MASTER'S THESIS

A Generic Event Logger For Real-Time, Web-Based Applications

Timelines, Selection & Intelligent Replay

Supervisor: Prof. Dr. Anton Eliens

Leonidas Diamantis
l.diamantis@student.vu.nl
VU University 2014

Chapter 1: Introduction

The concept of posterior analysis of system behaviors is essential in the world of science, inspiring and concerning multiple fields from Mathematics to Computer Science and Business Intelligence, stimulating people to consider & invent advanced techniques and methods which would help them analyze the behavior of certain systems and draw useful conclusions about how they (mis)behave.

The whole process of such analysis boils down to one main task: extracting information from a dataset with data specific to the system's execution, and performing different kinds of analyses to determine how that system behaved, based on some criteria. Obviously, the nature of this dataset as in what types of data it carries will produce different kinds of conclusions at different depths. With the most common approach (from a Computer Science perspective) on this matter being that such information set would mainly include raw, computationally produced data (e.g the sum of two numbers for an addition application) the conclusion-making process is bound to follow specific analysis patterns and produce conclusions following respective patterns as well.

The system built for this thesis comes to offer an alternative approach to the one described above, in respect to the creation of the dataset: for our scope, we focus on capturing the actual interaction between user and application and creating a dataset which, instead of the data that the application produces, consists of meta-information about how has the application been made use of, from the user's standpoint. And with the (selective) replay features our system implements, the conclusion-making process for the application's behavior could be done by examining exactly what kind of usage made the system behave in the way it did.

Motivations

The research carried out for this Thesis was motivated by two main domains: computational analysis & the digital arts.

While considering the scope & approach for this project, we considered various use-case scenarios and types of systems which would render such a logging system useful. We considered collaborative applications which impose distortions in the communication between participants and various distributed systems which have the tendency to fail on arbitrary moments and modes. For such scenarios, we thought that such a system would make “the building blocks of determining what went wrong in a distributed system”, from a

user's perspective, as the information in which the logger is interested is the actual usage that made the application behave in the (possibly faulty) way that it did.

As a counterpart to the strict, Computer Science driven domain, we also considered a more abstract platform where a generic event logging system would bring some value. Inspired from time & synchronization oriented artistic performances, we gained motivation to create a system which would help "use the past to shape the future", in the sense that things that happened could (selectively) re-happen, possibly extended or manipulated with an artistic twist.

Research Questions, Scope & Approach

The ways we chose to approach this problem are mostly based in exploration. First, we considered a number of real-life scenarios and metaphors where we believe that such a system would be useful and capable of producing valuable results. We created a number of simple applications to represent these scenarios and metaphors, and built the logging system around them, always keeping in mind that it has to be generic so we tried not to bind the system to the applications in ways that it would become an application-specific plugin. Our purpose was to test the flexibility of the platform (the Web), the levels of abstraction and generic representation that such a system can achieve and the degree of how can such a system stand in a distributed environment.

The two main research questions of this thesis are:

- How and in what depth can the information of user interaction be captured as a generic entity?
- What reliable way is there for the logged user interaction to be replayed and produce a consistent behavior of the system?

For the scope of this project, even though we provide insights on how could this information be used later on, we do not focus on the conclusion-making process as much as creating the platform for logging such interaction events & producing such datasets, and ultimately offering features which would re-produce the occurrence of these events in a flexible way. Furthermore, the platform of choice for this thesis is the Web, focusing on Web-based applications which could be used by one or multiple users, possibly in a real-time manner. The reasons for forming the scope as such are the following:

- The Web has evolved in ways that made it a platform for purely distributed systems.

- The feature sets of Web applications and the services they provide become richer and more popular every day, making the Web the prominent platform for most types of Internet usage.
- There has been a considerable amount of research for capture/replay techniques for stand-alone applications (mostly focused on Java & JVM-based technologies), yet there is not much research done in that respect for the ever-growing Web platform.

Thus, it is reasonable to say that the scope we define covers some cutting-edge, real-world systems with a highly distributed nature in all senses and allows us to touch upon fresh ground to research and conclude with interesting insights for the future.

Outline

Chapter 2 aims to position the topic of this project in a theoretical space by discussing the major components of the research and by elaborating on the different scenarios which were taken into account for designing and implementing the logging system.

Chapter 3 focuses on the technical aspect of the project, explaining the architecture design of the system, the technologies which were used and the reasons why those technologies were selected and examining the techniques that were implemented on each phase and use case. Additionally, it discusses issues that were raised by the distributed nature of the system & the application and ways they can be tackled.

