. Thus, as shown in Figure 1, development aims at a single usage view, which results from a common understanding of the system behavior by designers and users.

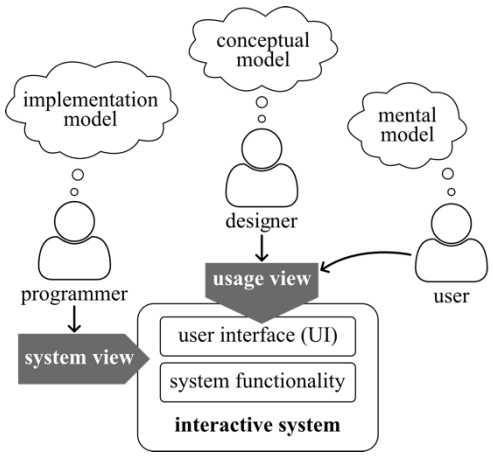


Figure 1: Different views and models.

Communication throughout the development process and between the different team roles, however, is based on different notations and on inconsistent gesture terminology. Furthermore, in user-centered design target users are participating not only in the evaluation but typically already in the earlier phase of ascertainment of gestures (Künkel et al. 2015). For example, in the usage view, gestures are referenced often via demonstration of the movements (live or pre-recorded video) and pictorial representations, such as shown in Figure 2. However, a common standard for describing gestures is not available.

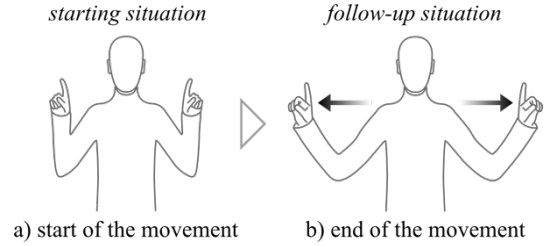


Figure 2: Pictorial representation of a gesture movement.

An additional aspect is how the UI responds to a gesture. Therefore, gestures should not be described in isolation but need to be shown in the context of UI behavior. In the case of the gesture depicted in Figure 2, for example, a UI element may be highlighted as soon as the movement starts, be enlarged until the end, and then be de-highlighted (implementing, e.g., a zoom function). The User Action Notation (UAN) (Hartson et al. 1990) is well-known for specifying the usage view. The authors also argue the necessity not only of specifying the input device interactions in detail, but also the observable UI behavior, timing conditions and invocation of system functionality. In the literature (e.g. Loke et al. 2005) some gesture specific notations are proposed that consider these aspects.

While the usage view omits implementation, these are subject of the system view. Programmers care about sensors and recognition algorithms. For example, using Microsoft's Kinect technology (https://developer.microsoft.com/en-us/windows/kinect/hardware) to recognize the movement depicted in Figure 2 the sensor captures all the movements of a user, whether they are meaningful or not for the interaction. A movement is captured by the sensor as a continuous sequence of images (frames). Every frame is analyzed by a gesture detector. This detector checks continuously if a user executes one of the gestures defined beforehand. It then provides the result to the application logic by indicators, whether a gesture is recognized or not. Therefore, in the system view a gesture occurs as a value that the gesture detector indicates. So, these values and how to process them, is important when working with gestures. The following pseudocode shows how to technically bind a specific gesture to the corresponding UI behavior and the system functionality as part of the implementation model:

```
while(gestureFrames){
   //check indicator
   If zoomInGesture.isDetected(){
      //invoke system funtinality
      object.zoomIn();
      //invoke UI feedback
      UI.updateView();
   }
   If further gestures …
}
```

It is a particular challenge for the developer to integrate this technically-driven, today often still very sensor-specific implementation with the required implementation of the conceptual model of the designer.

Members of a project team plus test users, as shown above, take different views while developing an interactive system. The dependencies and correlations of the respective models and perspectives emphasize the need of common gesture terms to facilitate the communication throughout the development process and between the different roles. Such a gesture notation must consider all three aspects: the movements and positions of the human body, the UI behavior, and the system functionality.

