International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 4 143 – 147

Event Transformation for Browser Based Web Devices

Dilip Prajapati
M Tech,
SEC Sikar
prajapati.dilip@gmail.com

Dr.Sudhir Rathi
Professor,
SEC Sikar
rathisudhir@gmail.com

Deepika Gupta
Assistant Professor,
CET Bikaner
deepika.gupta1218@gmail.com

Mrs. Manju Kumari
Lecturer, Computer Engineering
Govt. Polytechnic College, Bikaner
manjusunita08@gmail.com

***Abstract*—**Today a smartphone or tablet supports seven to eight ways by which user can interact with it. These interaction methods are touch, mouse, keyboard, voice, gestures, hover & stylus. Future is going towards IoE (Internet of everything) but if we really want to realize this vision then we need someone who can deal with these various existing and upcoming device interaction methods.

This paper talks about a custom JavaScript library, which is accountable for registering native events coming from different event sources and maps it with the user defined key map to form a proper gesture. It is not a plain mapping because it takes care of many parameters like event state, occurrence, time interval of key press etc. If the events are coming from touch screen device then complexity increases many folds because forming a touch gesture involves all mathematical steps related to identification of swipe direction. Also in order to support the acceleration, its required to know till how long key was pressed and when it was released else no gesture will be formed and all events will be discarded. Based on device capability supported events could be discarded to completely knock off a device interaction method. It could be touch, mouse or key anything. This paper investigates heterogeneity of device interaction method events to form uniform gestures so that application developer need not to write code for each and every device interaction method.

Keywords*—Gestures, Event transformation, Event simulation*

*****

I. INTRODUCTION

Form the introduction of the touch screen devices many companies followed suit and started to introduce touch screen inputs for their mobile device. Mobile & Tablet touch screen devices are becoming more and more common in our daily lives. Within the domain of human computer interaction, touch has been considered an interaction medium for modern generation. Until recently however, it was limited to recognizing single touch interactions such as selecting menu options or entering numbers in systems in banks, stores, etc. Touch was basically treated as a replacement of mouse.

Recent past innovations in multi-touch devices, have initiated new opportunities for computer interaction that are fundamentally intuitive and natural. Comparatively, multi-touch is a newer interaction technique, where different types of touches including more than one fingers, hands or arbitrary objects, can be used to interact with a system. While research to find the most suitable multi-touch technology is in progress, a number of devices are available that use different approaches to support this form of interaction. One of the majorchallenges of handheld devices is that they have to be tiny so they are limited to a smaller screen. Even though high resolution displays built into digital gadgets enable a larger workspace for applications, only some limited amount of information is displayed on a screen at any given time.The simple method to display high volume of information on the screen is to display the content in full resolution and let the user zoom and tap to the area of interest. It has become a very common activity to perform zoom gesture to check or search information of the interest.

In current world mouse and keyboard provide the means of virtually all input. Use of other options than keyboard or mouse which is grasping virtual objects, head, hand or body gesture, eye fixation tracking are getting popular with popularity of ubiquitous and ambient devices. We will see more elderly and fewer younger people as a process of huge demographic change. This population will continue to increase significantly in the future. It is widely accepted that we need to address this issue through more research work [1]. This paper reviews some studies regarding device interaction methodologies. From the days of microprocessors and controllers we have been interacting with switches. With the growing phase of embedded systems these switches have been taken a standard form keyboard. Its one of the mostly used way of device interaction and still almost many of our household digital gadgets supports it. For the developer community writing code in C/C++ to handle each state of keyboard key was not easy. It used to require proper code design, thanks to hierarchical state charts [2]. It gave a new direction to developer community so that they can write maintainable code [3].

Introduction to computer systems introduced us to new device interaction method “Mouse”, from the initial days of its usages; it has become such a popular device interaction method. Not only it provides the ease to select small objects on

screen but at the same time introduces to gestures for various user operations. Due to mouse only DoS based 2D games got huge popularity. In current digital edge the king of device interaction method is touch. Touch has changed the entire way human used to interact with devices. Touch has given new dimension to device interaction. It's fast easy convenient and more intuitive that is attracting its usages in all segments of digital devices. It is not in scope of this paper to conduct a detailed survey of the history of touch based interactions. Rather, I would like to highlight two important changes of gesture based interactions in the history. First is the shift from pen gestures to finger gestures and the second is the change from stroke gestures to multi-touch gestures. Stylus was the first medium for drawing stroke gestures on touch screens, before this stroke gesture research
focused on the digital pen as the drawing device and most stroke gesture HCI research work published to date, such as [4, 5, 6] has been based on data collected from gestures produced with high quality inductive digital styli. However, recent commercial product design avoids the use of the pen for user convenience and simplicity. Hence, a major recent focus in gesture design refers to finger gesture design, as well as the combinational use of finger and pen gestures. This kind of shift raises several research questions regarding gesture-based interactions.

