D3.2: Conceptual model for context observation and user profiling

Lead contractor: TUM

Author(s): Emitzâ Guzmán, Dennis Pagano, Tobias Roehm, Joao Coelho, Sergio Zamarripa, Raian Ali, Walid Maalej.

Submission date: March 2011

Dissemination level: Public
Abstract: This document describes a high level design for the context observation and user profiling components of FastFix. In this document we discuss the main requirements for context observation and user profiling, and present a solution approach that allows for application independent context observation. The approach is based on the notion of context events, and interpreters which process these events in order to generate knowledge useful for user profiling, event correlation, and fault replication. The document also proposes a model for determining the relevance of the sensed information when creating user profiles.

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 is respect of this document, which is merely representing the authors view.
Contents

1 Introduction ........................................ 6

2 Design Goals .................................... 7
   2.1 Purpose ............................................ 7
   2.2 Scope ............................................. 7
   2.3 Functional Requirements ....................... 8
   2.4 Non-functional Requirements ................... 9

3 Context and User Conceptual Model ............... 11
   3.1 Context as Events ................................. 11
   3.2 Types of Context Events ......................... 12
      3.2.1 User Action ................................ 12
      3.2.2 Application Reaction ....................... 13
      3.2.3 Execution Environment ....................... 13
   3.3 Use of Context Events in FastFix Components .... 14
   3.4 User Model ....................................... 16

4 Context System .................................. 18
   4.1 High Level Design .................................. 18
   4.2 Context Hub ....................................... 19
   4.3 Sensing ........................................... 21
   4.4 Processing ........................................ 23
      4.4.1 Processing Tasks ............................... 24
      4.4.2 Types of Processing ......................... 27
      4.4.3 Implementation Example ....................... 27
   4.5 Target Application Independence ................. 28

5 User Profiler .................................. 31
   5.1 User - Application Interaction ..................... 31
   5.2 Sessions and User Intentions ..................... 32
   5.3 User Profiler and Context System ................. 33
   5.4 FDA Model: Filtering Noisy Data .................. 34
5.5 Determining user’s capabilities and preferences . . . . . . . . . . . . . . . . . . . 36

6 Summary 37

Bibliography 39
List of Figures

3.1 Types of Context Events. ................................. 12
3.2 User Model. ............................................. 16

4.1 FastFix model for the context system. .................... 18
4.2 Event interfaces in the Context Hub component. ........... 19
4.3 Interactions between objects while listening to context. ....... 21
4.4 Main objects of sensing component. ........................ 22
4.5 Overview of processing tasks performed by the Interpreter subclasses. 25
4.6 Event processing in a configuration error scenario. .......... 28
4.7 Application independence using SensingStrategies. ........... 29

5.1 Relationship between context, users, application and interactions .... 31
5.2 User-Application Interaction Model. .......................... 32
5.3 Relation between Intentions and Sessions ...................... 33
5.4 Intention and Session model ................................. 33
5.5 Relationship between User Profiler and Interpreters. ........... 34
5.6 Relationship between user’s capabilities, experience and the age, duration and frequency of her actions. ................. 36
1 Introduction

Context elicitation and context processing play a major role in FastFix. Context elicitation is responsible for collecting information related to the user and the target application, whereas context processing aggregates the collected information and produces new knowledge. The collected information is useful for user profiling, fault replication, and event correlation. It is used in order to detect the necessary steps to produce a fault, as well as to detect problematic situations. The processing of the collected information is useful in event correlation where higher granular information might be desired and in user profiling where the processed information is used in order to infer users’ capabilities, knowledge, and preferences. The information generated by the user profiler is relevant for the event correlator, as it provides information that can help identify problematic situations.

In this document we describe a context system, which models context as events and includes context elicitation and context processing components. The context system is designed in order to allow for the elicitation of information generated from different types of applications, such as Desktop and Web-based applications, as well as applications running on mobile devices. The processing component allows for the realization of different processing tasks such as classification, aggregation, inference, and sequence pattern detection.

The structure of this document is as follows. Section 2 defines the scope of the context system, as well as the functional and non functional requirements. Section 3 describes how context will be represented in FastFix. It also proposes a user model and describes the information that will be collected by the context system in order to perform event correlation, fault replication and user profiling in FastFix. Section 4 describes the context system and its three main components: the sensing component, context hub, and processing component. Section 5 describes how the information obtained from the processing components can be used to realize user profiles and proposes a model to identify relevant information for the user profiler. Finally, section 6 summarizes the content presented in this document.
2 Design Goals

In this section we present the functional and non-functional requirements of FastFix concerning context observation and user profiling. We use the term context system in order to describe the system responsible for context elicitation and context processing in FastFix. In this section we first sum up the purpose of the context system described in this document. Then, we specify the scope of the context system. After that, we specify the services of the system as functional requirements. Finally, we derive the non-functional requirements, which serve as a detailed specification of the design goals.

2.1 Purpose

In D3.1 [4] we defined context as the aggregation of the application execution and the user interaction with the application. The context system is responsible for managing the context information in FastFix. Managing involves observing, capturing, analysing, predicting, and delivering context to interested components. The context system enables capturing the context of objects such as a users’ behavior (i.e. context elicitation), its analysis (i.e. context processing), as well as triggering the knowledge manipulation mechanism (i.e. knowledge delivery).

