Criteria, Challenges and Opportunities for Gesture Programming Languages

Lode Hoste and Beat Signer
Web & Information Systems Engineering Lab
Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium
{lhoste,bsigner}@vub.ac.be

ABSTRACT
An increasing number of today's consumer devices such as mobile phones or tablet computers are equipped with various sensors. The extraction of useful information such as gestures from sensor-generated data based on mainstream imperative languages is a notoriously difficult task. Over the last few years, a number of domain-specific programming languages have been proposed to ease the development of gesture detection. Most of these languages have adopted a declarative approach allowing programmers to describe their gestures rather than having to manually maintain a history of event data and intermediate gesture results. While these declarative languages represent a clear advancement in gesture detection, a number of issues are still unresolved. In this paper we present relevant criteria for gesture detection and provide an initial classification of existing solutions based on these criteria in order to foster a discussion and identify opportunities for future gesture programming languages.

Author Keywords
Gesture language; multimodal interaction; declarative programming.

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

INTRODUCTION

With the increasing interest in multi-touch surfaces (e.g. Sony Tablet, Microsoft Surface or Apple iPad), controller-free sensors (e.g. Leap Motion, Microsoft Kinect or Intel's Perceptual SDK) and numerous sensing appliances (e.g. Seeeduino Films and Nike+ Fuel), developers are facing major challenges in integrating these modalities into common applications. Existing mainstream imperative programming languages cannot cope with user interaction requirements due to the inversion of control where the execution flow is defined by input events rather than by the program, the high programming effort for maintaining an event history and the difficulty of expressing complex patterns.

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

http://ceur-ws.org/Vol-1190/.

For example, commercial multi-touch hardware has evolved from simple two-finger support to multi-user tracking with up to 60 fingers[1]. Similarly, commercial depth sensors such as the Microsoft Kinect were introduced in 2010 and supported the tracking of 20 skeletal joints (i.e. tracking arms and limbs in 3D space). Nowadays, numerous depth sensors such as the Leap sensors or the DepthSense cameras by SoftKinetic also provide short-range finger tracking. Recently, the Kinect for Windows supports facial expressions and the Kinect 2 supports heart beat and energy level tracking. This rapid evolution of novel input modalities continues with announcements such as the Myo electromyography gesture armband [21] and tablet computers with integrated depth sensors, fingerprint scanning and eye tracking.

In this paper, we consider a gesture as a movement of the hands, face or other parts of the body in time. Due to the high implementation complexity, most gesture recognition solutions rely on machine learning algorithms to extract gestural information from sensors. However, the costs of applying machine learning algorithms are not to be underestimated. The capture and annotation of training and test data requires substantial resources. Further, the tweaking of the correct learning parameters and analysis for overfitting require some expert knowledge. Last but not least, one cannot decisively observe and control what has actually been learned. Therefore, it is desired to have the possibility to program gestures and to ease the programming of gestural interaction. We argue that research in software engineering abstractions is of utmost importance for gesture computing.

In software engineering, a problem can be divided into its accidental and essential complexity [1]. Accidental complexity relates to the difficulties a programmer faces due to the choice of software engineering tools and can be reduced by selecting or developing better tools. On the other hand, essential complexity is caused by the characteristics of the problem to be solved and cannot be reduced. The goal of gesture programming languages is to reduce the accidental complexity as much as possible. In this paper, we define a number of criteria to gain an overview about the focus of existing gesture programming languages and to identify open challenges to be further discussed and investigated.

[1] 3M Multi-Touch Display C4667PW

MOTIVATION AND RELATED WORK

Gesture programming languages are designed to support developers in specifying their gestural interaction requirements more easily than with general purpose programming languages. A domain-specific language might help to reduce the repetitive boilerplate that cannot be removed in existing languages as described by Van Cutsem[2]. General purpose programming languages such as Java sometimes require an excessive amount of constructs to express a developer's intention which makes them hard to read and maintain. Van Cutsem argues that languages can shape our thought, for instance when a gesture can be declaratively described by its requirements rather than through an imperative implementation with manual state management. A gesture programming language can also be seen as a simplifier where, for example, multiple inheritance might not be helpful to describe gestural interaction. Finally, domain-specific languages can be used as a law enforcer. Some gesture languages, such as Proton [16], disallow a specific sequence of events simply because it overlaps with another gesture definition. It further enables the inference of properties that help domain-specific algorithms to obtain better classification results or reduced execution time.

Midas [7, 22] by Hoste et al., the Gesture Description Language (GDL) by Khandkar et al. [14], GeFormt by Kammer et al. [13] and the Gesture Description Language (GDL) by Echtler et al. [4] form a first generation of declarative languages that allow programmers to easily describe multi-touch gestures rather than having to imperatively program the gestures. This fundamental change in the development of gestures moved large parts of the accidental complexity—such as the manual maintenance of intermediate results and the extension of highly entangled imperative code—to the processing engine.

