ICT PROJECT 258109
Monitoring Control for Remote Software Maintenance

Project number: 258109
Project Acronym: FastFix
Project Title: Monitoring Control for Remote Software Maintenance
Call identifier: FP7-ICT-2009-5
Instrument: STREP
Thematic Priority: Internet of Services, Software, Virtualization
Start date of the project: June 1, 2010
Duration: 33 months

D3.6: Refined and Integrated Version of Context Observer, User Profiler and Error Reporting

Lead contractor: TUM
Author(s): Tobias Roehm, Joao Garcia, Dennis Pagano, Walid Maalej
Submission date: June 2012
Dissemination level: Public

Project co-funded by the European Commission under the Seventh Framework Programme
© S2, TUM, LERO, INESC ID, TXT, PRODEVELOP
Abstract: The scope of this document are the FastFix components context observer, error reporting and user profiler. We give an overview how each component works, describe how they interact to realize FastFix functionality, and summarize the changes since the first prototype(s).

This document is a supplement to the source code which can be accessed at the FastFix repository.

This document has been produced in the context of the FastFix Project. The FastFix project is part of the European Community’s Seventh Framework Program for research and development and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

1https://repository.fastfixproject.eu/svn/fastfix/software/trunk/
## Contents

1 Introduction 5

2 Context Observer 6

### 2.1 Component Overview 6

#### 2.1.1 Purpose 6

#### 2.1.2 Conceptual Overview 6

#### 2.1.3 Sensor Operation 7

#### 2.1.4 Implementing Bundles/Modules 7

#### 2.1.5 System Requirements and Sensor Operation 8

### 2.2 Changes since 1st Prototype 8

3 User Profiler 12

### 3.1 Component Overview 12

#### 3.1.1 Purpose 12

#### 3.1.2 Conceptual Overview 12

#### 3.1.3 Implementing Bundle 13

#### 3.1.4 System Requirements 13

#### 3.1.5 Starting User Profiler 14

### 3.2 Changes since 1st Prototype 14

4 Error Reporting 15

### 4.1 Component Overview 15

#### 4.1.1 Purpose 15

#### 4.1.2 Conceptual Overview 15

#### 4.1.3 Implementing Bundle 15

#### 4.1.4 System Requirements 16

#### 4.1.5 Starting Error Reporting component 16

### 4.2 Changes since 1st Prototype 16

5 Integrated Version of Context System, User Profiler and Error Reporting 17

### 5.1 Context Bus Architecture 17

### 5.2 Integration of Context Observer and User Profiler 18
1 Introduction

This document is a supplement to the source code of the integrated and refined versions of context observer, user profiler and error reporting component of FastFix. It describes briefly the concepts behind these components, the changes made since the 1st prototypes (Chapters 2, 3, 4), and the integration of the components (Chapter 5).

As this document is a source code supplement, it is rather short and abstract. For more detailed information we refer to the conceptual deliverables D3.2 [1] and D3.2e [2]. Further, we also refer to the deliverables on the first prototype of each component: D3.3 1st Prototype of Context Observer [3], D3.4 1st Prototype of User Profiler [4] and D3.5 1st Prototype of Error reporting [2].

The corresponding source code can be found in the SVN repository [4]. We will mention the bundles that contain the implementation of each component.

[1]https://repository.fastfixproject.eu/svn/fastfix/software/trunk/
2 Context Observer

2.1 Component Overview

2.1.1 Purpose

The context observer collects information concerning application execution, user interactions and the application environment. It consists of a set of sensors and the application bridge (see D2.4 [2] for more information about the FastFix architecture). The sensors are either embedded in the target application, the framework below the target application or are independent applications. Each sensor senses a certain type of data, creates events representing the sensed data and sends the events to the application bridge via an RMI or a REST connection.

The context observer collects data that is used by the user profiler component, the error reporting component, the event correlation component and others. Hence, it is a central component of FastFix.

