<table>
<thead>
<tr>
<th><strong>Project number:</strong></th>
<th>258109</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Acronym:</strong></td>
<td>FastFix</td>
</tr>
<tr>
<td><strong>Project Title:</strong></td>
<td>Monitoring Control for Remote Software Maintenance</td>
</tr>
<tr>
<td><strong>Call identifier:</strong></td>
<td>FP7-ICT-2009-5</td>
</tr>
<tr>
<td><strong>Instrument:</strong></td>
<td>STREP</td>
</tr>
<tr>
<td><strong>Thematic Priority:</strong></td>
<td>Internet of Services, Software, Security</td>
</tr>
<tr>
<td><strong>Start date of the project:</strong></td>
<td>June 1, 2010</td>
</tr>
<tr>
<td><strong>Duration:</strong></td>
<td>30 months</td>
</tr>
</tbody>
</table>

**D.7.1: Global performance, security and privacy requirements.**

**Lead contractor:** S2 GRUPO

Roberto Amado, David Monteagudo, Fernando Seco García de Ceca, Sergio Zamarripa López, Emitzá Guzmán, João Coelho García, María del Carmen Calle

**Author(s):**

García de Ceca, Sergio Zamarripa López, Emitzá Guzmán, Joao Coelho García, María del Carmen Calle

**Submission date:** January 2011

**Dissemination level:** Public

---

Project co-founded by the European Commission under the Seventh Framework Programme

© S2, TUM, LERO, INESC ID, TXT, PRODEVELOP
**Abstract:** The state of research and practice regarding performance optimization on run-time monitoring will be examined, documented and discussed with the Consortium in order to find out the best alternatives to meet FastFix requirements. Special attention will be paid to reduce footprint on monitored applications and the possibility to relay on resources provided by multicore processor. On the other hand, security requirements will be reflected in both business and functional requirements for each iteration. Special attention will be paid to confidentiality solutions for network traffic, confidentiality solutions for long-term storage of key data or privacy concern, specially for personal data. Functional security requirements will show how the basic security services are addressed for each resource in the system attending to access control (authorization), authentication and integrity, confidentiality (including privacy), availability and accountability (including non-repudiation).
Contents

1. Global performance requirements 7
   1. Introduction 8

2. Foundations 9
   2.1. Monitoring ........................................ 9
       2.1.1. Tracing ........................................ 9
       2.1.2. Profiling ..................................... 10
       2.1.3. Collector patterns ............................ 10
   2.2. Performance requirements, goals and testing objectives identification ... 13

3. Techniques. 14
   3.1. Execution environment ................................ 14
       3.1.1. Operating System .............................. 14
       3.1.2. Virtual Machine ............................... 16
       3.1.3. Java Virtual Machine ......................... 17
       3.1.4. Desktop application monitoring ................. 20
       3.1.5. Web application, monitoring the browser environment ........ 22
   3.2. Execution environment independent monitoring .................. 26
       3.2.1. Software Monitoring Controllable Overhead .... 26
       3.2.2. Java Performance Tuning .................... 30
   3.3. Application Recording ................................ 32

4. Related projects and tools 33
   4.1. Tools ................................................ 33
       4.1.1. Windows ....................................... 33
       4.1.2. Linux ......................................... 34
   4.2. Projects .............................................. 38
       4.2.1. Health Center ................................ 38
       4.2.2. Tracing and Monitoring Framework (TMF) .... 39

5. FastFix Global Performance Requirements 41
   5.1. Introduction ........................................ 41
   5.2. Requirements ....................................... 42

II. Security and privacy requirements 44

Introduction 45
# List of Figures

1.1. FastFix’s Runtime Platform ........................................ 8
2.2. Listening collection pattern [3] ....................................... 11
2.3. Interception collection pattern [3] .................................. 12
3.1. Hypervisor ...................................................................... 17
3.2. Architecture of J2SE 5.0 Monitoring and Management Support[33] .... 19
3.3. Exposing Application Bean [50] ....................................... 20
3.4. Common Problems in JavaSE Applications ............................. 21
3.5. Observer pattern ............................................................... 22
3.8. Web Worker example (running the calculation of a prime number in a different thread) ................................................. 25
3.9. HCSM control theory [5] .................................................. 27
3.10. Structure of SMCO[23] .................................................... 27
3.12. SMCO cascade control architecture [23] .............................. 29
3.13. Cascade controller observed overhead [23] ............................ 29
3.14. MBeans tab[33] .............................................................. 31
4.1. LTITng Arquitecture ......................................................... 35
4.2. Dtrace’s schema ............................................................... 36
4.3. Basic Flow of SystemTap .................................................. 37
4.4. SystemTap’s script ............................................................ 37
4.5. Run-time impact of ReTrace capturing [62] ............................ 38
6.1. Security methodology ....................................................... 48
8.1. OSGi architecture ............................................................ 65
## List of Tables