With these existing solutions, gestures are described in a domain-specific language as a sequence or simultaneous occurrences of events from one or multiple fingers. The modularisation and outsourcing of the event matching process paved the way for the rapid development of more complex multi-touch gestures. With the advent of novel hardware such as the Microsoft Kinect sensor with a similar or even higher level of complexity, domain-specific solutions quickly became a critical component for supporting advanced gestural interaction. The definition of complex gestures involves a number of concerns that have to be addressed. The goal of this paper is to enumerate these criteria for gesture programming languages and to provide an overview how existing approaches focus on different of these criteria.

CRITERIA

We define a number of criteria which can shape (1) the choice of a particular framework, (2) the implementation of the gesture and (3) novel approaches to solve open issues in gesture programming languages. These criteria were compiled based on various approaches that we encountered over the last few years, also including the domains of machine learning or template matching which are out of the scope of this paper. We aligned the terminology, such as gesture spotting and segmentation, and performed an indicative evaluation of nine existing gesture languages as shown in Figure 1. For each individual criterion of these nice approaches we provide a score ranging from 0 to 5 together with a short explanation which can be found in the online data set. A score of 1 means that the approach could theoretically support the feature but it was not discussed in the literature. A score of 2 indicates that there is some initial support but with a lack of additional constructs that would make it useful. Finally, a score in the range of 3-5 provides an indication about the completeness and extensiveness regarding a particular criterion. Our data set, an up-to-date discussion as well as some arguments for the indicative scoring for each criterion of the different approaches is available at `http://soft.vub.ac.be/~lhoste/research/criteria/images/img-data.js`. We tried to cluster the approaches based on the most up-to-date information. However, some of the criteria could only be evaluated subjectively and might therefore be adjusted later based on discussions during the workshop. An interactive visualisation of the criteria for each of the approaches can be accessed via `http://soft.vub.ac.be/~lhoste/research/criteria`. Note that the goal of this assessment was to identify general trends rather than to draw a conclusive categorisation for each approach.

Software Engineering and Processing Engine

The following criteria have an effect on the software engineering properties of the gesture implementation. Furthermore, these criteria might require the corresponding features to be implemented by the processing engine.

Modularisation

By modularising gesture definitions we can reduce the effort to add an extra gesture. In many existing approaches the entanglement of gesture definitions requires developers to have a deep knowledge about already implemented gestures. This is a clear violation of the separation of concerns principle, one of the main principles in software engineering which dictates that different modules of code should have as little overlapping functionality as possible. Therefore, in modular approaches, each gesture specification is written in its own separate context (i.e. separate function, rule or definition).