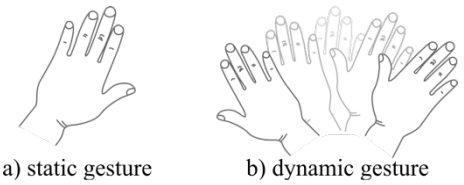
4 GESTURE INTERACTION

In this section we propose an approach for defining gestures in the context of both, the UI and the system behavior. The terms extend definitions from the literature by referring to the execution time intervals of user's movement, UI feedback, and system functionality.

4.1 Gesture, Static and Dynamic

In the field of HCI, a gesture is a coordinated movement or position of a body or parts of a body with the intent to interact with a system (Hummels and Stappers 1998; Saffer 2008). For example, a swipe of a hand to the side is a coordinated movement, whereas holding a hand over a button specifies a position. The distinction between coordinated movement and position, i.e., the dynamic of gesture execution (*form* in Wobbrock et al. (2009)) is a frequently used classification criteria.

A *static gesture* is a meaningful pose or posture, e.g. a hand and finger position, without any movements (Nielsen et al. 2004). Holding a hand steady towards the system for a specified time (e.g. two seconds) is an example for a static gesture (see Figure 3 a).



a) static gesture b) dynamic gesture

Figure 3: Example for static and for dynamic gesture execution.

A *dynamic gesture* is defined by its motion sequence (Karam and Schraefel 2005). It is a movement like a specific trajectory of the hand and/or a transition between two or more postures (Nielsen et al. 2004). Figure 3 b) shows the wave of a hand as an example for a dynamic gesture. In contrast to static postures, dynamic gestures have more variance in their execution (different speeds or kinematic impulse c.f. Ruiz et al. (2011)).

Each gesture execution possesses a start, an end, and a duration defining its time interval. Execution of a swipe gesture may take a few milliseconds, while execution of a holding gesture may require a few seconds.

Throughout this paper, the start, end, and duration of a time interval are referenced by means of the dot notation. The 2-second-interval of a holding gesture, for example, is noted by *holding_gesture.duration = 2* seconds. In the figures, a time interval is represented by an arrow symbol (note that the length of the arrow is not intended to represent the actual duration). A gesture symbol (✋) is added to the arrow in case that the time interval of a gesture execution is to be denoted (see Figure 4).

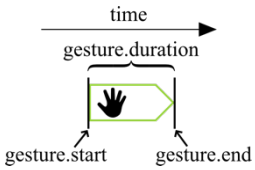


Figure 4: Gesture notation to denote the time interval of a gesture execution (the included dot notation is not part of the actual representation but serves the purpose to introduce the graphical elements).

4.2 Discrete and Continuous Gesture Interaction

A gesture is often defined as a meaningful physical movement of the body or parts of the body in combination with the UI feedback (Mitra and Acharya 2007; Obaid et al. 2014; Pavlovic et al. 1997; Wobbrock et al. 2005). This definition takes into account that interactions are not only composed of a gesture execution but also include UI feedback (called as *flow* by Wobbrock et al. (2009); or *temporal* by Ruiz et al. (2011)). Using the term gesture for both, a body movement and a body movement in conjunction with the UI feedback, mixes these two concepts. In our gesture projects, the developers experienced that this leads to confusion in communication. This holds true particularly in discussions about so-called discrete and continuous gestures (Ruiz et al. 2011). Therefore, we suggest to differentiate between the definition of a gesture (as defined in section 4.1) and a gesture interaction.

A *gesture interaction* is a gesture in conjunction with the respective UI behavior triggered by the gesture. Analogous to a gesture, the execution of a UI feedback has a duration (time interval). The feedback may be for example a navigation to a web page that takes place within a few milliseconds, or a graphical animation taking several seconds. In the following, again for simplicity, we neglect the exact duration since the focus of our approach lies on the temporal relationship between gestures and UI feedback.

The dichotomous aspects discrete and continuous refer to the relation between the execution of a

gesture and the feedback. Both characteristics can be defined precisely based on the term gesture interaction.

A *discrete gesture interaction* is a gesture interaction in which the UI feedback is executed right after the end of the gesture, i.e. *gesture.end = UI.start*. Figure 5 depicts this temporal relation by means of our suggested gesture notation. The arrow for the UI interval (containing an illustrated user silhouette with a small arrow pointing at it) represents feedback in arbitrary form (visual, auditory, haptic, etc., combinations).