2.2 Scope

The context system is a central component of FastFix, which interacts with all other components. We consider all objects having at least one of the following characteristics as part of the context system:

- Objects that detect and instrument the context.
- Objects that represent an aggregate of the context in memory.
- Objects that represent the context as a whole in memory.
- Objects that represent the relationship between context atoms and the context that information annotates this relationship.
D3.2: Conceptual Model Context Observation and User Profiling

- Objects that interpret and process the context.
- Objects the deliver the context to the other FastFix components.

The following objects do not constitute part of the context system:

- Parts of the graphical user interface of FastFix.
- Parts of the targets that we will monitor to define the context (e.g., the applications to be fast-fixed).
- The persistent storage of the context-annotated knowledge.
- Objects in charge of detecting the correlation of context events.
- Objects in charge of generating patches for occurred problems in the target application.

2.3 Functional Requirements

This section describes the required functionality of our context system, we use the description of this functionality to develop the context system described in section 4. As defined in deliverable D3.1 [4], context includes all information related to user’s actions on the target application, the target application execution and the execution environment.

**Capture Context** Capturing context implicitly includes selecting the relevant context sources, as well as associating context with the additional knowledge when necessary.

**Observe Change** The context system should observe which changes occur to which properties of which entities.

**Store Context** The elicited context should be persistently stored in order to enable reproducing and understanding errors.

**Access Context** The context system should make the collected information accessible to other FastFix components.

**Log Application Behaviour and Application Environment** Information related to the target applications such as method calls, network, java virtual machine, and operating system events, as well as configuration information should be captured and stored. This information will later be used for error reproduction and error correlation.
Log Interaction  All information about a user’s behaviour should be captured in a log file. Examples of such interactions are for example: “the user posts a query” or “the user performs a search”.

Interpret Captured Information  The interpretation process includes generalising the context, identifying relevant context, identifying/classifying context, and associating context to other knowledge. Some activities that are related to the interpretation of the captured information are the following:

- **Synchronise Context**: The context system should determine the start and the end of a user’s session. A user session is a set of interactions between one or more applications and a user. This information is needed in order to determine which context information belongs to an error.

- **Discover Preference**: The usage preferences of the user should be discovered while analyzing previous user interactions. Examples of users’ preference can be the use of determinate configurations or the use of certain application features.

- **Discover Usage Patterns**: The context system should discover usage patterns in order to identify some important characteristics of the user. Therefore, log information should be analyzed from different perspectives and for different periods of time (i.e. set of events). For this analysis, machine learning and data mining mechanisms will be applied. However, the interpretation strategy should be expendable to further mechanisms. Changes in the usage patterns should also be detected.

2.4 Non-functional Requirements

This section describes the required non-functional characteristics of the context system. These characteristics were taken in consideration when designing the context system that will be described in the section 4.

Performance  The context system will perform its activities mainly in the background, in most of the cases without a concrete interaction with the user. The user should not face any performance problems while using the target applications. D7.1 [1] further details the performance requirements that will be fulfilled by FastFix.

Modularity  One main requirement of the system is to roll out as much functionality as possible from the specific target application. Completely integrating development environment modules or just communicating with them can present ambiguity during the design.
process. Nevertheless, internal context interpretation and structuring functionalities have to be realized independently from the underlying platforms.

**Extensibility**  To deal with highly unstable environments, resulting from changes in technical surroundings, the context system has to be extensible. The system should be able not only to support current solutions but also future technologies and standards. Such condition shall not require modification of the existing context system. Additional modules to support new technologies should be easily added.

**Privacy**  Information identifying a specific user should be obfuscated and the user should authorise the monitoring of its interaction on the different applications. D7.1 [1] further details the privacy requirements that will be fulfilled by FastFix.

**Security**  Appropriate measures such as authentication, authorisation and data encryption should be ensured in FastFix. Further details of FastFix security requirements can be found in D7.1 [1]
3 Context and User Conceptual Model

This section describes how context information will be represented in the context system, described in section 4, and gives an overview of the information the context system needs to collect in order to realize event correlation, fault replication, and user profiling. Furthermore the section presents a user model that is utilized by the user profiler described in section 5.

3.1 Context as Events

Representing context as events is a central property of our context system. Representing context as an event means that context is relative to time and represents “a change”. Sensors detect changes to targets’ properties instead of absolute values. Each context event $e$ includes the following information:

1. A time stamp $t_e$ specifying the absolute time when the context event occurred. This allows us to globally order a sequence of context events in time.

2. A duration $d_e$ specifying how long the context event is valid, that is holds a determinate value collected by the sensor.

3. A semantic context information $c_e$, describing the context event and its detected values. This information is represented as a set of RDF statements that instantiate a FastFix ontology (FastFix ontologies will be further explained in D2.3). An example is depicted in the code below. It shows that the context event $pre:e1$ has the type $art:MethodCall$, which is a subtype of $art:ApplicationReaction$ (implicit information from the FastFix ontology). The concerned application $fastfix:fastfixed_app?name=appTxt$ has the type $art:Application$. Semantic context information enables semantic processing as well as the execution of advanced context queries, such as: which method calls lasted the longest? Semantic processing will be further explained in the section 4.
3.2 Types of Context Events

In this section we describe the different types of context events that will be observed in FastFix. Context events in FastFix mainly concern user actions on applications, such as reading and editing artifacts, applications’ reactions to user input, such as method calls and the writing of logs, as well as execution environment events, such as network, java virtual machine (FastFix will focus on maintaining Java applications), and operating system events; in addition to information related to the configuration of the application. An example of a network event is the change in the network traffic, whereas examples of operating system events are memory management and CPU performance. Figure 3.1 shows the different types of context events that will be collected in FastFix.