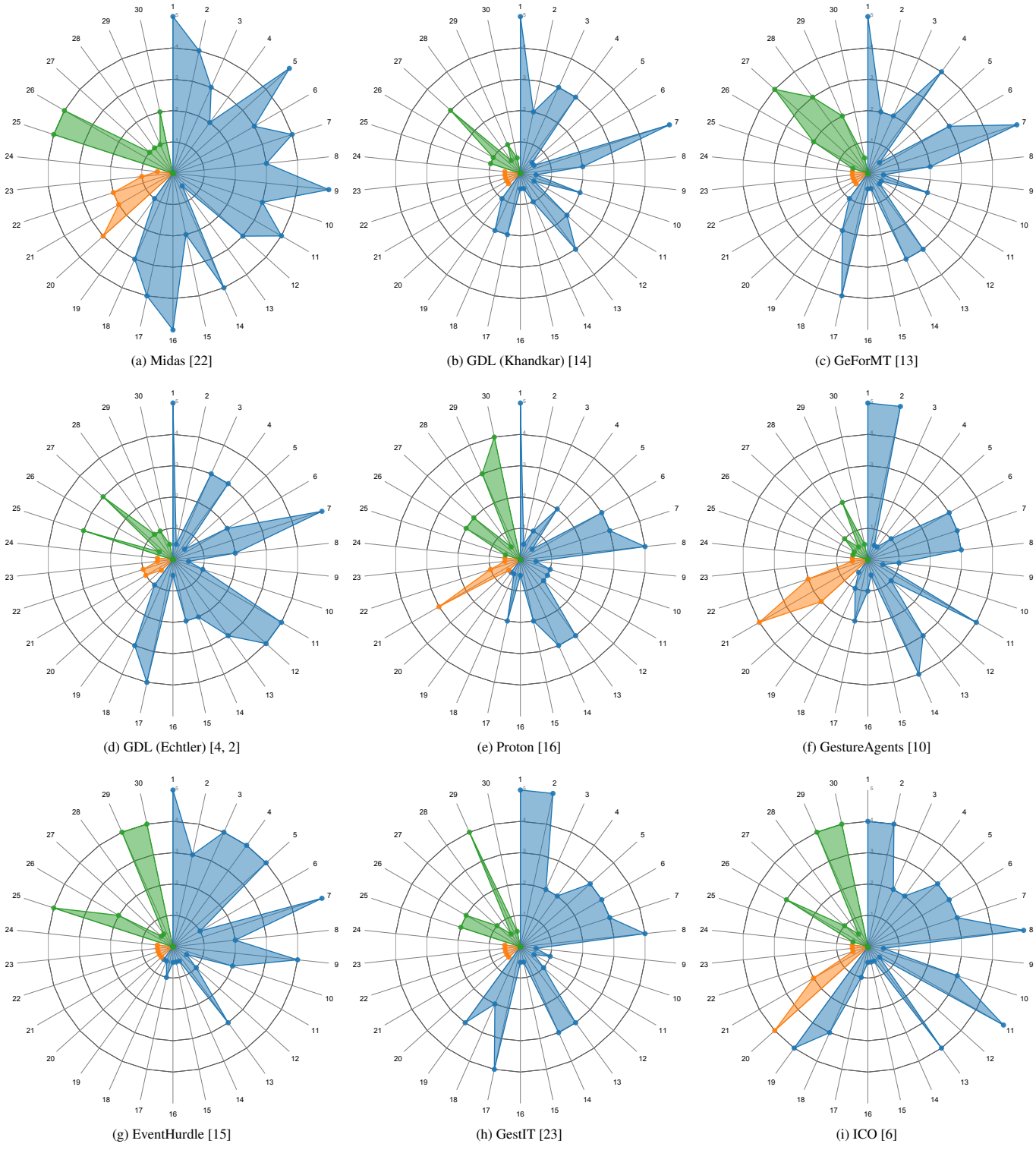
Composition

Composition allows programmers to abstract low-level complexity by building complex gestures from simpler building blocks. For instance, the definition of a double tap gesture can be based on the composition of two tap gestures with a defined maximum time and space interval between them. A tap gesture can then be defined by a touch down event shortly followed by a touch up event and with minimal spatial movement in between. Composition is supported by approaches when a developer can reuse multiple modular specifications to define more complex gestures without much further effort.

Customisation

Customisation is concerned with the effort a developer faces to modify a gesture definition in order that it can be used in a different context. How easy is it for example to adapt the

[2] `http://soft.vub.ac.be/~tvcutsem/invokedynamic/node/11`



(a) Midas [22]
(b) GDL (Khandkar) [14]
(c) GeForMT [13]
(d) GDL (Echtler) [4, 2]
(e) Proton [16]
(f) GestureAgents [10]
(g) EventHurdle [15]
(h) GestIT [23]
(i) ICO [6]

Figure 1. Indicative classification of gesture programming solutions. The labels are defined as follows: (1) modularisation, (2) composition, (3) customisation, (4) readability, (5) negation, (6) online gestures, (7) offline gestures, (8) partially overlapping gestures, (9) segmentation, (10) event expiration, (11) concurrent interaction, (12) portability, serialisation and embeddability, (13) reliability, (14) graphical user interface symbiosis, (15) activation policy, (16) dynamic binding, (17) runtime definitions, (18) scalability in terms of performance, (19) scalability in terms of complexity, (20) identification and grouping, (21) prioritisation and enabling, (22) future events, (23) uncertainty, (24) verification and user profiling, (25) spatial specification, (26) temporal specification, (27) other spatio-temporal features, (28) scale and rotation invariance, (29) debug tooling, (30) editor tooling.

existing definition of a gesture when an extra condition is required or the order of events should be changed? For graphical programming toolkits, the customisation aspect is broadened to how easy it is to modify the automatically generated code and whether this is possible at all. Note that in many machine learning approaches customisation is limited due to the lack of a decent external representation [11].

Readability
Kammer et al. [13, 12] identified that gesture definitions are more readable when understandable keywords are used. They present a statistical evaluation of the readability of various gesture languages which has been conducted with a number of students in a class setting. In contrast to the readability, they further define complexity as the number of syntactic rules that need to be followed for a correct gesture description, including for example the number of brackets, colons or semicolons. Languages with a larger number of syntactic rules are perceived to be more complex. However, it should be noted that the complexity of a language as defined by Kammer et al. is different from the level of support to form complex gestures and we opted to include their definition under the readability criterion.