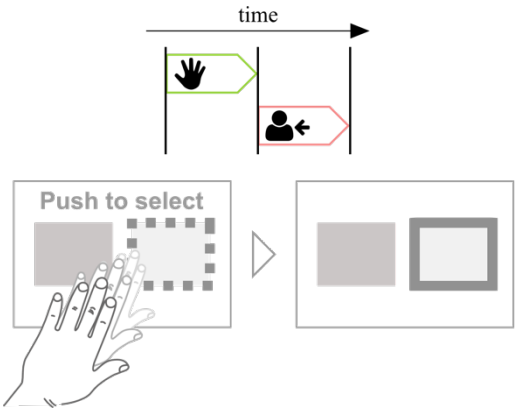


Figure 5: Discrete gesture interaction: Push gesture to select an already preselected UI element. Gesture notation (top) and exemplary pictorial representation of gesture and UI (bottom).

Additionally, the figure shows an example of a discrete gesture interaction: a push gesture to select a UI element. In the situation on the left hand, the element is highlighted and the user moves the hand towards it (like a push). While the gesture is performed the state of the UI remains unchanged, but immediately after the execution of the gesture is complete the UI element is marked to be selected (see follow-up situation shown on the right hand).

A *continuous gesture interaction* is a gesture interaction where the UI feedback is performed simultaneously with the gesture execution, i.e. *gesture.start = UI.start && gesture.end = UI.end* (see Figure 6). In this case, in contrast to a discrete gesture interaction, the UI feedback is triggered once the gesture execution starts and ends when the gesture execution is finished.

The example in Figure 6 (zooming to scale a 3D object) illustrates a continuous gesture interaction. The UI feedback (cube zooming) occurs continuously while the gesture is performed and is finished upon the completion of the gesture. In this example, not only the cube gets bigger, but also the zoom widget on the left is updated to show the current zoom level.

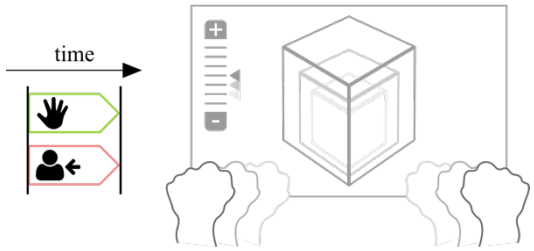


Figure 6: Continuous gesture interaction: Grab-move gesture to decrease or increase the size of a cube.

Discrete and continuous gesture interaction does not necessarily affect the internal status of the system, i.e. execution of system functionality. This is the case particularly in graphical user interfaces: A mouse movement on a virtual desktop, for example, results in UI behavior such as changing the position of the mouse cursor and highlighting an icon once the cursor enters its space. Similarly, in Figure 5 the user may select an element to apply an operation afterwards (e.g., a database update); in Figure 6 the user may enlarge an object just to have it displayed larger on the screen.

4.3 Gesture Interaction and System Functionality

As part of our notation, the above two types, discrete and continuous gesture interaction, are expanded by the aspect system functionality, resulting in four additional variants. A third arrow that contains an icon symbolizing system functionality (illustrated as a gearwheel) is used in the figures to represent the according time interval.

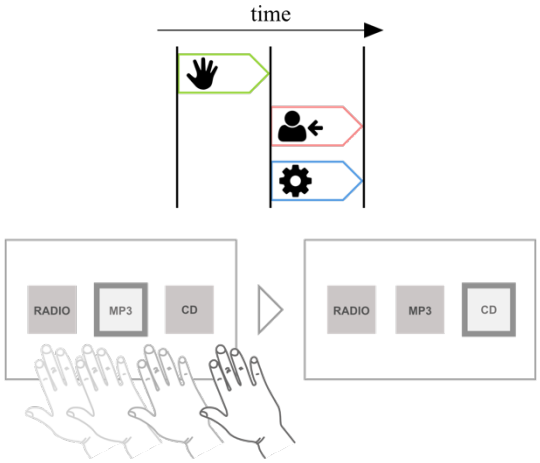


Figure 7: Discrete gesture interaction with discrete system reaction: Swipe gesture to change the input media.

A *discrete gesture interaction with discrete system reaction* is a gesture interaction in which both, the UI feedback and the system functionality are executed right after the gesture, i.e. *gesture.end = UI.start = system.start* (see Figure 7).