![Figure 3.1: Types of Context Events.](image)

In sections 3.2.1, 3.2.2, and 3.2.3, we describe the types of information that will be collected. Section 3.3 describes the information that the event correlator, fault replicator and user profiler FastFix components need to perform their tasks.

3.2.1 User Action

The analysis of user actions on the target applications can help understand the source and reasons of failure, a task for which the event correlator is responsible. For example,
the analysis of a drop SQL statement introduced by the user in an application’s GUI and the subsequent deletion of a database table belonging to the target application can help the correlator deduce that the target application is not protected against SQL injection. Information concerning user action is important for fault replication because the fault replicator needs information concerning the users input to the target application in order to create an exact replication of an occurred error. Furthermore, information concerning user action is important for user profiling, where it is used to create a user profile. Examples of user actions on the application are the execution of commands, as well as the editing and reading of artifacts made available by the target application.

3.2.2 Application Reaction

Information concerning the application reaction to user action is useful for event correlation, user profiling and fault replication. This information can help the event correlator identify problematic situations and possible causes for errors. Further, application reaction information is necessary for fault replication, as this component needs to know about the applications behavior in order to replay problematic situations. In user profiling information concerning the application reaction is used to determine users capabilities and preferences. The target application’s reaction to users’ action can be classified in the following:

- **Information related with application processes**: This kind of information is strongly associated with the operating system information. Some information can be useful, like the start time, the number and identifier (TID) of the threads associated to the application, process identifiers (PID), performance information (CPU usage, available physical and kernel memory, paging, virtual memory, I/O bytes of the process), services registered in the process, command line, parent, environment variables values and user permissions.

- **Specific application data**: Refers to data inside the application, like trace log events, which can be from different subtypes: exceptions, warnings, database connection information, etc. Additional configuration information regarding static (or almost static) application data must be provided, regarding the logger level (which can be checked from the JVM), connection database pool, current version of the application or currently installed patches.

3.2.3 Execution Environment

Execution environment information is useful for fault replication and event correlation. In fault replication this information allows for more precise error replays. In event cor-
relation it enables the detection of problematic situations and in some cases it allows for
the detection of the causes leading to an error. Information concerning the execution
environment of the target applications can be classified as follows:

- **Operating system events**: Some information needed for the event correlation
  and fault replication is related to changes in the state of the operating system
  resources, since these changes can be faults or effects of operating system crashes
  (or performance degradations). On the application side, they can be strongly related
to the occurrence of an application failure, or they can also showing some of the
failure effects. Examples of these events can be understood when several resources of
the system exceed an specified threshold (CPU usage, available physical and kernel
memory, paging, virtual memory, I/O bytes).

- **Configuration Information**: Configuration information describes the actual state
  and configuration of the system. Examples of configuration information are: user/users
  currently accessing to the computer, session ID, last installed OS updates, firewalls
  or anti-virus, locale, maximum file descriptors, unmask configuration, as well as
  versions of operating systems, virtual machines, and browsers.

- **Java Virtual Machine events**: The monitoring of the Java platform, and the
  applications running on it, is also a key factor to determine failures and fault replication.
  Examples of java virtual machine events are garbage collector transitions
  and states, memory heap usage, and information related to thread deadlocks, and
  CPU time consumed by deadlocks.

### 3.3 Use of Context Events in FastFix Components

In this section we describe the types of context events needed by the components in charge
of realizing event correlation, fault replication, and user profiling in FastFix.

**Event Correlation**

Event correlation aims to detect and identify failure situations and performance degra-
dations and to determine the possible causes that led to these problems the application
is facing (even root-cause analysis), as well as providing associated likely problems and
possible solutions or workarounds. Another scenario where event correlation plays an
important role is fault prevention and pattern matching.

With these objectives, the type of information that needs to be monitored consist basically
of all sensor values that can be associated with failure symptoms. These values will be
involved with user actions, application reactions, and events related to the execution environment.

**Fault Replication**

The fault replicator in FastFix will make use of the three types of context events presented in section 3.2: user action, application reaction, and execution environment events.

Information needs to be elicited from the execution environment and device, and then recorded, in order to ensure that application replaying is deterministic, i.e. if we repeatedly input the same pre-recorded data into a program, all replays will exhibit exactly the same behaviour.

The sources of information that may influence a program’s behaviour are: the environment where the application is running, the application’s inputs and the application scheduling.

The FastFix project is targeting Java applications. Therefore the application’s execution environment is delimited by the Java virtual machine being used and the libraries it accesses.

From the perspective of fault replication, all external data can be considered as input. This includes not only data that is explicitly input in the user interface, but also data from other parts of the computer that may vary when the execution is repeated.

Application scheduling is also relevant for replaying because scheduling determines the order in which variables shared among threads are accessed and therefore the outcome of the execution. FastFix is approaching this issue by recording the order in which accesses to the application’s variables is performed.

From the application itself and information mentioned above, other information may be derived which may also be relevant for the FastFix fault replication such as, the control flow graph of the program, the path taken in that graph by a particular application, alternatives to that path and the logical conditions that lead the application through those paths.

**User Profiling**

The user profiler detects and saves information concerning users’ level of experience, preferences, goals, and knowledge. In order to extract this information the user profiler needs that the following context information be collected:

- **User actions on the application:** The user profiler needs information concerning users’ actions on the target applications. This is due to the fact that users may
work in different applications in order to achieve a goal. The collection of users’ actions help the profiler deduce a users’ level of experience, preferences, goals, and knowledge.