Negation
Negation is a feature that allows developers to express a context that should not be true in a particular gesture definition. Many approaches partially support this feature by requiring a strict sequence of events, implying that no other events should happen in between. However, it is still crucial to be able to describe explicit negation for some scenarios such as that there should be no other finger in the spatial neighbourhood or a finger should not have moved up before the start of a gesture. Other approaches try to use a garbage state in their gesture model to reduce false positives. This garbage state is similar to silence models in speech processing and captures non-gestures by "stealing" partial gesture state and resetting the recognition process.

Online Gestures
Some gestures such as a pinch gesture for zooming require feedback while the gesture is being performed. These so-called online gestures can be supported in a framework by allowing small online gesture definitions or by providing advanced constructs that offer a callback mechanism with a percentage of the progress of the larger gesture. Note that small online gesture definitions are linked with the segmentation criterion which defines that gestures can form part of a continuous event stream without an explicit begin and end condition.

Offline Gestures
Offline gestures are executed when the gesture is completely finished and typically represent a single command. These gestures are easier to support in gesture programming languages as they need to pass the result to the application once. Offline gestures also increase the robustness due to the ability to validate the entire gesture. The number of future events that can change the correctness of the gesture is limited when compared to online gestures.

Partially Overlapping Gestures
Several conditions of a gesture definition can be partially or fully contained in another gesture definition. This might be intentional (e.g. if composition is not supported nor preferred) or unintentional (e.g. if two different gestures start with the same movement). Keeping track of multiple partial matches is a complex mechanism that is supported by some approaches, intentionally blocked by others (e.g. Proton) or ignored by some approaches.

Segmentation
A stream of sensor input events might not contain explicit hints about the start and end of a gesture. The segmentation concern (also called gesture spotting) gains importance given the trend towards the continuous capturing and free-air interaction such as the Kinect sensor and Z-touch [25], where a single event stream can contain many potential start events. The difficulty of gesture segmentation manifests itself when one cannot know beforehand which potential start events should be used until a middle or even an end candidate event is found to form the decisive gesture trajectory. It is possible that potential begin and end events can still be replaced by better future events. For instance, how does one decide when a flick right gesture (in free air) starts or ends without any knowledge about the future? This generates a lot of gesture candidates and increases the computational complexity. Some approaches tackle this issue by using a velocity heuristic with a slack variable (i.e. a global constant defined by the developer) or by applying an efficient incremental computing engine. However most language-based approaches are lacking this functionality. Many solutions make use of a garbage gesture model to increase the accuracy of the gesture segmentation process.

Event Expiration
The expiration of input events is required to keep the memory and processing complexity within certain limits. The manual maintenance of events is a complex task and most frameworks offer at least a simple heuristic to automatically expire old events. In multi-touch frameworks, a frequently used approach is to keep track of events from the first touch down event to the last touch up of any finger. This might introduce some issues when dealing with multiple users if there is always at least one active finger touching the table. Another approach is to use a timeout parameter, effectively creating a sliding window solution. An advantage of this approach is that the maximum memory usage is predefined, however a slack value is required. A static analysis of the gesture definitions could help to avoid the need for such a static value.

Concurrent Interaction
In order to allow concurrent interaction, one has to keep track of multiple partial instances of a gesture recognition process. For instance, multiple fingers, hands, limbs or users can perform the same gesture at the same time. To separate these instances, the framework can offer constructs or native support for concurrent gesture processing. In some scenarios, it is hard to decide which touch events belong to which hand or user. For example, in Proton the screen can be split in half to support some two player games. A better method is to set

a maximum bounding box of the gesture [4] or to define the spatial properties of each gesture condition. The use of GUI-specific contextual information can also serve as a separation mechanism. Nevertheless, it is not always possible to know in advance which combination of fingers will form a gesture, leading to similar challenges as discussed for the segmentation criterion where multiple gesture candidates need to be tracked.

Portability, Serialisation and Embeddability

Concerns such as portability, serialisation and embeddability form the platform independence of an approach. Portability is defined by how easy it is to run the framework on different platforms. Some approaches are tightly interwoven with the host language which limits portability and the transfer of a gesture definition over the network. This transportation can be used to exchange gesture sets between users or even to offload the gesture recognition process to a dedicated server with more processing power. The exchange requires a form of serialisation of the gesture definitions which is usually already present in domain-specific languages. The embeddability has to do with the way how the approach can be used. Is it necessary to have a daemon process or can the abstractions be delivered as a library? Another question is whether it is possible to use the abstractions in a different language or whether this requires a reimplementation.

Reliability

The dynamic nature of user input streams implies the possibility of an abundance of information in a short period of time. A framework might offer a maximum computational boundary for a given setting [19]. Without such a boundary, users might trigger a denial of service when many complex interactions have to be processed at the same time. Additionally, low-level functionality should be encapsulated without providing leaky language abstractions that could form potential security issues.