In the example illustrated by Figure 7 the user performs a lateral wiping movement (swipe gesture). Once *swipe_gesture.end* is detected, on the one hand, the CD element in the UI is highlighted instead of the MP3 element. The system, on the other hand, stops playing MP3s and starts to play the inserted CD.

A *discrete gesture interaction with continuous system reaction* is a gesture interaction in which the UI feedback is executed right after the gesture (*gesture.end = UI.start*), but the system functionality is executed simultaneously with the gesture (*gesture.start = system.start && gesture.end = system.end*) (see Figure 8).

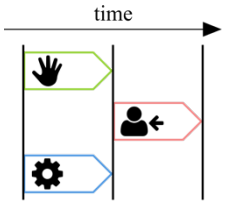


Figure 8: A discrete gesture interaction with continuous system reaction.

From our point of view, this makes hardly sense from a usability perspective (therefore, we do not provide an example here): It would violate the principle of permanent visibility of system status as published by Nielsen (1995).

A *continuous gesture interaction with discrete system reaction* is a gesture interaction with UI feedback executed simultaneously (*gesture.start = UI.start && gesture.end = UI.end*), and with system functionality executed right after the gesture, i.e *gesture.end = system.start* (see Figure 9).

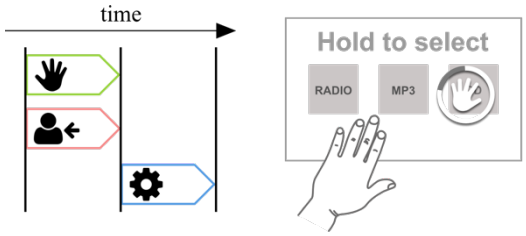


Figure 9: Continuous gesture interaction with discrete system reaction: Move-and-hold gesture to select a UI element.

An example is a selection interaction that is implemented as an overlay icon surrounded by a circle that continuously fills up over a specific period of time (see Figure 9). The user has to hold a hand over a UI element until the circle is completely filled. As soon as this is the case, the system functionality is executed (e.g. starting a CD).

A *continuous gesture interaction with continuous system reaction* is a gesture interaction in which the gesture, the UI feedback and the system functionality are executed simultaneously, i.e. *gesture.start = UI.start = system.start && gesture.end = UI.end = system.end* (see Figure 10).

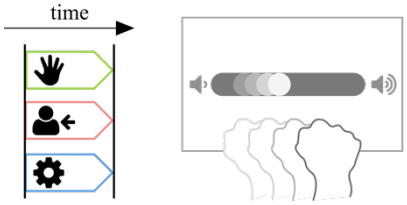


Figure 10: Continuous gesture interaction with continuous system reaction: Grab-move gesture to change the volume of a media player.

An example would be a slider that is mapped to the volume of a music player. The user can manipulate it by performing a grab gesture in combination with a horizontal move gesture (see Figure 10). During the grab-move gesture the UI updates the slider continuously showing the currently chosen volume level. At the same time the system changes the volume of the music simultaneously according to the current state of the gesture until the grab gesture has ended (e.g. by opening the hand).

5 SIMPLIFICATIONS AND EXTENSIONS

A simplification in the definitions given in section 4.2 and 4.3 is that the difference between static and dynamic gestures was not taken explicitly into account. Therefore, a gesture arrow may represent either a static or a dynamic gesture. Considering this difference in the notation doubles the gesture interaction types.

In Figure 10 a dynamic gesture is involved in a continuous gesture interaction with continuous system reaction. Replacing the gesture by a static one results in a static-continuous gesture interaction with continuous system reaction. For example, such a static gesture may be the holding hand shown in Figure 3 a) (e.g. taking into account the constraints of a reduced interaction space). To increase the

volume, the right hand is held up (see Figure 11), and to lower the volume the left hand is held up.