- **Application reactions to user actions**: Application’s responses to user actions are important in order to gain a further understanding of what users are trying to do, as well as their level of experience and knowledge about how the application works.

### 3.4 User Model

We model users based on the following characteristics, shown in Figure 3.2:

![User Model Diagram](image)

**Figure 3.2: User Model.**

- **Capabilities**: Refers to the level of expertise a user has for executing a certain task.
- **Preferences**: Refers to the users predilections when using an application. Detecting users’ preferences can be helpful for adapting applications and for determining a user’s level of expertise.
- **Intentions**: Refers to what the user wants to achieve and how he wants to do it. This information needs to be inferred from the user’s interaction with the application and is useful for augmenting the information contained in the error report.
- **Knowledge**: Refers to what a user knows or does not know about the application. This information is can be acquired explicitly or through inference rules and is useful for determining a users level of expertise.
In the first design and development iterations of FastFix we will focus on determining user’s capabilities and preferences. We presume that user’s capabilities and preferences provide the most useful information for the event correlator concerning the user. Besides, we consider that user’s capabilities and preferences add the most relevant information, in the maintenance point of view, to the error reports that will be generated by FastFix. Section 5.5 describes how user’s capabilities and preferences can be inferred from user actions and the target application’s responses.
4 Context System

The FastFix context system includes functionality for the elicitation and processing of context. Maalej [6] defines elicitation as the collection of context information mainly by instrumenting the work environment, and processing as the aggregation of information through semantic reasoning, data mining, filtering, etc. In this section we discuss the main concepts and components of our context system. The modeling of our system is based on the work presented by Maalej [6]. Section 4.1 presents the conceptual model the context system, while sections 4.2, 4.3, and 4.4 present the main components of our context system.

![Diagram of FastFix model for the context system.](image)

4.1 High Level Design

The main concepts of the context system are Targets, ContextEvents, and the ContextBus. Figure 4.1 shows how these concepts are interrelated. A Target is an object which can be monitored in order to elicit context information. Example of targets are applications, a specific interaction between the application and the user, or an artifact property. Targets consist of Properties. Property values represent the Target state. Sensors observe the changes of target Properties and are attached to the Targets. Therefore a Sensor is a ContextGenerator. Sensors implement how particular context is elicited. Context is represented by Events. When a Sensor detects changes in the target’s Properties it creates ContextEvents, and adds them to the ContextBus. The ContextBus is a central
buffer for collecting and communicating the context information. Observers continuously monitor the ContextBus. Interpreters are a type of Observers, whose role is to process context and generate aggregated context information. Interpreters form part of the processing subsystem and their role will be further explained in section 4.4.1.

The context system includes three different components: the Context Hub where the context events are collected and diffused, and the Sensing component which collects the context information, and the Processing component where new information is generated. In the following sections we describe the models of these three components.

### 4.2 Context Hub

Figure 4.2 shows the major interfaces of the Context Hub component, which implements an observer pattern [2].

![Event interfaces in the Context Hub component.](image)

**ContextBus**

The Context Bus is a central singleton which buffers the collected context information. Its main methods are the following:

- **Authentication register(ContextListener, ContextPattern)** Context listeners can register for context patterns of their interest through this interface. Listeners
D3.2: Conceptual Model Context Observation and User Profiling

get notified when context information that matches the pattern is added to the context bus. To enable a listener registering for different patterns, each registration is identified with a unique identifier, which is an integer returned at the registration time. Listeners must authenticate themselves in order to avoid misuse of the context information (such as the intrusion of the user’s privacy).

- **void unregister(Authentication)** Eliminates the registration based on its identifier. To unregister a listener must also authenticate itself with an Authentication object, that is created at registration time.

## Context Generator

**ContextGenerators** create events that are fed to the **ContextBus**. One type of event generated by the **ContextGenerator** is the **ContextEvent** described below.

## ContextEvent

This class represents a context event that is generated by a **ContextGenerator**. A **ContextEvent** denotes semantically rich information which instantiates the ontology, as explained in section 3.1. A **ContextEvent** can be of different types, as explained in section 3.2. Its most important methods are the following:

- **Statement[] getData()** returns the event’s data as an array of RDF statements. This method delivers all context information, which was captured by a context generator in RDF format.

- **OntClass getType()** returns the ontological type of the context event.

## ContextListener

The **ContextListener** gets notified by the **ContextBus** when events that match the pattern the **ContextListener** gave when registering to the **ContextBus** occur. The patterns used by the **ContextListener** implement the **ContextPattern** interface described below. **ContextListener** use the **ContextEvents** to further process the context information.

## ContextPattern

This class describes the pattern which needs to be fulfilled in order to notify a **ContextListener**. Patterns are defined in form of RDF statements with variables, for example ?event
**D3.2: Conceptual Model Context Observation and User Profiling**

\[
\text{rdf:type int:editing} \text{ where } \text{?event} \text{ is the variable resource, which is matched if a context event of type int:editing is observed. Patterns are matched semantically. For example, if the pattern is “user is editing”, the context “user executes a copy-paste” is considered to match the pattern.}
\]

Figure 4.3 describes the sequence of interactions required in order to use the notification service offered by the Context Hub. The Figure shows that a **ContextGenerator** creates a **ContextEvent**, which is then added to the **ContextBus**. In the case that the **ContextEvent** matches a pattern from one of the **ContextListeners** registered in the **ContextBus**, the **ContextBus** sends a notification to the corresponding **ContextListener**.

![Diagram](image)

Figure 4.3: Interactions between objects while listening to context.