Graphical User Interface Symbiosis

The integration of graphical user interface (GUI) components removes the need for a single entry point of gesture callbacks on the application level. With contextual information and GUI-specific gesture conditions the complexity is reduced. For instance, a scroll gesture can only happen when both fingers are inside the GUI region that supports scrolling [4]. This further aids the gesture disambiguation process and thus increases the gesture recognition quality. Another use case is when a tiny GUI object needs to be rescaled or rotated. The gesture can be defined as such that one finger should be on top of the GUI component while the other two fingers are executing a pinch or rotate gesture in the neighbourhood.

Activation Policy

Whenever a gesture is recognised, an action can be executed. In some cases the developer wants to provide a more detailed activation policy such as *trigger only once* or *trigger when entering and leaving a pose*. Another example is the sticky bit [4] option that activates the gesture for a particular GUI object. A shoot-and-continue policy [9] denotes the execution of a complete gesture followed by an online gesture activation. The latter can be used for a lasso gesture where at least one complete circular movement is required and afterwards each incremental part (e.g. per quarter) causes a gesture activation.

Dynamic Binding

Dynamic binding is a feature that allows developers to define a variable without a concrete value. For instance, the x location of an event A should be between 10 and 50 but should be equal to the x location of an event B. At runtime, a value of 20 for the x location of event A will therefore require an event B with the same value of 20. This is particularly useful to correlate different events if the specification of concrete values is not feasible.

Runtime Definitions

Refining gesture parameters or outsourcing gesture definitions to gesture services requires a form of runtime modification support by the framework. The refinement can be instantiated by an automated algorithm (e.g. an optimisation heuristic) by the developer (during a debugging session) or by the user to provide their preferences.

Scalability in Terms of Performance

The primary goal of gesture languages is to provide an abstraction level which helps developers to express complex relations between input events. However, with multimodal setups, input continuously enters the system and the user expects the system to immediately react to their gestures. Therefore, performance and the scalability when having many gesture definitions is important. Some approaches, such as Midas, exploit the language constructs to form an optimised direct acyclic graph based on the Rete algorithm [5]. This generates a network of the gesture conditions, allows the computational sharing between them and keeps track of partial matches without further developer effort. In recent extensions, Midas has been parallelised and benchmarked with up to 64 cores [19] and then distributed such that multiple machines can share the workload [24]. Other approaches such as Proton and GDL by Echtler et al. rely on finite state machines. However, it is unclear how these approaches can be used with continuous sensor input where segmentation is a major issue. EventHurdle [15] tackles this problem by using relative positions between the definitions but might miss some gestures due to the non-exhaustive search [8].

Scalability in Terms of Complexity

The modularity of gestures allows for a much better scalability in terms of complexity. When adding an extra gesture, no or minimal knowledge about existing definitions is required. However, when multiple gestures are recognised in the same pool of events, the developer needs to check whether they can co-exist (e.g. performed by different users) or are conflicting (i.e. deciding between rotation, scaling or both). A lot of work remains to be done to disambiguate gestures. For example, how do we cope with the setting of priorities or disambiguation rules between gestures when they are detected at a different timestamp? How can we cope with these disambiguation issues when there are many gesture definitions? Furthermore, it is unclear how we need to deal with many variants of a similar gesture in order that the correct one is used during the composition of a complex gesture.

Gesture Disambiguation

When multiple gesture candidates are detected from the same event source, the developer needs a way to discriminate between them. However, this is not a simple task due to the lack of detail in sensor information, unknown future events and the uncertainty of the composition of the gesture.

Identification and Grouping

The identification problem is related to the fact that sensor input is not always providing enough details to disambiguate a scenario. Echtler et al. [3] demonstrate that two fingers from different hands cannot be distinguished from two fingers of the same hand on a multi-touch table due to the lack of shadowing information. Furthermore, when a finger is lifted from the table and put down again, there is no easy way to verify whether it is the same finger. Therefore, a double tap gesture cannot easily be distinguished from a two finger roll. Similar issues exist with other types of sensors such as when a user leaves the viewing angle of a sensor and later enters again. Multimodal fusion helps addressing the identification problem. The grouping problem is potentially more complex to solve. For instance, when multiple people are dancing in pairs, it is sometimes hard to see who is dancing with whom. Therefore the system needs to keep track of alternative combinations for a longer time period to group the individuals. Many combinations of multi-touch gestures are possible when fingers are located near each other.

