COMP 533: Object Oriented Software Development

Concern-Oriented Reuse Project - Phase II

Jeremie Poisson - 260627104
Anna Jolly - 260447198

December 15, 2017

1 Recap of 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. The monitoring feature provides a way to monitor performance-related and/or user-related events. 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). On the other hand, the *User analysis* feature provides a way of monitoring user-initiated events. These events can be page access events, login events, or exit events (closing the app or logging out). 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 entries, which are the textual representations of such events to be collected in the logs. The *collection* feature enables application developers to specify collectors defining how the generated 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.

2 Flow of Information

2.1 Event Generation

Events can originate from three places, logging (e.g. an exception was generated), user analysis (e.g. a session was started with a login action), or performance analysis (e.g. a resource was used). In the case of logging, the *Logger* is in charge of generating the event automatically. For user analysis, the user can generate their own events in their code, or they can map their objects/functions to the pre-defined aspects of *User Analysis*. In this latter case, the user does not need to deal with events, since they will be generated "behind the scenes". One example of this is the *Login* aspect. The user of our concern needs only to map their login function to that of our *Login* aspect, and at the end of every login, the concern will automatically generate a login event and process it correctly.
Events can be regular (single) events, or they can be of type EventSequence. This means that several events are joined together as one event. An example of this is *SessionEventSequence*, which is is an event consisting of a *LoginEvent* and an *ExitEvent*. Again, the user can implement their own type of event sequence that they wish to monitor, or they can map to the pre-existing sequences available: session (as described above), or page access (which is a sequence of accesses to a single page made by a user).

2.2 Entry Generation

Once an event is generated, it is passed to the *DataCollectionManager*, which holds several kinds of *EventHandlers*. The manager gives the event to the appropriate event handler, which is charged with generating an entry from the event. This entry is then passed on to the registered *EntryCollector*, which adds the entry to the collector. The *EntryCollector* can be a single collector (e.g. a file collector), or a composite collector (e.g. a file collector as well as an output stream collector). In the case of the composite collector, the entry is added to all its collectors. The user must define their collectors in their code and register them with the *DataCollectionManager*.

2.3 Extending the DataCollection Library

If the user wishes to extend our library with a new type of event, they can do so by implementing a new type of *Event* (or *EventSequence*) and *EventHandler*.

3 Sample application

4 Notes

- Weaving of our concern fails.