In Chapter 4, we look upon some efficiency considerations with regards to the performance of the system in various cases. Chapter 5 provides a series of insights for creating a more generic and performant event logging system, by examining specific areas in which it can be improved.

The thesis ends with Chapter 6 where we present a list of ideas for future work regarding the use of such a system's results and datasets, along with certain features that would transform it into a generic, domain-agnostic, cutting edge tool. Furthermore, we reflect upon the contributions of this research, followed by our final conclusions.

Chapter 2: Background

In this chapter, we try to provide a theoretical background about the system that was built for this project. We explain the working principles of the major components of the logger, define the basic entities of interest and sketch the high level scenarios & criteria we de-

cided to investigate. We also give an overview of the application(s) we built for exploring the features of the logging system, and attempt to enrich the application concepts with some metaphor(s) from real life, allowing the reader to place the event logger in a context more tangible than pure, abstract academic research.

Definition Of The Event

An event, as we decide to define it in the context of this project, is the product of any action or interaction which will provide an application with enough input to stimulate it to behave in the manner it was designed for. For this project, the event is the quintessential piece of information, as it is what we are interested in logging and reproducing while in the replay phase. Through streams of events, or *timelines* as they are also interchangeably referred to in this document, we get the picture of *"what happened"* while the application was used.

Since the event logger operates in the Web realm, we expect the events we are dealing with to belong to a -broad, yet apparent- scope. For example, we know a-priori that the typical target element of an (inter)action occurrence will most probably be a DOM element, and that the primitive types of the events themselves (as objects or entities) will follow the computational scheme & representation prototype of a Web-interpreted language (e.g. JavaScript). In a later chapter we discuss concepts and cases where the event object could be extended for different (more advanced) kinds of logging and replay, but for the sake of consistency with the design principles of the logging system, at this stage we will focus on the main characteristics of the event object which enable the logger to perform some meaningful work.

The main characteristics (or properties, since we are treating the event as an object) of an event that we are interested in are the following:

- **Type**, which indicates whether the event is the press of a keyboard button, a mouse click or movement, a native JavaScript event or any other event type that is relevant to the application and the logger.
- **Timestamp**, is the real UNIX timestamp of when the event occurred, typically measured at the source.
- **Relative Timestamp**, indicating the time difference between the occurrence of the current event and the previous one.

- **Source**, and identifier of the party that initiated the event occurrence.
- **Target**, which in turn is an object describing the (DOM) element of the application upon which the (inter)action took place. The target's properties are:
  - **ID**, the HTML id of the element (if it exists), given that it is a DOM object.
  - **Class**, the list of HTML classes of the element (if they exist), given that it is a DOM object.
  - **Tag-name**, the type of the HTML target element (div, form, etc).
  - **Text-content**, the inner content of the HTML target element, if it contains any.
  - **Type**, the value of the HTML *type* attribute of the target element, if it contains such an attribute (this attribute is usually a property of HTML form elements).
  - **Outer-HTML**, which is the source HTML code which describes the rendered target element.

Many of the event properties are optional, especially the ones which refer to the target. We want that for two main reasons: (1) the Web is a dynamic environment, and Web development is quite flexible regarding the definition of the elements of a web application, thus we want to be able to identify a target with as minimal amount of information as possible, and (2) we want the logger to treat event objects in the most generic way that it can, and that can be achieved by enabling a more loose event object structure.

System Overview

In order to provide with a high level overview of the logger, we want to list the main components which resemble it, as well as the phases in which its flow takes place.

The logging system integrates and functions in three main phases:

- **configuration**, where the nature of the application (multi/single modal), the types of events to be logged and the ways the replay is performed are defined,
- **event logging**, where the actual logging is registered and communicated from the application to the logger, and

- **event replay**, where an event stream is retrieved from the logging system to the application and the its behavior is replayed by the means of triggering the interaction events in that stream.

The aforementioned phases define a number of data flows between the application and the various components which resemble the logger. Specifically these components are:

- A set of **capture & replay mechanisms**, parts of which (typically concerning the types of events and the modality configuration of the application) are produced during the configuration phase. These mechanisms exist in a *client-side* context, a concept which is discussed extensively further on in this document.
- A **middleware layer**, which acts as a gateway of computation & logic between the application (specifically the capture & replay mechanisms) and the back-end of the logging system. Depending on the requirements and the scenario, this layer can be *fat* or *thin*, with regards to how rich functionality it has and how much computation and communication it needs to perform in the flow.
- A **API & storage server**, which exposes a set of web services to the application and/or the middleware for all relevant actions regarding storing or retrieving events & event streams, as well as maintaining states about the stage of the logging or retrieval/replay phase and generally performing global computations. Furthermore, the server maintains a datastore with all the relevant information about the interaction events and the participant(s) of the application.