### 4.3 Sensing

Figure 4.4 depicts the major classes of the sensing component, which we discuss in the following.

**Target**

A **Target** is something that we monitor in order to deliver context information (e.g. an application, a user interaction, or an environment property). A **Target** possesses specific properties. The **Property** values represent the state of the **Target**. A **Target** can be attached to **Sensors**, which observe the value of its **Properties**. The main **Target** interfaces are the following:

- **void generateTemplate()** initializes the **Target** by searching the ontologies for the semantics of its properties, i.e. how these properties are related to the target. First, it iterates over the properties as described in the ontology. Second, it checks
if the relationship-path between the property and the target is included in previous properties. Third, it creates the statement patterns and fills the variables. Later in the process, the sensor uses a template to generate the semantic context information.

- List<StatementVar> fillTemplate(Map<String, String>) fills the target template with the sensed values according to the ontology definition.

![Diagram of the sensing component](image)

**Figure 4.4: Main objects of sensing component.**

**Sensor**

A **Sensor** is a type of **ContextGenerator** that is attached to a **Target**. For example, user interactions represent a particular **Target**. The **Sensor** generates **ContextEvents** every time the sensed **Target** changes its **Properties**. The main methods are:

- **void reConfigure(SensingStrategy)** initializes the **Sensor** with the **SensingStrategy**, the object that will do the sensing job.

- **boolean isActive()** returns the current state of the sensor, i.e. if the sensor is currently detecting values or not.

- **boolean isDataSensed()** returns true if the sensor has already sensed some values. Sensors can continuously sense value changes, based on their sensitivity (i.e. if there
is a change that is higher than the sensitivity, a context event is created). Sensors
can also make a sensing break, e.g. for performance reasons if another sensor is
detecting semantically similar context information.

- **void dataSensed() throws SensingException** is called if the sensing strategy
detects changes to the properties of the observed target.

**SensingStrategy**

The objects of this class implement how the context information is detected by a sensor.
A **SensingStrategy** is a programmatic implementation of a **Sensor**. It implements how
value changes of the sensor’s target will be detected.

- **void config(Map<String, Object>)** configures the strategy. The configuration
  object includes the sensing logic. This can be for example the condition on which
  the values are detected, or a filter object. Each subclass has to know the keys to
  the map.
- **void start()** initializes the strategy, e.g. registers this object as a listener, creates
  communication channels, or threads.
- **void stop()** stops this strategy, e.g. unregistering it and closing open communi-
  cation channels.

While the sensing component specifies the interfaces needed for the sensing strategies, the
concrete strategy implementations will be included in separate sub-components that can
be added and substituted at runtime.

### 4.4 Processing

Context processing is the task of aggregating and interpreting the information that is
created by the sensing component (described in section 4.3) in form of events. Hence, the
input of the processing component is a stream of events which is bundled by the context
bus. The output of the processing component can take several forms: a stream of events,
a log file or database holding a (potentially enriched or filtered) subset of the information
contained in the input event stream, or a pattern that is detected in the input event
stream denoting a specific user behaviour or application execution situation.

In the following we examine at which levels context processing takes place, review what
processing tasks exist and differentiate between online and offline processing.

The stream of events generated by the sensors is the primary information source for several
FastFix components. This information is processed by observers and interpreters. In this
Subsection we summarize the processing activities that occur in the sensor and observer/interpreter level.

**Sensor Level**

The sensor strategy implements methods that describe how context information can be sensed. As described in Section 4.1 the sensor detects events and creates RDF statements containing information related to the context event. The aggregation of this information is considered a type of processing activity.

**Interpreter Level**

As depicted in Figure 4.1, the main components in the context system that process context events are observers. An observer is a component that registers at the context bus for a subset of all created context events and performs one of the processing tasks described in Section 4.4.1. An interpreter is a special case of an observer that generates new events on the context bus as result of its processing.

The event processing functionality in FastFix is designed similar to a Blackboard pattern [3]. Processing components are Sensors and special Interpreters (cf. section 4). Interpreters register for certain ContextEvents on the ContextBus. If an according event appears on the ContextBus, a processing component processes this event and places new information resulting from the processing back on the ContextBus. The resulting information can then be taken by other processing components that further process and refine these results. An example of the processing processing will be detailed in an example in section 4.4.3.

### 4.4.1 Processing Tasks

Processing of context information in form of events and context data consists of several tasks varying in complexity and the type of output. Depending on the concrete situation and the current information need, it is necessary to perform only a subset of the processing tasks. Further, some of the processing tasks use the output of other processing tasks as their input. Consequently, implementations of those tasks may implement a subset of all tasks or reuse implementations of other tasks.
In the following we describe the processing tasks performed by the Interpreter subclasses shown in Figure 4.5:

**Aggregation**

As sensors implement different levels of abstraction for events, those might not match the level of abstraction which is required and aggregation of events might be necessary. An example for this is to aggregate a stream of key stroke events (representing that the user pressed a specific key on the keyboard) to a typing event if we are not interested in the specific keys the user is typing. Aggregation is possible only for going from lower levels of abstraction to higher levels of abstraction and is usually not reversible.

**Detection of Patterns**

Detecting a set of events frequently occurring together or a certain event sequence is helpful to find typical user and application behavior and to identify causalities between certain events or actions. This information can be used to personalize the application for a specific user and to find flaws in system design and system implementation that are the reason for errors.

For example, if it is observed that most users of a web application view a certain sequence
of web page frequently the navigation structure can be adapted to that. Or, if a certain sequence of user interactions is followed by a system crash, this is a hint for an erroneous implementation and the error can be reproduced following this detected sequence. This processing task is the main purpose of the event correlation engine and will be detailed in D4.3.