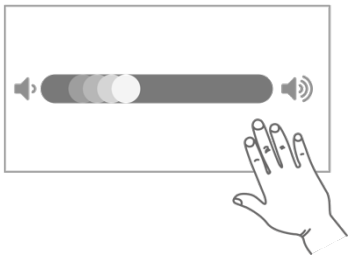
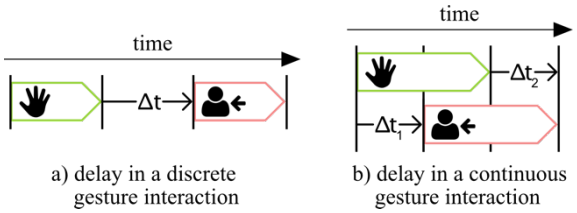


Figure 11: Static gesture with continuous UI feedback and continuous system functionality: Hold-up gesture (left or right) to change the volume (decrease or increase) of a media player.

In addition, the user may hold up the right hand although the volume has already reached the maximum. Thus, the application logic would have to ignore this gesture event since it is not meaningful for the current state of the system. In our gesture notation, such a prolonged occurrence of a gesture without consequences is not represented (the gesture arrow is not depicting the extended time interval). This is appropriate as the definition (*gesture.end = UI.end = system.end*) describes this situation correctly.

Delays, e.g. between the temporal intervals, are not considered so far. They can be added as shown in Figure 12. For information systems technically caused delays (TD) are typical. Thus, for example, a *gesture.end* does not equal *UI.start,* more precisely (*gesture.end + TD = UI.start*). It can be neglected most of the time. However, for usability reasons one may specify a maximum for TD, e.g. *gesture.end + TD = UI.start* with TD ≤ 2 seconds.



a) delay in a discrete gesture interaction

b) delay in a continuous gesture interaction

Figure 12: Delay in gesture interactions.

In addition, delays (*Δt*) are mechanisms to specify conditions and constraints, e.g. for the purpose of accessibility. For example, if users with cognitive impairments are using applications with gesture interaction, it could be necessary to add a delay between the execution of a gesture and the start of a UI feedback (*gesture.end + Δt = UI.start*) (see Figure 12 a). People with cognitive impairments might need some time to process the execution of the gesture and therefore would be confused if the UI feedback is executed right after the gesture. In this case the delay *Δt* is to be specified according to the degree of disability and may be subject of personalization.

In the case of continuous interactions two delays are introduced (see Figure 12 b). The delay Δt_1 separates the beginning of a gesture and the beginning of a UI feedback ($gesture.start + \Delta t_1 = UI.start$). The second delay Δt_2 can be used to define how long a UI feedback (e.g. an animation) is performed after a gesture movement is finished ($gesture.end + \Delta t_2 = UI.end$).

A further case of consciously specified delay is the usage of a reset timer. It can be illustrated by extending the example in Figure 9 with the option to cancel the gesture interaction. While the circle still fills up, the user may take the hand away to consciously abort the interaction. However, it may also happen that a cancelation is not wanted by the user, although he or she causes the virtual hand cursor (who's movements are mapped to the hand) to exit the interactive element. For example, the user may unconsciously turn away from the system during the activation of the UI element in order to talk to someone. Due to the user movement, the virtual cursor moves away from the UI element and the gesture interaction could be canceled. The following possibilities are therefore conceivable: The circle resets: *gesture.end = UI.start*. Or the circle does not reset for a specific time *Δt.* Subsequently, the circle resets to the initial position: *gesture.end + Δt = UI.start*.

In other works from the literature, a gesture is divided into the three phases *prestroke* (also called preparation), *stroke* (execution) and *poststroke* (retraction) (Mitra and Acharya 2007; Obaid et al. 2014; Pavlovic et al. 1997). For example, raising the hand to the start position of a swipe gesture is the prestroke, performing the swipe-movement is the stroke, and taking down the hand to a neutral position is the poststroke. Obaid et al. (2014) additionally take into account a *start-up* phase. It refers to the user's mental preparation process of preparing for a gesture and thus takes place before the prestroke. The temporal interval of a gesture of our gesture notation refers to the execution (stroke). However, it could be extended by adding separate time intervals for the pre- and the poststroke-phase. This would enable the explicit specification of conditions and constraints related to these additional phases.

6 CONCLUSION AND FUTURE WORK

This paper presented an approach to a more universal definition of gesture interactions and gesture types, respectively. Its purpose is to facilitate communication within the development process based on a common understanding of gesture terms. Indeed, in our current work on gesture-based systems, previously encountered misunderstandings could be avoided with the help of the terms and notation proposed in this paper. In addition, in our university lectures the students benefited from a resulting common understanding of the most important aspects when designing gesture interaction. All in all compared to the various taxonomies from the literature, our approach appears to us to be more practical.