Past surface gesture studies focused on stroke gestures, which are usually drawn by a pen or a finger. Advances in touch screen allow users to draw touch gestures with multi fingers or combinational input styles with pens and fingers. Higher degree of freedom enables the user to easily perform some gestures, such as using zooming gesture to enlarge targets. Using multi fingers to perform such gestures is consistent with user experiences in real world situations. This input attribute meets the requirement of NUI. Therefore, multi touch gestures are becoming more and more widely used in commercial products. Also, the research of touch based gesture interactions became interested in multi-touch/sketch gestures. The above two changes indicate that gesture interactions, as a natural and convenient interaction style, are gaining popularity in interactive device design. Sketching and writing are natural activities in many cases. Using pen and paper, a person can quickly write ideas and draw pictures and diagrams, deferring details until later. The informal nature of pens allows user to focus on their task without having to worry about precision. However, more computing devices are coming facilitated with pens, there are few useful pen-based applications out there that take benefit of the fact that pens are nice for sketching. Most applications use pens only for dragging, tapping, and selecting. These applications simply consider the pen as another pointing device.

Increasingly our surroundings is populated with a variety of intelligent devices, each groomed in a particular function. The modern drawing room, for example, typically have a HD television, amplifier, DVD player, lights, and so on. In the future, we can look forward to these gadgets becoming more interconnected, more types and more specialized as part of an increasingly sophisticated and integrated intelligent environment. This presents a challenge in designing good user experience & interfaces. For example, today's drawing room coffee table is typically cluttered with many user interfaces in the form of infrared remote controls, each equipped with many buttons. Seldom each of these interfaces controls a single device, and requires the user to pay attention to finding the right button rather than attending to the device under control. Future's intelligent eco system environment presents the chance to present a single intelligent user interface to control many such gadgets when they are networked. Web applications are fundamentally reactive. Code in a web page runs in reaction to events, which are triggered either by external interaction or by other events. The DOM, which defines these behaviors, is therefore central to the behavior of web applications. Modern web applications are fluid collections of script, styles and markup that react and adapt to user interaction. Because their programming model differs from traditional desktop applications. To date, most efforts have directed on individual portions in isolation, notable progress has been made in clarifying the semantics of JavaScript, in modeling the tree structure of HTML, and in understanding the overall behavior of the web browser as a runtime environment. Web programming is basically event driven, and employs a powerful mechanism for event propagation. Perhaps counter intuitively, the script loaded in web applications is largely inactive, and only run when triggered by events dispatching through the HTML structure in which it resides. Events dispatching through the HTML structure in which it resides. Web apps are large codebases in languages with (currently) poor support for modularity. An intuitive but not complete model for program web pages is that of an asynchronous event loop. In this model, events will be triggered by user interaction, and event callbacks have access to an object graph representing the HTML, known as the Document Object Model (DOM).

II. PROPOSED METHODOLOGY

Event Manager is a very lightweight, simple, highly configurable and thought out script for handling browser events. In order to address the heterogeneity of device interaction methods, we have a solution and its Event Transformer. Event Transformer works like a translator between application and platform. Based on interaction methods supported by device's platform, event transformer can be configured and ready to use. Now applications just need to rely on event transformer events, so no need to include code in your application for various event/interaction sources. The best

part is it's written in Java script so compatible with all browsers so all internet enabled devices can make use of it.

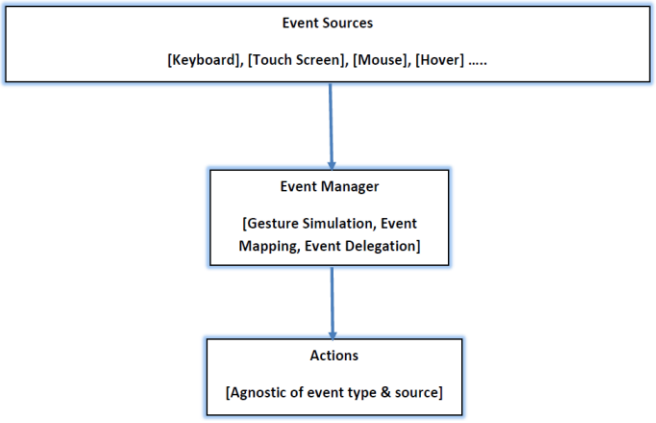


Fig. 1. Basic concept of Event Transformer

In order to perform event transformation in actions, it passes through many modules of event transformer. Some of them are described below. If a developer wants to handle both raw as well as actions delivered via event transformer, it is possible. It makes event transformation design compatible with old codes and libraries.