2.1.2 Conceptual Overview

The conceptual architecture of the context observer is illustrated in Figure 2.1. The main concepts of the context observer are Sensors, ContextEvents, and the ContextBus. Sensors observe user interactions and events happening during application execution. Therefore a Sensor is a ContextGenerator and implements how particular context information is elicited. Context information is represented by ContextEvents. When a Sensor detects an event, it creates an instance of ContextEvent containing event-related information and adds it to the ContextBus. The ContextBus is a central buffer for collecting and communicating context information. Observers continuously monitor the ContextBus. Interpreters are a special type of Observers whose role is to process context information in form of events and generate aggregated context information. For a more detailed description, we refer to D3.2 [4].
2.1.3 Sensor Operation

Figure 2.1 gives an overview of a sensor in action. When a sensor observes a change or an event, it collects related information and creates a `RawEvent`. This `RawEvent` is sent to the `ApplicationBridge` either using RMI channel or REST channel. The `ApplicationBridge` creates a `ContextEvent` from the `RawEvent` and publishes it on the `ContextBus`. The distinction between `RawEvent` and `ContextEvent` is made to minimize the data transferred between sensor and client. Further, the `ApplicationBridge` adds additional information to a `ContextEvent`.

2.1.4 Implementing Bundles/ Modules

The following bundles/ modules implement a sensor of the context observer:
Each sensor communicates with the ApplicationBridge and sends observations in form of events. The ApplicationBridge is implemented in bundle eu.fastfix.client.applicationbridge.

### 2.1.5 System Requirements and Sensor Operation

As each sensor lives in another environment, the system requirements, installation process and operation process differ among the sensors. For every deployment of a sensor, an instance of FastFix client has to run on the same machine. For detailed information on how to deploy specific sensors, we refer to the latest FastFix release notes[1]. The release notes give detailed information how to download, install and operate all sensors. For more details on running the FastFix client we refer to D2.5 [5].

### 2.2 Changes since 1st Prototype

**New Sensors** Additionally to the sensors described in D3.3 [3], the following sensor have been developed:

- RCP exception sensor
  This sensor senses exceptions that occur during runtime by registering to the RCP logging mechanism.

- GMF diagram sensor
  This sensor senses addition and deletions of nodes and edges to GMF models by registering to diagram updates and eliciting the type of diagram manipulation performed.

- Tomcat user input sensor
  This sensor is a Tomcat valve, i.e. a module deployed to Tomcat, and elicits the text entered in HTML forms from an HTTP request.

---

• Tomcat HTTP request sensor
  This sensor is a Tomcat valve, i.e. a module deployed to Tomcat. It detects HTTP requests to Tomcat and elicits the requested URL, the handler code executed and jsp's rendered upon a HTTP request.

• Microsoft ClickOnce HTTP request sensor
  This sensor monitors HTTP requests from a MS ClickOnce client to the corresponding server. It extracts request specific information, sql statements contained in a HTTP request, and text input contained in a HTTP request.

• SQL statement sensor
  This sensor is a wrapper of the jdbc database driver and monitors communication between target application and database server. It elicits SQL statements sent to the database.

Sensor Lifecycle  Sensors are deployed to such different environments like Tomcat, RCP application and standalone application. Further, sensors are implemented using different technologies and languages such as Java and C#. Consequently, we found that a uniform way of sensor implementation as described in D3.3 is not possible. In order to allow a sensor to be implemented in any environment and using any language but being able to communicate with the FastFix client and be controlled by the FastFix client, we developed two mechanisms: the sensor lifecycle and two channels for sensor-client communication (see next paragraph).

We defined a sensor lifecycle (see Figure 2.3) which all sensors have to obey in order to be integrated into the platform. FastFix sensors are first installed in the client system using their own installation mechanisms (“installed” state). Upon start, sensors can register at the FastFix client (“registered” state). In the registered state, sensors regularly need to send heartbeats to the FastFix client to get new commands. If the FastFix client doesn’t respond after a specific time to the heartbeats, the sensor should return to the “installed” state. After receiving a “START” command, a registered sensor should become active (“active” state) and sense data. Whenever the sensor has gathered new data, it should send the data to the FastFix client, getting its next command as response. An active sensor shall send heartbeats as well.
Communication Channels for Sensor-Client Communication  To allow sensors to be implemented in any language and technology, the communication between sensors and the FastFix client has to follow a standardized, implementation independent way. To achieve this, we provide the ApplicationBridge interface as a set of REST services, allowing sensors to communicate with the FastFix client via REST. In order to speed up sensor development using Java, the same interface is provided via an RMI interface. For more details, see D2.4 [7].