**Inference of additional Information using Ontologies**

One component of the FastFix system are ontologies, which formalize the application the software maintenance domain. Context information is integrated into these ontologies (each event and its corresponding information is an instantiation of an ontology concept) and it is possible to infer additional knowledge that is not directly asserted or represented. For example, if the ontology component holds the information that reading is a certain type of interaction with a file and an interpreter observes a reading event for a specific file \( f \), it can conclude that the user interacted with file \( f \) (this becomes interesting when querying for all file interactions for \( f \)). Another example is, if the ontology component holds that two users with different names \( A \) and \( B \) are identical (because they registered twice), the system can take information asserted for user \( A \) when reasoning about user \( B \).

**Classification of a single Event**

A single event can be classified as being of a certain type based on the information associated with this event and additional information. For example, user interactions with files can be labeled to be critical if the files are system files.

**Enrichment with additional Knowledge**

Initially, an event is associated with a certain amount of context information by the sensor before it is published on the context bus. But for some processing activities, this initial information may not be sufficient or results will improve if more information would be available. If available, the information about an event could be enriched and information related to the event from other information sources can be added. For example, if an exception is sensed during application execution, static information about the current application release, the JVM version and current operation system could be added. Obviously, the additional information has to be available from another source and the task of knowledge enrichment must be able to associate the event and the additional information.
The implementation of the processing tasks will be done in a need-based fashion, according to the information needed by the event correlator, fault replicator and user profiler in FastFix.

### 4.4.2 Types of Processing

There are two types of processing context information which will be implemented in our context system: online and offline processing, in the following we give a brief explanation of each.

#### Online Processing

Online processing is a continuous process where information is constantly being fed from a dynamic source. In FastFix an Observer realizes online processing: it monitors the events continuously appearing on the context bus and processes them. As a consequence, the processor has to build history itself (storing the context information relevant to it in a local data store) and it will be notified about recent occurred events.

#### Offline Processing

Offline processing processes information (in the case of FastFix events and context) that is contained in a data store (e.g. log file or database). Offline processors are presented with information contained in the data store and not with current, “hot” events. As a consequence, offline processors cannot trigger immediate action in the system but are well suited for detecting patterns in historical data. In FastFix most of the offline processing will occur in the event correlator, whose conceptual model will be explained in D4.2.

### 4.4.3 Implementation Example

In this section we give a more detailed example to illustrate how different processing components can be used along with each other. We describe a concrete sample flow of events through different processing components in the FastFix context system. In the example the system processes events that denote a configuration error.

As shown in Figure 4.6, a DBSensor first creates a DBAccessEvent denoting that the target application tries to access a database. It then puts the created event on the ContextBus. The DBPatternDetector has earlier registered for database events on the ContextBus and thus gets notified by the ContextBus on the new DBAccessEvent. It processes the event, but does not identify any patterns.
Next, the **DBSensor** recognizes that the target application could not successfully login to the database. Thus it creates a **DBLoginFailedEvent** and puts it onto the **ContextBus**. Again the **DBPatternDetector** gets notified of this event and is now able to detect that database access is not working properly. It creates a **DBAccessErrorEvent** and puts it onto the **ContextBus**.

The **ConfigClassifier** listens to all events that may represent configuration errors. It gets notified of the **DBAccessErrorEvent** and creates a **ConfigErrorEvent** denoting that a configuration error occurred.

![Event processing diagram](https://via.placeholder.com/150)

**Figure 4.6: Event processing in a configuration error scenario.**

### 4.5 Target Application Independence

As mentioned in the non-functional requirement portability in section 2.4, FastFix should be used to maintain arbitrary applications. The sensing component therefore needs to be able to collect context information independently from the target application. We realize this independence through the use of a strategy pattern in the sensing component and through the design of our FastFix client architecture.

The strategy pattern in the **Sensing** component decouples the **Sensing** interface from its implementation. **SensingStrategy** implementations thus can be developed for arbitrary
applications. The pattern further allows the addition and exchange of the `SensingStrategies` at runtime.

The architecture of the FastFix client includes an additional layer between the maintained application and the context observer component as shown in Figure 4.7. This Application Bridge further decouples the conceptual `Sensing` model from the different implementations of the `SensingStrategies`.

![Diagram](image)

**Figure 4.7: Application independence using SensingStrategies.**

How a target application may be observed depends strongly on the target application itself. Some applications e.g. provide the possibility to install plugins that may serve as information providers for the context observer. In general, we distinguish the following observation approaches:

- **Plugin based.** If the target application provides means to include external plugins, `SensingStrategies` may observe application specific information exposed by an additional FastFix plugin.

- **Instrumenting execution environment.** This observation approach uses facilities given by the runtime environment to observe the target application.

- **Instrumenting applications.** In this approach the developers of the target application instrument their application in order to collect information.

- **Bytecode instrumentation.** The instrumentation of bytecode allows to inject functionality into an already compiled application. Using this technique, FastFix
D3.2: Conceptual Model Context Observation and User Profiling

functionality can be included in a target application to provide information for a SensingStrategy in the Application Bridge.