A. *Event Observer*

Event observer has collection of event listeners for various event sources enabled via event transformation configuration files. Always event observer listen all events in event propagation phase.

B. *Type Detector*

After the event observation, it will be passed to type detector. Type detector analyzes and parse event object to identify the event type. In browser event detection is really important as different browser sends different event details as part of event object.

C. *Event Mapper*

Based on event type, event mapper maps the current event with the configured action type and action code. In case of event sources other than touch events gestures simulated by gesture manager will be mapped with actions as per configured in action map.

D. *Gesture Manager*

In case of touch, mouse and hover series of event will be observed and detected to form best possible gesture. Gesture manager performs all such mathematics like angle and direction calculations to form best possible gesture. All gesture formation will be based on gesture confirmation done by application developer.

E. *Event Delegator*

Once event will be mapped against action then event delegator creates new action event and embeds some pre computed information. It helps developers to handle complex UI scenarios in coding.

There are several configuration maps are available as part of event transformer. With the help of these maps developers can define and control various features of event transformer. To make it compatible with latest technology, format of configuration is JavaScript object notation. In case developer misses anything in configuration maps then event transformer performs all calculations based on default settings.

A. *Action Map*

It maps key codes with actions. In case of touch events it maps gestures with actions. Action map is also useful in case of making a key as hot key. Long key press event can also be configured with a particular action. Grouping of events to an action can be done as part of action map.

B. *System Map*

It maps platform notifications with action notifications. In case there is any change at web kit level, related to any platform notification structure then this map is really useful because by just making some changes to system mapapplication can run (no need to make any code level changes).

C. *Source Map*

It helps in enabling and disabling a particular event source. In order to get event transformer actions at least one event source should be enabled. As of now five event sources keyboard, mouse, touch, system and orientation are enabled. In future more event sources could be added to event transformer and same can be enabled or disabled via the source map.

D. *Constants Map*

All kind of configuration related to gesture tuning could be part of constants map. As of now long key press interval, minimum swipe length, velocity levels, tap & hold interval and swipe interval are part of constants map. Under the constants map we do have option to enable DOM events as well.

 ISSN: 2321-8169
143 – 147

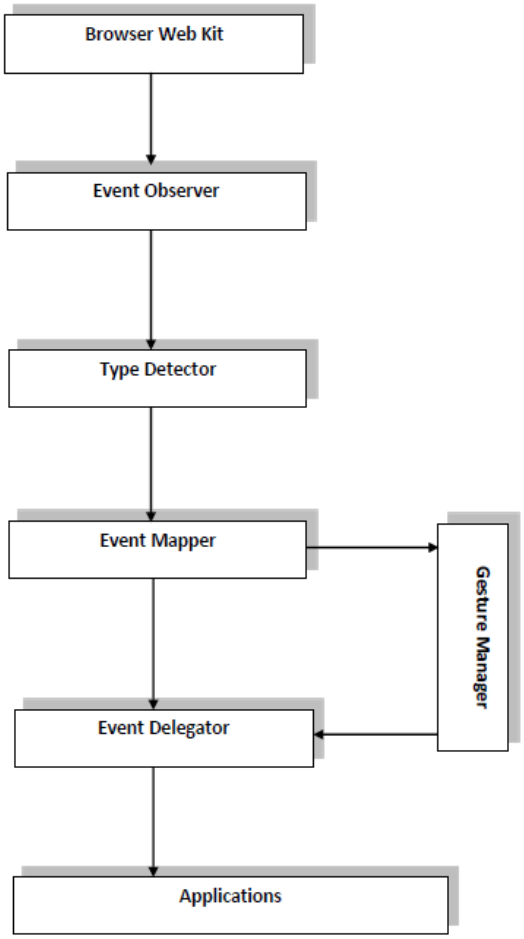
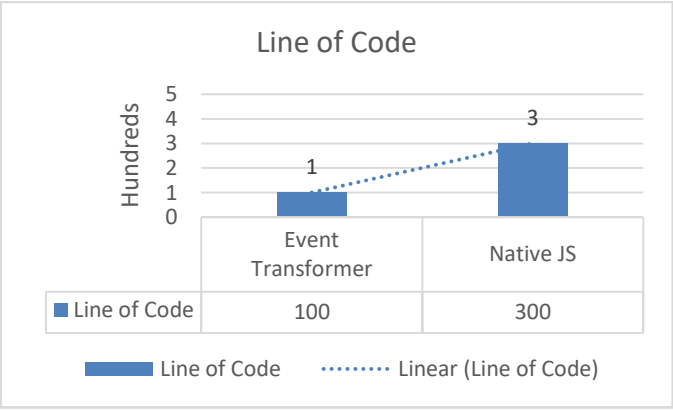


Fig. 2. Architecture of Event Transformer

III. Results

As per comparative study proposed in previous chapter event transformer is not a kind of event library but it enables developers to get rid-off heterogeneity introduced by various events sources like touch, mouse, hover, keyboard etc. None of the event library is providing this unique quality which allows developers to write applications, agnostic of event sources.



