FastFix: Monitoring Control for Remote Software Maintenance

Dennis Pagano
TU München
Munich, Germany
pagano@cs.tum.edu

Miguel A. Juan
S2 Grupo
Valencia, Spain
mjuan@s2grupo.es

Alessandra Bagnato
TXT e-solutions
Milano, Italy
alessandra.bagnato@txt.it

Tobias Roehm, Bernd Bruegge, Walid Maalej
TU München
Munich, Germany
{roehm, bruegge, maalejw}@cs.tum.edu

***Abstract*—Software maintenance and support services are key factors to the customer perception of software product quality. The overall goal of FastFix is to provide developers with a real-time maintenance environment that increases efficiency and reduces costs, improving accuracy in identification of failure causes and facilitating their resolution. To achieve this goal, FastFix observes application execution and user interaction at runtime. We give an overview of the functionality of FastFix and present one of its main application scenarios.**

***Keywords*-software maintenance; context-aware software engineering; event correlation; fault replication; self-healing**

I. Introduction

Software maintenance and support services constitute a growing percentage of the software market, in particular since business models are increasingly focusing on product quality, changing from license-centred to service-centred. Annual maintenance of software products takes 15–25% of the software license list prices [1]. Open source software licenses are often even free of charge, while service represents the major source of income.

Maintaining software is tedious, time-consuming, and intensively involves highly qualified personnel. Studies have shown that software engineers spend a significant amount of their time on maintenance tasks [5]. Assuming a typical software life cycle, more than 75% of the costs are associated to software maintenance, and this trend is expected to increase further [3].

The overall purpose of the FastFix[1] project is to provide software engineers with a "real-time" maintenance environment that increases efficiency and reduces total cost by (a) improving accuracy in identification of failure causes and (b) facilitating their resolution. FastFix includes a platform and a set of novel tools to remotely monitor customer environments, collecting information on application execution and user interaction. FastFix correlates the monitored information and *automatically* identifies symptoms of execution errors, performance degradation, or changes in users' behaviour.

We describe the FastFix approach (Section II), and demonstrate it in one specific scenario (Section III).

[1]www.fastfixproject.eu

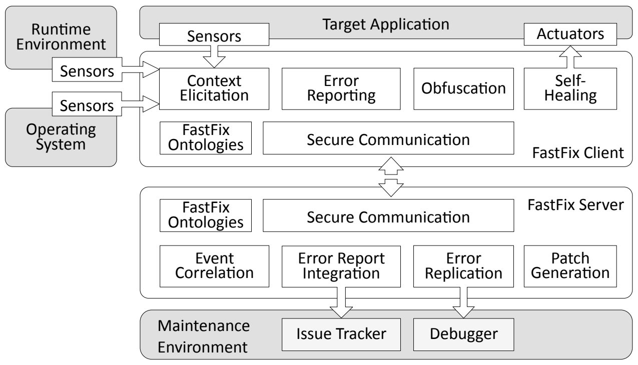


Figure 1. FastFix Conceptual Model

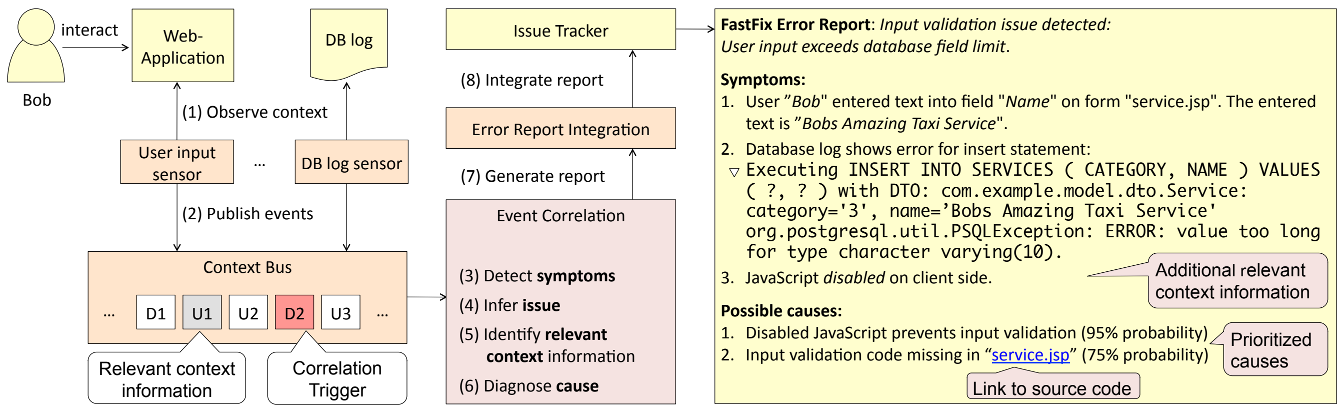
II. The FastFix Approach

As shown in Figure 1, FastFix comprises two main components: the FastFix *client* running in the execution environment of the target application, and the FastFix *server* running in the maintenance environment. The server communicates with all clients over a network. Both components include specialized subsystems that interoperate as follows.