FastFix’s most important sensing targets are the application and its environment and the interactions of the user and the application. We plan to collect context information about these targets either by using plugins, instrumenting the execution environment or the application or realizing bytecode instrumentations. Operating systems and virtual machines allow for the monitoring of particular types of events such as editing a document. The Windows operating system provides hooks, while Mac OS X offers an Apple Script interface to implement such functionality. Additionally the Java virtual machine and the Eclipse Runtime Environment have libraries to register to execution events. Another alternative are program-monitoring frameworks which are integrated into the operating system and execution environments in order to trace program execution. These program-monitoring frameworks enable a debugging-like operation at runtime. Dtrace\(^1\) and systemTap\(^2\) are two examples of these frameworks. Dtrace was developed Sun Microsystems and is integrated into Solaris and Mac OS X Leopard. SystemTap works on Linux. Both frameworks offer special programming languages for efficiently tracing and monitoring black-box applications. They can be used to listen to the execution of particular functions, which reveal particular interaction. For example, the collected information can be used in order to realize input validation and avoid SQL injection or other vulnerability errors, the information can also be used in order to replicate crashes of java applications due to concurrency problems or unhandled exceptions. Other examples where the collected information can be used is for the detection of configuration errors in desktop applications, as well as for the detection of environment related errors, usability errors, and errors related to the usage of third party components in mobile applications.

\(^1\)http://www.sun.com/bigadmin/content/dtrace/
\(^2\)http://sourceware.org/systemtap/
5 User Profiler

The user profiler is in charge of modeling users. We model users based on the characteristics described in section 3: knowledge, preferences, capabilities and intentions. The user profiler will infer these characteristics through context interpreters and the generated knowledge will be stored in ontologies. In this section we describe our model for user - application interaction, how we organize these interactions into sessions and user intentions, and how the user profiler uses the context system interpreters in order to infer users’ knowledge, preferences, capabilities and intentions. In addition in this section we propose a model for determining the relevance of the data collected by the sensors, that will be used by the user profiler.

5.1 User - Application Interaction

Users, applications and their interactions are part of the context, as shown in Figure 5.1. Maalej [6] defines interactions as events that occur when two or more objects have an effect on each other.

![Diagram](image)

**Figure 5.1:** Relationship between context, users, application and interactions

The FastFix user profiler aims at modeling the interactions that the user has on the target applications in order to identify users’ knowledge, capabilities, usage preferences and intentions. Figure 5.2 shows how Users, Applications, Artifacts and Interactions
are interrelated. Users interact with Applications in order to access and manipulate Artifacts. Examples of Applications are the target application, photo editors, email clients and web page browsers. Examples of Artifacts are emails, chat logs, todo lists, pictures and graphs. Some user Interactions change the Artifacts the Application is handling (we call this type of Interaction ChangeInteraction). Each Interaction is done through a specific Application and concerns a specific Artifact. Examples of Interactions are creating a PDFs, writing an email, or uploading a picture. The next section describes how these Interactions are organized in Sessions and Intentions.

![User-Application Interaction Model]

Figure 5.2: User-Application Interaction Model.

### 5.2 Sessions and User Intentions

Users interact with applications for a determinate period of time. We call each period of time when a user is interacting with one or more applications without a prolonged interruption a session. A session has a start and end time, and at a given time it is not possible for more than one session to be active. Intentions describe the goals the user wants to achieve when interacting with an application. For example, when writing an email, the actions realized by the user are keyboard input, scrolling and clicking the send button, whereas the intention of the user is to communicate with another person. The actions leading to the fulfillment of an intention can be organized in a single session or in several sessions spanned over time, as Figure 5.3 shows.
The Figure illustrates an Intention $I_1$ realized in the non-continuous Sessions $S_1$ and $S_3$, as well as an Intention $I_2$ realized in Session $S_2$.

Figure 5.4 shows a graphical representation of the relationship between intentions, sessions, interactions and the user. User’s Interactions with the different Applications are organized through Sessions. A Session consists on one or more Interactions concerning one or more Applications. Sessions are closed when the User remains idle for a certain time span. Intentions designate what a User wants to achieve through the Interaction with the Application. Intentions can be achieved through a number of Interactions organized in one or more Sessions.

Figure 5.4: Intention and Session model

5.3 User Profiler and Context System

Users’ interactions with the application are monitored by sensors, which create the context events that are sent to the context bus. These events are processed by different interpreters, in order to obtain the user’s intentions, preferences, knowledge and capabilities. In order to recognize these user characteristics interpreters need to analyze the users actual interactions with the application, the application responses, as well as the users
previous behaviour. Because of this, interpreters use the context events available in the context bus, as well as information stored in the user ontology. Online learning processing concerns interpreters exclusively using information provided by the context bus, whereas offline processing concerns interpreters using information from store. A user profiler is a type of interpreter that listens to the context events created by the other interpreters. Figure 5.5 shows different Interpreters for detecting user intentions, capabilities, preferences and knowledge, as well as interpreter in charge of profiling users.

Figure 5.5: Relationship between User Profiler and Interpreters.

5.4 FDA Model: Filtering Noisy Data

In this section we propose a model to access the relevance of information taken into consideration by the interpreters. Our relevance model is based on the combination of three metrics: the Frequency, Duration, and Age (we call this the FDA model) and is useful for removing noisy data. Our is based to the model proposed in the work of Maalej and Sahm [7]. The model is based on an observation, which is best explained by the following scenario.