In contrast to existing taxonomies our definitions distinguish between UI reactions (feedback) and system reactions (functionality). A further central extension to existing gesture taxonomies is the utilization of temporal intervals of execution of gestures (body movements), UI feedback and system functionality, and the relations between the intervals.

The previous section introduced and outlined some extensions that will be investigated in more detail in future work.

Concerning future work, the criteria shown above are to be used in more projects and evaluated in further studies and it should be checked if further dependencies between single criteria can be found. The extensions shown in this paper like further segmentation of the gesture execution and the use of delays seem particularly interesting. Furthermore, the use of further criteria (e.g. gesture styles) should be considered as another extension of our approach.

ACKNOWLEDGEMENTS

This research was financially supported by the German Federal Ministry of Education and Research within the program “Forschung an Fachhochschulen – IngenieurNachwuchs” (project no. 03FH007IX5).

REFERENCES

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. Communication of the ACM, November 1983. Vol. 26 Nr. 11, 832 – 843

Hartson, H. R., A. C. Siochi, and D. Hix. 1990. The UAN: A user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems (TOIS), 8(3), 181-203.

Hix, D. and H.R. Hartson. 1993. Developing User Interfaces: Ensuring Usability Through Product and Process. John Wiley & Sons, Inc.

Hummels, C., and P. J. Stappers. 1998. Meaningful gestures for human computer interaction: beyond hand postures. Third IEEE Int. Conf. Autom. Face Gesture Recognit.

Karam, M., and M. C. Schraefel. 2005. A Taxonomy of Gestures in Human Computer Interactions. Tech. Report, Eletronics Comput. Sci., 1–45.

Künkel, D. , Bomsdorf, B., Röhrig, R., Ahlbrandt, J., and M. Weigang. 2015. Participative Development of Touchless User Interfaces: Elicitation and Evaluation of Contactless Hand Gestures for Anesthesia. International Conferences Interfaces and Human Computer Interaction, 43–50.

Loke, L., Larssen, A.T. and T. Robertson. 2005. Labanotation for design of movement-based interaction. Proceedings of the second Australasian conference on Interactive entertainment. Creativity & Cognition Studios Press, Sydney, Australia, 113-120.

Microsoft. 2013. Visual Gesture Builder: A Data-Driven Solution to Gesture Detection. [online] Available at: http://aka.ms/k4wv2vgb [Accessed 12 Sep. 2017].

Mitra, S., and T. Acharya. 2007. Gesture recognition: A survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 37(3), 311-324.

Nielsen, J. 1995. 10 usability heuristics for user interface design. Nielsen Norman Group, vol. 1, no. 1.

Nielsen, M., M. Störring, T. B. Moeslund, and E. Granum. 2004. A procedure for developing intuitive and ergonomic gesture interfaces for HCI. Gesture-Based Commun. Human-Computer Interact., 409–420.

Obaid, M., F. Kistler, M. Häring, R. Bühling, and E. André. 2014. A Framework for User-Defined Body Gestures to Control a Humanoid Robot. International Journal of Social Robotics, 6(3), 383-396.

Pavlovic, V. I., R. Sharma, and T. S. Huang, 1997. Visual interpretation of hand gestures for human-computer interaction: A review. IEEE Transactions on pattern analysis and machine intelligence, 19(7), 677-695.

Ruiz, J., Y. Li, and E. Lank. 2011. User-defined motion gestures for mobile interaction. Annu. Conf. Hum. factors Comput. Syst. - CHI ’11, 197–206.

Saffer, D. 2009. Designing Gestural Interfaces.

Wobbrock, J. O., H. H. Aung, B. Rothrock, and B. A. Myers. 2005. Maximizing the guessability of symbolic input. In CHI'05 extended abstracts on Human Factors in Computing Systems, 1869-1872, ACM.

Wobbrock, J. O., M. R. Morris, and A. D. Wilson. 2009. User-defined gestures for surface computing. 27th Int. Conf. Hum. factors Comput. Syst. - CHI 09, 1083–1092.