Prioritisation and Enabling

The annotation of gestures with different priority levels is a first form of prioritisation which can have an impact on the gesture recognition accuracy. However, it requires knowledge about existing gestures and if there are many gestures it might not be possible to maintain a one-dimensional priority schema. Nacenta et al. [20] demonstrate that we should not distinguish between a scale and rotate gesture on the frame-by-frame level but by using specialised prioritisation rules such as magnitude filtering or visual handles. A developer can further decide to enable or disable certain gestures based on the application context or other information.

Future Events

One of the major issues with gesture disambiguation is that information in the near future can lead to a completely different interpretation of a gesture. A gesture definition can, for instance, fully overlap with a larger, higher prioritised gesture. At a given point in time, it is difficult to decide whether the application should be informed that a particular gesture has been detected or whether we wait for a small time period. If future events show that the larger gesture does not match, users might perceive the execution of the smaller gesture as unresponsive. Late contextual information might also influence the fusion process of primitive events that are still in the running to form part of more complex gestures. The question is whether these fused events should be updated to reflect the new information and how a framework can support this.

Uncertainty

Noise is an important parameter when dealing with gestural interaction. The jittering of multi-touch locations or limb positions might invalidate intended gestures or unintentionally activate other gestures. To analyse gestures based on imprecise primitive events, a form of uncertainty is required. This might also percolate to higher level gestures (e.g. when two uncertain subgestures are being composed). The downside is that this introduces more complexity for the developer.

Verification and User Profiling

Whenever a gesture candidate is found, it might be verified using an extra gesture classifier. An efficient segmentation approach may, for example, be combined with a more elaborate classification process to verify whether a detected recognition is adequate. Verification can also be used to further separate critical gestures (e.g. file deletion) from simple gestures (e.g. scaling). Note that couples of classifiers (i.e. ensembles) are frequently used in the machine learning domain. In order to further increase the gesture recognition accuracy, a developer can offer a form of user profiling for gestures that are known to cause confusion. Either a gesture is specified too precisely for a broader audience or it is specified too loosely for a particular user. This influences the recognition results and accidental activations. Therefore, the profiling of users by tracking undo operations or multiple similar invocations could lead to an adaptation of the gesture definition for that particular user. User profiling is valuable to improve recognition rates but it might also be interesting to exploit it for context-sensitive cases. Future work is needed to offer profiling as a language feature.

Gesture Specification

The description of a gesture requires a number of primitive statements such as spatial and temporal relations between multiple events as described in the following.

Spatial Specification

We define the spatial specification of a gesture as the primitive travelled path to which it has to adhere. The path can be formed by sequential or parallel conditions (expressed by using temporal constructs) where events are constrained in a spatial dimension such as $10 < event1.x < 50$. The use of relative spatial operators (e.g. $event1.x + 30 > event2.x$ as used in [8, 15]) also seem useful to process non-segmented sensor information. Note that approximation is required to support the variability of a gesture execution.

Temporal Specification

Gestures can be described in multiple conditions. However, these conditions cannot always be listed in a sequential order. Therefore, most gesture languages allow the developer to express explicit temporal relations between the conditions. An example of such a temporal relation is that two events should (or should not) happen within a certain time period.

Other Spatio-temporal Features

Many frameworks offer additional features to describe a gesture. For instance, Kammer et al. [13] use atomic blocks to specify a gesture. These atomic blocks are preprocessors such as direction (e.g. north or southwest) that abstract all low-level details from the developer. They can also rely on a template engine to offer more complex atomic building blocks. This integration is a form of composition and is an efficient way to describe gestures. Kinematic features can be used

to filter gestures based on motion vectors, translation, divergence, curl or deformation. Khandkar et al. [14] offer a closed loop feature to describe that the beginning and end event of a gesture should be approximately at the same location.

Scale and Rotation Invariance

Scale invariance deals with the recognition of a single gesture trajectory regardless of its scale. Similarly, rotation invariance is concerned with the rotation. Most approaches offer this feature by rescaling and rotating the trajectory to a standard predefined size and centroid. However, a major limitation is that scale and rotation invariance requires segmentation and therefore does not work well for online gestures.

Debug Tooling

In order to debug gestures, developers usually apply prerecorded positive and negative gesture sets to see whether the given definition is compatible with the recorded data. Gesture debugging tools have received little attention in research and are typically limited to the testing of accuracy percentages. It might be interesting to explore more advanced debugging support such as notifying the developer of closely related gesture trajectories (e.g. gesture a is missed by x units [17]).

Editor Tooling

Gesture development can be carried out in a code-compatible graphical manner such as done with tablatures [16], hurdles [15] or spatially [18]. When dealing with 3D input events from a Kinect, a graphical representation is valuable to get to the correct spatial coordinates.