Figures 1 & 2 demonstrate the architecture and data flow in the logging phase and the replay phase respectively. We will revisit each component from a technical perspective in Chapter 3.

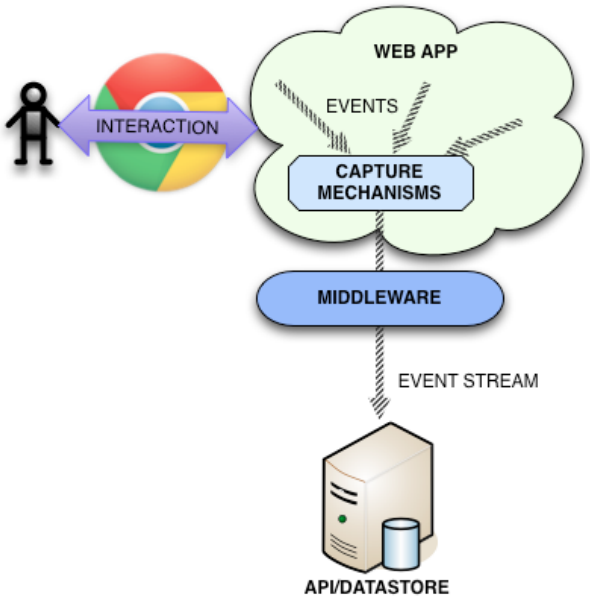

Figure 1: The flow of event logging.

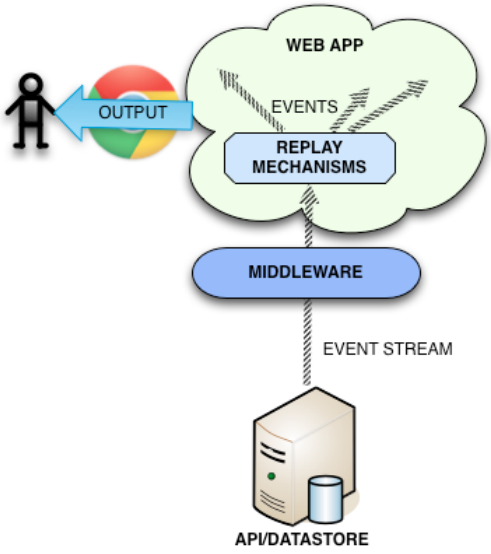

Figure 2: The flow of event replay.

Sketching Of Scenarios

In this section, we examine the cases which affect the design, implementation & feature set of the logging system as well as the application(s) under which it will operate. Each scenario is defined by applying a set of criteria of how should the application work, and in

turn it shapes the design process of the system. At this stage, we are interested in two main criteria: the application modality and how critical is the event replay operation.

With the term modality, we refer to the level of collaborative quality in the application usage: by single-modality, we imply that the application is used by one user at a time, whereas a multi-modal application is one that can be used by multiple users simultaneously (e.g a multi-player game or a voting system).

Considering the critical nature of the replay, we mainly refer to how strongly are the event occurrences coupled with each other, either in a causal or a temporal manner. This section elaborates on how each of the cases above affects the approach & risks of designing and implementing the logging & replay features of the system.

The application we built to implement & study the logger was created in versions in which we combined a number of variations between critical & uncritical replay cases of single and multi-user modes.

Single versus multi-modality

For this project, we are generally interested in applications which encourage the user to interact with the interface, because this is the main type of information we want to capture: events of (inter)action that will make the application expose its behavior. As a rough overview of our thoughts regarding application scope, our focus included applications which make the user participate in some sort of a “dialogue”, either with the application components or with other users or participants (e.g Web-based games, chat applications, voting applications, art-driven Web-based installations, etc).

In our context, a single-modal application is one where the aforementioned “dialogue” takes place between a single user and the application’s components, for example a single-player game or a drawing application. In this scenario, we have the a-priori knowledge that the timelines logged by the system originate from one source only, since there is only one user who interacts with the application. This knowledge brings specific implications when designing the logger and shapes the system in the following ways:

- When storing an event, the identity of the source is not critical because there is only one source and, therefore, only one recipient of a timeline to be replayed.
- The event logging & timeline retrieval API does not have to consider distribution of events and timelines to multiple recipients.