The above plotted graph provides a fair idea that what benefit we are getting out of event transformer over native java script. Consider a web application supporting three event sources keyboard, mouse and touch. When we use event transformer then approximately we need to write 100 lines of code to handle all the normal use cases were as if we use native java script then it goes minimum 300 lines of code to suffice same set of use cases. With these results we can say that if complexity increases some more folds then definitely developer will end up in writing puzzled and messy code. When we use event transformer then DOM event processing happens two times. First all event captured by event transformer lib and then it will be captured by application. Event transformer takes less than 50 milliseconds to form an action event and then its matter of application to consume it. Same will be the case when we use native java script as event will be captured directly by the application and proper interpretation will happen to take appropriate action so introducing event transformer in web application doesn't cause any kind of increment in time complexity of the application responsiveness.

IV. Conclusion

The paper investigated transformation of events generated by different event sources in actions and its uniform handling at application level. An action also contains dynamic information to implement animations as part of user interface.Application developers need to write code only for actions. Different event sources can be integrated with event transformation library. Currently keyboard, remote, mouse and touch event sources are integrated with it. In case of touch events only basic actions are supported but in order to suffice requirement of recent user interface trends, it's possible to plug-inany third party touch library for rapid application development.The best part is event transformation action follows W3C DOM3 event specifications, so web application developer need not to learn anything about actions provided by event transformation. Animations have become integral part for almost all user interface paradigms so in order to provide ease to implement animations, event transformation is very useful as it calculates and adds animation related information with all actions. Miscellaneous use case related to keyboard and remote control are also handled gracefully by event transformation library are event grouping, hot keys handing, long key press, hot key's long key press, WAC notifications, event map and key code variations. Key codes for remotes keys could be different from platform to platform but event transformation configuration helps to deliver uniform key codes. Overall event transformation looks promising in resolving heterogeneity of handling events coming from various event sources.

V. Future Work

We only explored the possibility of efficient event transformation of touch, mouse & keyboard events in actions

so that application views should be agnostic of device interaction methods. In near future many new device interaction methods are coming some of them are finger hover and stylus hover, voice recognition, air & eye gestures. It would be good and challenging to understand event provided by these event sources and their integration with event transformation library. If it goes fine then application developer will love it as they need not to write code for such device interaction methods and at the same time application footprint and complexity will also be less. It also need to be analyzed further that with various event sources till what level, we can resolve the heterogeneity of event handling mechanism, for example some actions between keyboard events and touch events are common but a wide range of actions generated by touch events will be completely different from keyboard events like pinch & zoom. Many open sources libraries are available for specific event sources to form gestures and evolving day by day and getting popularity as well. It would be also interesting to see how easy it would be to integrate those libraries with event transformation library to benefit wide range of developer community.

REFERENCES

[1] M. Bhuiyan and R. Picking, "A Gesture Controlled User Interface for Inclusive Design and Evaluative Study of Its Usability," Journal of Software Engineering and Applications, Vol. 4 No. 9, 2011, Doi: 10.4236/jsea.2011.49059, pp. 513-521.

[2] Harel, David, "State charts: A Visual Formalism for Complex Systems", Science of Computer Programming, 8, 1987, Elsevier North-Holland, Inc, Doi: 10.1016/0167-6423(87)90035-9, pp. 231-274.

[3] Samek, Miro, Practical State charts in C/C++: Quantum Programming for Embedded Systems, CMP Books, 2002, ISBN: 1-57820-110-1.

[4] Andersen and Zhai “Writing with music: exploring the use of auditory feedback in gesture interfaces”, ACM TAP Volume 7, Issue 3 (June 2010) Article No. 17, doi: 10.1145/1773965.1773968, pp. 1-24.

[5] Appert and Zhai, “Using strokes as command shortcuts: cognitive benefits and toolkit support”, In Proc. CHI 2009 of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press (2009), ISBN: 978-1-60558-246-7, Doi: 10.1145/1518701.1519052, pp. 2289-2298.

[6] Bau and Mackay, “A dynamic guide for learning gesture-based command sets”, In Proc. UIST 2008, ACM Press (2008), ISBN: 978-1-59593-975-3, Doi: 10.1145/1449715.1449724, pp. 37-46.