DISCUSSION

We can identify some general trends with regard to open issues and underrepresented criteria in existing work. Figure 2 reveals that some criteria such as (1) modularisation and (7) offline gestures are well supported by most approaches and are important to provide a minimal ability to program gestures in a structured manner. Commonly used multi-touch gestures such as pinch and rotate are (6) online gestures supported by most frameworks. However, additional work can be done to streamline the implementation of online gestures by providing (2) composition support, a deeper (14) GUI integration and more advanced (16) activation policies. We believe that these challenges can be resolved with additional engineering effort in existing systems.

However, we see that more challenging issues such as (9) segmentation, (19) scalability in terms of complexity or dealing with (22) future events are poorly or not at all supported in existing approaches. (9) Segmentation is crucial to deal with continuous streams of information where no hints are given by the sensor with regard to potential start and end conditions. The recent trend towards near-touch sensors and skeletal tracking algorithms makes the segmentation concern of crucial importance.

With the adoption of the discussed frameworks there is an increasing demand to deal with (19) scalability in terms of complexity. It is currently rather difficult to get an overview on how many gestures work together. For instance, when a two-finger swipe needs to be implemented by composing two

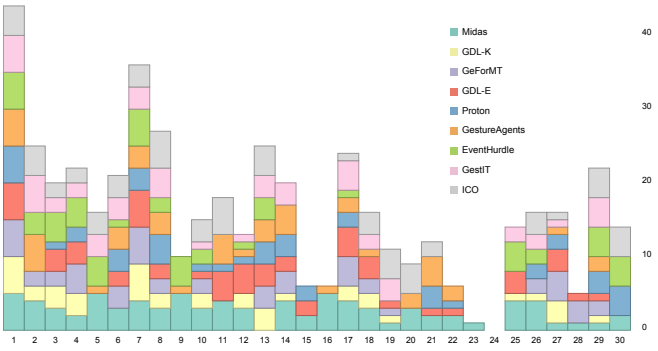


Figure 2. A stacked bar chart of the summed values for each criterion

primitive swipes. However, more than one primitive swipe variation might exist (long/short, fast/slow) and choosing the correct one without introducing conflicts is challenging. A way to deal with the issue is to provide test data but this is work intensive. There might be software engineering-based abstractions that could improve this situation.

Information coming from (22) events in the near future can lead to a completely different interpretation of a gesture. Especially in a multimodal context where context or clarifying information comes from additional sensors. An active gesture might fully overlap with a larger and higher prioritised gesture or be part of a composition that might or might not succeed. In other cases the gesture should not be triggered at all due (late) context data. There are currently no adequate abstractions helping developers to deal with these concerns.

Finally, we would like to highlight that (23) uncertainty and (24) user profiling abstractions are also lacking. Dealing with uncertainty is currently completely hidden from the programmer. Nevertheless, a gesture might be better neglected when composing from two or more uncertain subgestures. Additionally, when a user consistently undoes an operation executed by a particular gesture, the gesture might require some adjustments. This can be checked by profiling users and by offering several resolutions strategies.

CONCLUSION

Gesture programming languages allow developers to more easily express their interaction patterns. When designing such a language, a number of concerns need to be addressed. Our goal was to categorise and explicitly expose these design decisions to provide a better understanding and foster a discussion about challenges, opportunities and future directions for gesture programming languages. We observed a number of underrepresented concerns in existing work and highlighted challenges for future gesture programming languages.

REFERENCES

1. Brooks, Jr., F. P. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer 20*, 4 (April 1987), 10–19.
2. Echtler, F., and Butz, A. GISpL: Gestures Made Easy. In *Proceedings of TEI 2012, 6th International Conference on Tangible, Embedded and Embodied Interaction* (Kingston, Canada, February 2012), 233–240.

3. Echtler, F., Huber, M., and Klinker, G. Shadow Tracking on Multi-Touch Tables. In *Proceedings of AVI 2008, 9th International Working Conference on Advanced Visual Interfaces* (Napoli, Italy, May 2008), 388–391.

4. Echtler, F., Klinker, G., and Butz, A. Towards a Unified Gesture Description Language. In *Proceedings of HC 2010, 13th International Conference on Humans and Computers* (Aizu-Wakamatsu, Japan, December 2010), 177–182.

5. Forgy, C. L. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence 19*, 1 (1982), 17–37.

6. Hamon, A., Palanque, P., Silva, J. L., Deleris, Y., and Barboni, E. Formal Description of Multi-Touch Interactions. In *Proceedings of EICS 2013, 5th International Symposium on Engineering Interactive Computing Systems* (London, UK, June 2013), 207–216.

7. Hoste, L. Software Engineering Abstractions for the Multi-Touch Revolution. In *Proceedings of ICSE 2010, 32nd International Conference on Software Engineering* (Cape Town, South Africa, May 2010), 509–510.