Sensor Deployment Example  Figure 2.4 shows two different ways to deploy and connect sensors to the FastFix client. The client.logsensor runs as native Java application and observes a log file. The client.faultReplication.guiReplication and .sensing components are installed into an RCP application as plug-ins, and monitor this application. While the deployment in both cases is different, both sensor types register via REST at the ApplicationBridge, and are controlled via this interface. Both sensor types appear in the same way in the client.ui after having registered.
Figure 2.4: Deployment of FastFix sensors
3 User Profiler

3.1 Component Overview

3.1.1 Purpose

The user profiler aggregates the behavior of individual users over time and creates a profile for each individual user. For example, it contains information on how often a user performed a certain action or on how often a user visits a certain part of the application. It operates on all user-related data, which is a subset of all events generated by the context system.

3.1.2 Conceptual Overview

The user profiler component registers at the server context bus. When it receives an event, it updates the user profile for the user, event type and artifact of the current event. Figure 3.1 summarizes the working mode of the user profiler.

Figure 3.1: User Profiler Overview
Each user profile stores information on how often (frequency), how long (duration) and when (age) a specific user \( u \) performs a certain action such as clicking on a button or entering text. The information is combined with the current artifact as context, i.e. it is stored if \( u \) enters text on view Login or on View Invoice. Each individual user profile is implemented using the FDA Model [5]. More specifically, for each combination of user id, event type and artifact id, its frequency, its duration and its timestamp are stored. Frequency denotes the number of occurrences of this combination. Duration denotes the sum of the duration of all occurrences. And timestamp denotes the last occurrence of this combination and is used to derive the age. An experience value is calculated from these data using the following formula.

Let \( \mathcal{E} \) be the set of events, \( \mathcal{A} \) the set of artifacts and \( \mathcal{U} \) the set of users. We define the experience of user \( u \in \mathcal{U} \) with an event \( e \in \mathcal{E} \) regarding an artifact \( a \in \mathcal{A} \) formally as

\[
\text{experience}(u, e, a) \; := \; \frac{\text{freq}(u, e, a) \cdot \text{dur}(u, e, a)}{\text{age}(u, e, a)},
\]

with \( \text{freq} \geq 0, \text{dur} > 0, \) and \( \text{age} > 0 \) for all \( e \in \mathcal{E}, a \in \mathcal{A} \) and \( u \in \mathcal{U} \).

Frequency, duration and last timestamp are stored in the database for each combination of user id, event type and artifact id. Age and experience value are calculated when a query is made based on the values retrieved from the database. Table 3.1 shows the database table holding the user profile information.

<table>
<thead>
<tr>
<th>User Id</th>
<th>Event type</th>
<th>Artifact Id</th>
<th>Freq.</th>
<th>Dur.</th>
<th>Last timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Button Click</td>
<td>LoginView.LoginButton</td>
<td>6</td>
<td>6</td>
<td>03.05.12, 13:00 h</td>
</tr>
<tr>
<td>2</td>
<td>Button Click</td>
<td>LoginView.AboutButton</td>
<td>2</td>
<td>5</td>
<td>03.02.12, 13:00 h</td>
</tr>
<tr>
<td>1</td>
<td>TextInput</td>
<td>LoginView.UsernameField</td>
<td>8</td>
<td>12</td>
<td>08.02.12, 09:13 h</td>
</tr>
<tr>
<td>2</td>
<td>Button Click</td>
<td>LoginView.LoginButton</td>
<td>237</td>
<td>311</td>
<td>07.05.12, 18:12 h</td>
</tr>
</tbody>
</table>

Table 3.1: User Profiles in Database

For more details about the user profiler we refer to D3.2e [9] and D3.4 [10].

### 3.1.3 Implementing Bundle

The user profiler is implemented in bundle eu.fastfix.server.user.profiler.

### 3.1.4 System Requirements

The following infrastructure is required to run the user profiler:
• MySQL installed (e.g. MAMP or LAMP)
• Java 1.5 installed

Further, the FastFix server has to be configured to be able to communicate with MySQL by setting the correct database credentials (port, username, password).

3.1.5 Starting User Profiler

The user profiler is started by running the FastFix server and ensuring that the bundle eu.fastfix.server.user.profiler is incorporated and running. The user profiler is included in all FastFix releases from release 1.1. on.

Additionally to the bundle eu.fastfix.server.user.profiler the following bundles and their required bundles should be included and started:
eu.fastfix.server.communication, eu.fastfix.server.context.core.

Please note that when only the FastFix server is started as described here, nothing will happen as the user profiler is triggered by context events that arrive from the FastFix client. Hence, the FastFix client and at least one sensor have to be started additionally.

