Concern-Oriented Reuse Project

Jeremie Poisson - 260627104
Anna Jolly - 260447198

October 27, 2017

1 CONCERN AND FEATURES

We have chosen to develop a new concern entitled *Data Collection*. This concern consists of a joint monitoring/logging system that can be used to collect and log data as an application is used.

1.1 MONITORING

The monitoring feature provides a way to monitor events of two kinds: performance-related and/or user-related. The monitoring feature itself contains a *MonitorableSource* class. This class is used to track the source of monitoring events. It is introduced here because it is only necessary in monitoring, since in the logging feature, sources are not monitored. The *Performance analysis* feature provides time events, which are timestamps of certain actions (e.g. the time a page is requested) and performance events, which include throughput, utilization, and count events. These events can then be aggregated into entries (e.g. the time event of a page request and the time event of when the page is done rendering might be aggregated into an entry called "Page loading duration"). On the other hand, the *User analysis* feature provides a way of monitoring user-initiated events. These events can be request events (e.g. requesting a document download), a page access event, a login event, or an exit event (closing the app or logging out). We introduce the concept of a user here, since in performance, we do not need to know the source user. Here, we need to know which events belong to which user, or the data will not be useful. An important constraint is that a user must have at least one associated event, since a user does not exist until they have logged in at least once. We can also keep track of sessions,

where a session entry consists of a login and an exit event.

1.2 LOGGING

The logging feature provides a way for application developers to trace different kinds of events in the system. Logging entries are created from logging events launched by applications developers at some point in their system. Two kinds of logging events can be launched: *TextLoggableEvent* and *ExceptionLoggableEvent*, used to log simple text or an exception, respectively. All events are then used to generate *LoggingTextEntry*, which is the textual representation of such events to be collected in the logs. The *collection* feature enables application developers to specify collectors defining how the generated logging entries must be collected (in a file, over the network, over an output stream, etc.). The *error reporting* feature enables an end-user of the application to report errors whenever a fatal error causes the application to crash during use. This feature captures the user personal information as well as their system configuration and an optional message.

1.3 NOTES

- We decided to remove the *DataCollectionManager* class. However, we were unable to delete it without causing touchCORE errors. We then tried to remove it from the XML files, but that caused subsequent weavings to crash, so we had to leave the classes in. Please ignore them.
- The logging feature is mandatory and is used to collect entries generated by the monitoring feature too. The whole premise is that it would not make sense to monitor a system without having any means of logging the monitoring results.

2 IMPACTS

We have chosen the following goals: "Increase debugging ease", "Increase system traceability", and "Increase user experience". These were chosen because the purpose of having a joint monitoring/logging system is to be able to trace events (which can be used for debugging or can be analysed to improve the software) and performance benchmarks (which can be used to determine how to make the software faster and more efficient). As such, we would like to be able to assess how well each feature is able to increase traceability, debugging ease, and how much it contributes to improving the user experience directly or indirectly (via facilitating software upgrade).

2.1 INCREASE DEBUGGING EASE

Logging is given the highest contributing value of 12, since it is the logging that allows the developer to have access to debugging info. *Error reporting* is given a 2, since a user reporting a crash can be used to find and fix bugs, although this remains less important. *Performance analysis* is given a 3 since performance events can help a developer find where the software is slow or using up a lot of CPU/memory. *Collection* is given a 4, since being able to group logging entries and send them to the right place is very helpful for debugging.

2.2 INCREASE SYSTEM TRACEABILITY

Logging is given the highest contributing value of 10 since without logging, traces would be impossible to store and comb through. *Monitoring* is given a 7, since it is the monitoring feature that allows crucial performance and user data to be traced. *Error reporting* is given a 2, since allowing users to report crashes increases traceability slightly. *Collection* is given a 1, since the ability to group traces could be said to enhance traceability.

2.3 INCREASE USER EXPERIENCE

User analysis is given an 8 since analyzing where a user spends most of their time while using an app and what features a user rarely or never uses allows the developer to tailor the app to the users, consequently increasing user experience. *Performance analysis* is given a 5 since the performance data can be used to improve app efficiency and speed, making the user happier. *Error reporting* is given a 2 for both direct and indirect reasons. Directly, reporting a crash can make a user feel better, and indirectly, reporting a crash will likely lead to a fix, which will enhance the user experience for all users.