- The middleware layer between the application and the storage/API server needs not be aware of any peers, because there are not any.

Regarding the scenario of a multi-modal application, the fundamental difference with the single-modality case in the application level is that the input -and therefore the event streams- originate from multiple sources, often occurring simultaneously (e.g. multi-player games or collaborative art performance tools). This fundamental difference adds more depth to the design of the logger, bringing a number of intricate matters to be handled by the system.

Firstly, we know that in the multi-modal scenario the source where the events originate from matters, because the timeline can consist of events that come from different sources. The identity of the application users or participants (whether it is explicit or implicit -a concept we are discussing in detail in Chapter 3) matters and needs to be taken into account both in the phase of logging the events as well as in the phase of replaying a timeline back to the application.

The second main concern for the logger in this scenario is the temporal relationships of the event occurrences and their synchronization. Since we have multiple sources interacting simultaneously and in real-time, the logger will most probably have to process events which originate from machines which do not have fully synchronized system clocks and behave under different network schemes (which adds a variety of communication delays in the whole process), so computing the relative times of event occurrences in a timeline of such a diverse nature of origins is not a trivial task. We thus had to consider various mechanisms for maintaining a synchronized environment, also depending on how critical the replay is (more on this concept in the next section), keeping some state infrastructure on the server side and ensuring that a multi-modal timeline reliably represents the original chain of interactions in the application.

Uncritical versus critical replay

As we denoted already, the most essential feature of the event logger is its replay functionality, by which the user(s) of a given application will be allowed to fetch the stream of events they created with their usage and make them happen again in the application environment. In this feature, we wanted to provide the user(s) with certain degrees of flexibility, such as the ability to selectively replay a subset of events from a given timeline or create multiple timelines with different properties. To achieve such flexibility, we used techniques which applied both in logging-time and in the selection phase of the event stream, and we

discuss their technical details in a later chapter of this document. Similarly to the modality factor we discussed before, in this case we had to take the replay criticality into account and design our replay techniques based on it. In order to decide on our approach, the basic question we had to answer for each scenario was: *How important is a **correct** behavior of the application while/after replaying a timeline?*

At this point, it might be necessary to put the term ***correct** behavior* into context. When a user interacts with an application, they obviously expect to get some results back. The nature of these results is vastly variable; they can be computationally generated, multi-media output, a networked invocation or an interaction with another application or user, just to name a few examples. For the scope of our project, we see correctness as the combination of the application-specific set of results and the expectations of the user as to what is the minimum information they need to get back from the application. For example, in a color-changing application, if the user commands the application to change the color from grey to green and all they want to see is a color changing, without caring about whether the changed color is green, then in the case that the application changes the color to red is still considered to be a correct behavior (even though probably something is going wrong in the application internals), from the user's aspect. This approach might seem unconventional, but if we consider our initial motivations and the fact that one of our targets for this project is the possible artistic expression through real-time computer systems (in which case randomness is a usual means for sparkling artistic glory), it becomes valid because it widens the horizons for interpreting the timelines as well as implementing the logging and replay functionalities of the system. Of course, if the application architecture dictates that there should be a deterministic relationship between usage and result output for its function to be sensible & correct, then the definition of correctness takes its traditional deterministic form and the user's expectations should not differ from the application-set paradigm of correct behavior.

We use the above note about correct application behavior to speak about the criticality of the replay functionality. In our system, we look at the importance of a correct application behavior when replaying the events that happened during usage. And in the scenario where the replay is not critical, we imply that the system behavior (as it is formed by the replayed timeline) can be acceptable even if it does not match the event stream from the original usage in a one-to-one basis. Such a concept refers to applications where there are no causal dependencies between interaction events, regardless of the application modality and multi-user support. Furthermore, uncritical replay applies to systems where there are no temporal dependencies in their usage, and if there are they do not matter when replay-

ing a timeline. Thus, in the case of uncritical replay we most likely speak about applications through which the user(s) aim to produce non-deterministic results, and possibly some artistic expression through replaying their interactions.