8. Hoste, L., De Rooms, B., and Signer, B. Declarative Gesture Spotting Using Inferred and Refined Control Points. In *Proceedings of ICPRAM 2013, 2nd International Conference on Pattern Recognition Applications and Methods* (Barcelona, Spain, February 2013), 144–150.

9. Hoste, L., and Signer, B. Water Ball Z: An Augmented Fighting Game Using Water as Tactile Feedback. In *Proceedings of TEI 2014, 8th International Conference on Tangible, Embedded and Embodied Interaction* (Munich, Germany, February 2014), 173–176.

10. Julia, C. F., Earnshaw, N., and Jorda, S. GestureAgents: An Agent-based Framework for Concurrent Multi-Task Multi-User Interaction. In *Proceedings of TEI 2013, 7th International Conference on Tangible, Embedded and Embodied Interaction* (Barcelona, Spain, February 2013), 207–214.

11. Kadous, M. W. Learning Comprehensible Descriptions of Multivariate Time Series. In *Proceedings of ICML 1999, 16th International Conference on Machine Learning* (Bled, Slovenia, June 1999), 454–463.

12. Kammer, D. *Formalisierung gestischer Interaktion für Multitouch-Systeme*. PhD thesis, Technische Universität Dresden, 2013.

13. Kammer, D., Wojdziak, J., Keck, M., Groh, R., and Taranko, S. Towards a Formalization of Multi-Touch Gestures. In *Proceedings of ITS 2010, 5th International Conference on Interactive Tabletops and Surfaces* (Saarbrücken, Germany, November 2010), 49–58.

14. Khandkar, S. H., and Maurer, F. A Domain Specific Language to Define Gestures for Multi-Touch Applications. In *Proceedings of DSM 2010, 10th Workshop on Domain-Specific Modeling* (Reno/Tahoe, USA, October 2010).

15. Kim, J.-W., and Nam, T.-J. EventHurdle: Supporting Designers' Exploratory Interaction Prototyping with Gesture-based Sensors. In *Proceedings of CHI 2013, 31th ACM Conference on Human Factors in Computing Systems* (Paris, France, April 2013), 267–276.

16. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton: Multitouch Gestures as Regular Expressions. In *Proceedings of CHI 2012, 30th ACM Conference on Human Factors in Computing Systems* (Austin, Texas, USA, November 2012), 2885–2894.

17. Long Jr, A. C., Landay, J. A., and Rowe, L. A. Implications for a Gesture Design Tool. In *Proceedings of CHI 1999, 17th ACM Conference on Human Factors in Computing Systems* (Pittsburgh, USA, 1999), 40–47.

18. Marquardt, N., Kiemer, J., Ledo, D., Boring, S., and Greenberg, S. Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TouchID Toolkit. Tech. Rep. 2011-1004-16, Department of Computer Science, University of Calgary, Calgary, Canada, 2011.

19. Marr, S., Renaux, T., Hoste, L., and De Meuter, W. Parallel Gesture Recognition with Soft Real-Time Guarantees. *Science of Computer Programming* (February 2014).

20. Nacenta, M. A., Baudisch, P., Benko, H., and Wilson, A. Separability of Spatial Manipulations in Multi-touch Interfaces. In *Proceedings of GI 2009, 35th Graphics Interface Conference* (Kelowna, Canada, May 2009), 175–182.

21. Nuwer, R. Armband Adds a Twitch to Gesture Control. *New Scientist 217*, 2906 (March 2013).

22. Scholliers, C., Hoste, L., Signer, B., and De Meuter, W. Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of TEI 2011, 5th International Conference on Tangible, Embedded and Embodied Interaction* (Funchal, Portugal, January 2011), 49–56.

23. Spano, L. D., Cisternino, A., Paternò, F., and Fenu, G. GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition. In *Proceedings of EICS 2013, 5th International Symposium on Engineering Interactive Computing Systems* (London, UK, June 2013), 187–196.

24. Swalens, J., Renaux, T., Hoste, L., Marr, S., and De Meuter, W. Cloud PARTE: Elastic Complex Event Processing based on Mobile Actors. In *Proceedings of AGERE! 2013, 3rd International Workshop on Programming based on Actors, Agents, and Decentralized Control* (Indianapolis, USA, October 2013), 3–12.

25. Takeoka, Y., Miyaki, T., and Rekimoto, J. Z-Touch: An Infrastructure for 3D Gesture Interaction in the Proximity of Tabletop Surfaces. In *Proceedings of ITS 2010, 5th International Conference on Interactive Tabletops and Surfaces* (Saarbrücken, Germany, November 2010), 91–94.