3.2 Changes since 1st Prototype

No changes have been made since the 1st prototype of the user profiler.
4 Error Reporting

4.1 Component Overview

4.1.1 Purpose

The FastFix error reporting component is responsible for creating error reports at FastFix client installations and submitting them to the FastFix server and to the error tracking system used by the maintenance team on the server-side.

4.1.2 Conceptual Overview

The error report generation system allows FastFix components to send error reports to the maintenance team. This component is informed of all relevant software errors, gathers all information for an error report and submits the reports to the FastFix Server.

The component begins by registering at the context bus and listening for all events directed at it. Whenever one such event arrives, an error report is created. These reports have two parts: a report header with a description of the error, which is added to a ticket tracking system, and a set of report log files. The report header contains information describing the error and additional context information. The log files are created by runtime sensors in the application environment and are appended to the error report created. The log files are transparently handled by the error reporting system, which simply transfers them to the maintenance server and stores them in a folder associated with the specific error report. They can be any files that the component submitting the error report specifies.

4.1.3 Implementing Bundle

FastFix error reporting is implemented by several OSGi bundles. On the client-side, it is implemented in the eu.fastfix.error.report.generation. On the server side, it is implemented in eu.fastfix.server.error.reporting.abstractions, eu.fastfix.server.error.reporting.trac and eu.fastfix.server.error.reporting.jira.
4.1.4 System Requirements

The following infrastructure is required to run the error reporting component:

- Java 1.5
- SSH server (on the machine of the FastFix server)
- SCP client (on the machine of the FastFix client)
- An error tracking system (TRAC or Jira)

Further, the FastFix server has to be configured to be able to communicate with the error tracking system. Similarly, the FastFix client has to be configured to sent error reports using the SCP client. See D2.5 for more details about the configuration.

4.1.5 Starting Error Reporting component

The error reporting component is started by starting the FastFix server and client with the inclusion of the following bundles:

- eu.fastfix.server.error.reporting.abstractions
- eu.fastfix.server.error.reporting.trac or eu.fastfix.server.error.reporting.jira
- eu.fastfix.server.communication
- eu.fastfix.client.error.reporting
- eu.fastfix.client.communication

4.2 Changes since 1st Prototype

The main addition since the submission of the 1st prototype has been the modification of the interfaces between the server-side error reporting and the error tracking systems. The first prototypesupported the TRAC error reporting system. The API used has been abstracted into an interface contained in the eu.fastfix.server.error.reporting.abstractions bundle. Currently, there are two different implementations of that API that support two different error tracking systems: a connector to TRAC (in bundle eu.fastfix.server.error.reporting.trac) and a connector to Jira (in bundle eu.fastfix.server.error.reporting.jira).
5 Integrated Version of Context System, User Profiler and Error Reporting

5.1 Context Bus Architecture

Figure 5.1 gives an overview of the ContextBus architecture and shows how a ContextListener registers at the ContextBus and how it receives ContextEvents. A ContextBus exists both on FastFix client and on FastFix server side. The events from each FastFix client are pushed to the FastFix and published on the ServerContextBus. Both user profiler component and error reporting component register themselves at the ContextBus and receive ContextEvents as explained in this chapter.

![Context Bus Architecture Diagram]

Figure 5.1: Overview of Context Bus Architecture
5.2 Integration of Context Observer and User Profiler

The user profiler runs in the FastFix server. It registers at the *ServerContextBus*, i.e. the *ContextBus* on the server side, and updates the user profiles whenever it is notified about the occurrence of a new event. Figure 5.2 gives an end-to-end overview how context events travel from sensors via *ClientContextBus* and *ServerContextBus* to the *UserProfiler*.

![Figure 5.2: Integration of Context Observer and User Profiler](image)

5.3 Integration of Context Observer and Error Reporting

There are two integration points between the context observer and error reporting components. The error reporting component registers at the *ClientContextBus*, i.e. the *ContextBus* on the client side. It listens for events that trigger error reports. These are all events directed at the error reporting component. As mentioned above, events that trigger error reports include information relevant to the description of the reported error as well as the identification of any log files that will be transferred to the FastFix Server. Figure 5.3 gives an overview.
The other important integration issue is the transfer of sensed events to the FastFix Server. Apart from any execution log files, the error reporting component does not explicitly transfer sensed context information from FastFix client to server. Because the context observer is constantly streaming sensed events to the FastFix server database – independent of the occurrence of errors –, the error reporting component assumes that context events will be available at the server side for replay. These are read from the FastFix server datastore whenever an error is replayed. Figure 5.4 gives an overview.
Bibliography