1) Error Report Generation: Error reports typically lack relevant context information needed by developers to understand the conditions under which the error happened [7]. FastFix automatically *generates error reports* that contain *relevant* information to reproduce and fix errors. To this end, FastFix sensors continuously *gather context information*, including data on *application execution* and *user interaction*. The sensors are integrated into target application, runtime environment, and operating system.

2) Error Reproduction: By recording execution traces, FastFix enables deterministic *reproduction of errors*. The traces are *obfuscated* before sending them to the maintenance environment. FastFix links a video-like *replay* with the debugger and *augments* it by relevant *context* information, such as current CPU load, key interactions of the user, or her level of experience.

3) Issue and Cause Detection: Developers typically manually correlate symptoms in error reports with error causes [6]. FastFix *automatically* identifies incorrect execution patterns (symptoms) and infers probable issues and their causes using *event correlation* and *pattern matching*. Maintenance



FastFix Error Report: *Input validation issue detected: User input exceeds database field limit.*

Symptoms:

1. User "*Bob*" entered text into field "*Name*" on form "service.jsp". The entered text is "*Bobs Amazing Taxi Service*".
2. Database log shows error for insert statement:

```
▽ Executing INSERT INTO SERVICES ( CATEGORY, NAME ) VALUES
  ( ?, ? ) with DTO: com.example.model.dto.Service:
  category='3', name='Bobs Amazing Taxi Service'
  org.postgresql.util.PSQLException: ERROR: value too long
  for type character varying(10).
```

3. JavaScript *disabled* on client side.

Possible causes:

1. Disabled JavaScript prevents input validation (95% probability)
2. Input validation code missing in "service.jsp" (75% probability)

Figure 2. Example Scenario: Cause Detection and Error Report Generation in FastFix

knowledge is stored in ontologies. Correlation results are reported to maintenance engineers to facilitate error diagnosis.

4) Patch Generation and Self-Healing: Manual development of small patches is time-consuming and complex for highly available systems [2]. FastFix is able to *automatically generate patches* that remove specific issues by analyzing the application code. FastFix prevents identified errors from happening by employing supervision mechanisms (*actuators*), excluding faulty execution paths from the application [4].

III. Sample Application Scenario

Figure 2 shows one implemented scenario of FastFix. It illustrates how FastFix identifies an input validation issue in a web application and generates an error report including relevant context information.

In this scenario, user *Bob* enters the name of his company "*Bobs Amazing Taxi Service*" in a service brokering web application, and clicks the "*Next*" button. The application shows an error "*The operation could not be completed*". FastFix sensors continuously monitor user input, browser configuration, and database log files (1). From the observed values the sensors create events and publish them on a common context bus (2). The FastFix event correlation system continuously analyzes the event patterns on the bus. Event "*D2*" represents an *SQL insert error* observed in the database log. The event correlation system detects that this matches a critical symptom (3) of *input validation issues* (4). It identifies the previous *user input* event, and the previous event representing the *browser configuration* as relevant context information for input validation issues of web applications (5). The event correlation system determines two possible causes for the specific issue and calculates their probabilities in the given context (6). It requests the generation of the illustrated error report (7), which is automatically inserted into the maintenance team's issue tracker (8). The error report contains a description of the *issue*, a prioritized list of its probable *causes*, and all relevant information needed by developers to find *what* (database field limit for field "*Name*" exceeded, JavaScript *disabled*) and *where* (link to "*service.jsp*") to fix.

IV. Conclusions

We presented FastFix, a platform that remotely monitors applications and their contexts to identify symptoms of incorrect execution or performance degradation. Our approach includes event correlation to diagnose errors, automatic generation of relevant error reports, deterministic and context augmented error replication, automatic patch generation, and self-healing in specific cases.

Acknowledgements

FastFix is funded by the 7th Framework Programme of the European Commission, grant agreement no. FP7-258109.

References

[1] J. B. Disbrow and B. Igou. Application Software: Maintenance and Support Guidelines. *Gartner Research*, 2008.

[2] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the Vision of Autonomic Computing. *IEEE Computer*, 43(1), 2010.

[3] A. Eastwood. Firm Fires Shots at Legacy Systems. *Computing Canada*, 19(2):17, 1993.

[4] B. Gaudin, E. Vassev, and P. Nixon. A control theory based approach for self-healing of un-handled runtime exceptions. In *8th ACM International Conference on Autonomic Computing*. ACM, 2011.

[5] W. Maalej and H. J. Happel. Can development work describe itself? In *7th International Working Conference on Mining Software Repositories*. IEEE, 2010.

[6] A. Zeller. Isolating Cause-Effect Chains from Computer Programs. In *10th Symposium on Foundations of Software Engineering*. ACM, 2002.

[7] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, 36(5), 2010.