When a user is working on a task $t$, the more an artifact $a$ is important for the task, the more she interacts with that artifact (Frequency). The duration $d$ of the interaction with that artifact plays a role for assessing its importance as well. If two artifacts are edited with the same frequency, say 3 times, but one is edited for a short period of time and the other for a long period, then the latter is more important for the task (Duration). Finally, if an artifact was edited several days before and the other is just being edited, than the recently edited one is likely to be more important and the more aged is likely to be less important (Age). For a particular session, we define the following constants:

- **Session Duration $D$**: the absolute duration of a session in seconds ($D > 0$).
- **Total frequency $F$**: the number of all context events related to user action and application response observed in the session ($F > 0$).
In addition we define the following sets:

- **IN**: the set of all interaction events of the session.
- **AR**: the set of all artifacts used in the session or the set of all applications used in the session.

We now define functions to measure the Frequency, Duration and Age for each artifact. Let \( e_{i,a} \) be the context event denoting an interaction of type \( i \) (e.g. a change interaction) concerning the artifact \( a \).

- The Frequency \( Frq(i,a) \) denotes the normalized count (i.e. relative value) of the interaction \( i \) with the artifact or application \( a \) in the session, as defined in 5.1.

\[
Frq : \begin{cases} 
IN \times AR \rightarrow (0..1) \\
Frq(i,a) \rightarrow \frac{\text{count}(e_{i,a})}{P}
\end{cases}
\]  

(5.1)

- The Duration \( Dur(i,a) \) denotes the normalized time of the interaction type \( i \) with the artifact or application \( a \), as defined in 5.2.

\[
Dur : \begin{cases} 
IN \times AR \rightarrow (0..1) \\
Dur(i,a) \rightarrow \frac{\sum d(e_{i,a})}{P}
\end{cases}
\]  

(5.2)

- The Age \( Age(i,a) \) denotes the absolute time in seconds since the last interaction of type \( i \) with the artifact or application \( a \) occurred (\( max(t_{i,a}) \)), as defined in 5.3.

\[
Age : \begin{cases} 
IN \times AR \rightarrow IN \\
Age(i,a) \rightarrow \text{max}(t_{i,a})
\end{cases}
\]  

(5.3)

Based on these functions, we now define in 5.4 the relevance \( REL(i,a) \), which denotes the relevance of the artifact \( a \) concerning the interaction of type \( i \) in the session. \( REL \) is proportional to \( Frq \) and to \( Dur \), while inverse proportional to \( Age \). \( REL \) enables us to interpret, e.g. what the most read artifact is, what the most often changed artifact is. Thus the associations between the change and the context can be order based on its relevance.
During the implementation of the context system we will evaluate the correctness of the proposed model through empirical evaluation. In our future work we will define the necessary metrics to evaluate the model.

### 5.5 Determining user’s capabilities and preferences

Frequency, Duration and Age, as defined in equations 5.1, 5.2, and 5.3 can also be used in order to determine users’ capabilities and preferences. The more and longer a user utilizes an artifact or application, the more the user prefers that artifact or application. In this case the $Age(i, a)$, gives access to a historical dimension, where more recently used artifacts or applications are considered more relevant for determining users’ preferences. On the other hand through the $REL(i, a)$, defined in equation 5.4, the most used applications and artifacts can be discovered and with this information we can then determine the user’s level of expertise. The more a user utilizes an artifact or application that is classified as **difficult** the more expertise the user is considered to posses. A more precise classification of a user’s capability can be obtained by applying machine learning techniques such as sequential pattern matching [5] to the action sequences performed by the user on certain artifacts or applications. Through sequential pattern matching we can identify when a user is, for example, formatting text by using keyboard short cuts or by navigating option menus, and then classify its level of expertise. In this example, the user making a more extensive use of short cuts will be classified as more advanced as the one using the menu options.

![Diagram](image)

**Figure 5.6:** Relationship between user’s capabilities, experience and the age, duration and frequency of her actions.
6 Summary

This document describes the functional and non-functional requirements for building a context system for FastFix. The identified functional requirements are: capture context, construct context, store context, query context, observe change, interpret captured information, log application execution and its environment, log user interaction, and interpret context information. The identified non-functional requirements are: performance, modularity, extensibility, flexibility, and privacy. These requirements represent criteria that were taken into consideration in the design of the context system, and should be taken into consideration in future iterations.

The document also presents our solution approach to address these requirements. Context information is represented as events consisting of a start time, duration, and related context information. In addition to modelling context as continuous change, we present a user model that takes into consideration user’s capabilities, knowledge, preferences and intentions to rate the single error reports and their context. The document also summarizes the type of context information needed in order to realize event correlation, fault replication, and user profiling. This context information can be classified into information concerning user interaction, application reactions, and the application environment.

In this document we propose a context system for FastFix, which allows for the collection and processing of information concerning user and application interactions, as well as application execution. The document describes the domain model of the main components of the context system: the sensing, context hub, and processing components. The sensing component allows for the collection of information independently of the target application. This is done by using a strategy pattern, bytecode injection, instrumenting the application or the environment, and by using plugins. Context is represented as a type of event that denotes the change of the objects it is sensing. The context hub is in charge of providing the sensed context information to all other FastFix components. The processing component generates knowledge from the collected context information through processing tasks such as: classification, inference, pattern detection and aggregation.

The document also presents a user model for the user profiler where the interactions of the user and the application are organized in working sessions and intentions in order to identify context information relevant for error reports. Furthermore the document
proposes a model for the identification of the users’ capabilities and preferences based on the frequency, duration, and age of user interactions with the application.

The sensing strategies and processing tasks described in this document will be implemented in a need driven fashion meaning that we will implement those which are necessary to obtain a certain kind of information or fulfill a certain FastFix requirement. We will implement reusable modules. This enables to build more complex processing components reusing simpler processing components and avoids redundant implementation of functionality.
Bibliography