The most profound implication brought by uncritical replay when designing the logging system is the fact that we do not have to perform extensive "event housekeeping", as we do not care much about the ordering of the events in time, the synchronization of interaction steps or sometimes even the source of the events, in the multi-modal case. Thus, the logging and replay phases are fairly simple and they require minimal specification. Additionally, the event structure as an entity becomes somewhat loose, in the sense that it needs to contain less meta-properties as there is a limited amount of information that the logger needs to know about the event objects in order to perform the logging & replay functions.

The landscape changes dramatically -as it is expected- in the case where replay has a critical nature. By critical replay we mean that the replayed event stream should reflect the exact original timeline which was produced by usage, or it should at least follow a set of strict criteria to achieve the desired (and inherently correct) behavior of the application during replay.

Typically, critical replay is required by applications which produce interaction timelines with events that are causally -and thus also temporally, from a timeline perspective- dependent on each other. It is important for such applications that the relationship of *"event B happened because event A happened"* is maintained while replaying a timeline, in order for the application to behave correctly, and this means that the logging system has to keep the correct ordering of the event occurrences into account. Additionally, since from a timeline perspective (and for the scope of this project) we want to assume that causally dependent events are also temporally dependent (rephrasing the causal concept to *"event A happened first and event B happened second"* indicates that events A & B are tied in time), the logging system needs to carefully handle the time of each event occurrence and order them accordingly in the timeline to be passed to the replay phase. With all the above in mind, and especially if we consider the scenario of a multi-user web-application with critical replay required, it is quite clear that achieving consistent logging and replay of events which occur in a distributed context and emerge from multiple sources with variable network and otherwise existent overhead is not a trivial issue. Chapter 3 discusses how we decided to tackle these issues in detail.

Application & Metaphor: A Simple Voting System

The application we built to test, prove & improve the features of the logging system had to comply with a set of criteria:

- It had to be a Web-based application.
- It should allow a single user to use it, as well as multiple users simultaneously.
- It needed to behave in a real-time fashion, with regards to the output the user(s) would receive as a product of their interaction.
- It had to represent a real-life metaphor, to assist our research and further argumentation regarding the usefulness of the logger.

For the purposes of this thesis, and given the fact that the application is merely a "playground" we used to base our case studies on and not the objective of the project, we wanted to create an environment that is minimal in terms of its visual interface & feature set, yet functional enough so we can observe the behavior of the logging system, as well as a representative example of a hypothetical real-life use case scenario.

The application we decided to build therefore consists of a fairly simple interface and functionality, and it supposedly represents a primitive, color based voting system, with which the user can express their stance on a given subject in three different levels, and each level is represented by a color that gets registered after their voting action and is shown as output on a square area, as Figure 3 illustrates. The user is able to act on the controls of the application indefinitely, and each time the color they choose is painted on the square area in real time.

The application was built in two versions, one where only one user can vote at a time and their results only appear to them, and one where there is collaboration involved and multiple users can vote simultaneously, and each vote appears in all participants' output square areas.

The way with which the user can interact in this application is by clicking one of the color labeled buttons above the square area. Thus, the event logging configuration we applied was tracking the click events on these buttons, together with the complete context of interest and event-specific properties we discussed in an earlier section of this chapter. We provide with a detailed technical description over what technologies we used as building

blocks to assemble the application's architecture, communication scheme & interaction with the logger in Chapter 3.

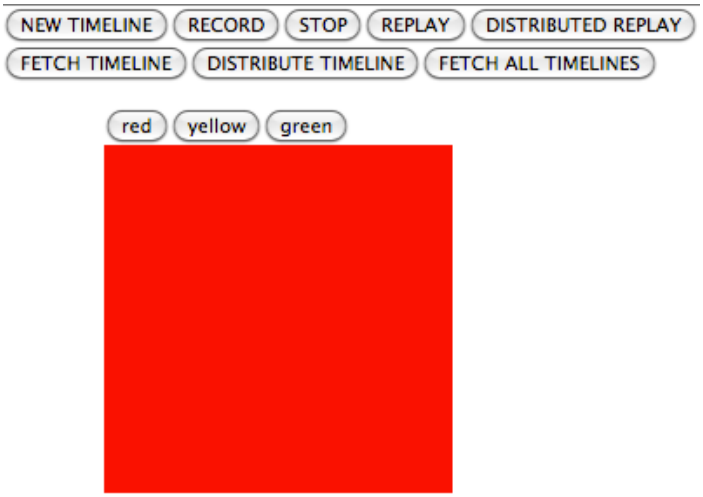

Figure 3: The interface of the application we built to test the logging system. The controls at the top of the window are application-independent and are used to manage the logger's features.