3.1. Proxy CSA injection overhead .............................................. 24  
3.2. CSA accessing all the elements of the window (overhead) ............ 25  
5.1. Global Performance Requirements ........................................... 42  
5.2. Reported vulnerabilities ..................................................... 45  
8.1. Global security requirements ................................................ 59  
8.2. Authentication requirements ............................................... 59  
8.3. Session management requirements ........................................ 60  
8.4. Authorization requirements ................................................ 61  
8.5. Input data validation requirements ....................................... 61  
8.6. Configuration management requirements .................................. 62  
8.7. Event correlation security requirements .................................. 62  
8.8. Java security requirements .................................................. 63  
8.9. Client and sensors security requirements .................................. 63  
8.10. External communications security requirements ........................ 64  
8.11. OSGi permissions requirements ............................................ 65  
8.12. OSGi conditions requirements ............................................. 66  
12.1. Privacy requirements ......................................................... 81
Part I.

Global performance requirements
1. Introduction

Runtime monitoring is critical to achieve and maintain a well-performing application because numerous external and internal factors can have an impact on application’s performance and availability. These influences are virtually impossible to be completely eliminated or emulated in a preproduction environment. But the severity and duration of these events can be significantly reduced by creating and maintaining a comprehensive system that monitors application environment.

The runtime environment is mainly composed by the Operating System (OS) or a Virtual Machine (VM), the Java Virtual Machine (JVM) and the Browser (talking about web-based applications). All of them must be monitored because their state can affect the performance and behaviour of the application being monitored.

In this paper, we will discuss lightweight techniques or tools for monitoring applications and their environment, with a low-impact in their performance; and modifications to be done against the current monitoring techniques (recommended configurations, tips,...) with the purpose of minimizing the performance impact in the target application.

Hence, in this part we’ll discuss about Operating System, Virtual Machine, Java Virtual Machine and Application monitoring techniques and tools. The discussion is focused on low impact, because the data collection needs to occur without both interrupting the user’s workflow and influencing the monitored production environment performance. Some of the techniques included in this set are independent of the execution environment, such as Java programming tips in order to minimize the FastFix Client impact. Some low impact techniques for application recording and keeping user privacy are also discussed.

For summarizing, this part of the document analyzes techniques and tools in order to provide low impact monitoring (gathering and recording data). It constitutes the state-of-the-art in this domain and will be used in order to obtain high-level low impact monitoring requirements.
2. Foundations

In this chapter we provide some definitions divided in two sections in order to build a common vocabulary. In the first one, some terms related to Monitoring are presented. This section includes four approaches to gather data, depending on the behaviour of the collector. Some of these patterns are used by the techniques shown in Chapter 3. The second section tries to distinguish between performance requirements, performance goals and performance testing objective.

2.1. Monitoring

Monitoring allows you to collect and analyze information about application execution and its environment in order to improve them. Two approaches can be distinguished depending on what kind of application data will be collected: *tracing* which provides information relative to the behaviour of an application (including the user interaction with it) and *profiling* mainly relative to performance.

2.1.1. Tracing

It’s the way of knowing the behaviour of an application without having the sources and with a minimum of disturbance [47]. This dynamic analysis technique allows to capture events of interest on a running program continuously. Some examples of traced events are the occurrence of a statement, the invocation of a function and the trigger of a signal [53]. Guzman et al. [21], as part of Deliverable D3.1 of this project, pointed out some of the main features for tracing and other techniques to collect data.

- Efficient Instrumentation: In order to elicit information the end user’s work environment needs to be instrumented. For unobtrusive context awareness data collection needs to occur without interrupting the user’s workflow. We are interested in context-aware frameworks which are able to work independently of the user’s workspace, as this will lead to context-aware systems that are independent of their domain. This point opens the question of how sensing instruments can be integrated into underlying frameworks, GUI libraries, operating systems, middleware and execution environments.

- Richness and Quality of Information: The quality and richness of the information delivered by a sensor varies over time and context models should be able to adapt themselves to this situation. Supporting quality and richness indication when modeling is a possible solution to this problem.
2.1.2. Profiling

It allows you to learn where your program spent its time [43]. It means profiler is focused in performance information.

- Performance: It’s an indicator of how well a software system or component meets its requirements for timeliness. There are two important dimensions to software performance - timeliness, responsiveness and scalability [59].

- Responsiveness: It’s the ability of a system to meet its objectives for response time or throughput. The response time is the time required to respond to stimuli (events). The throughput of a system is the number of events processed in some interval of time [4].

- Scalability: It’s the ability of a system to continue to meet its response time or throughput objectives as the demand for the software function increases [59].

- Capacity: Although the terms scalability and capacity are frequently used interchangeably, they are quite different in performance characteristics when an application experiences increased usage. Capacity is a reflection of size and volume limitations. An application may scale poorly as a result of a capacity limitation, but it may scale poorly for any number of other reasons as well. Capacity limitations don’t always reveal themselves during scalability testing.

Profilers will be used to measure the FastFix’s Context Observer impact in the monitored applications and its environment.

2.1.3. Collector patterns.

Sensors or data collectors typically use one of four patterns, which influences the tracer type that should be used [3]. Four patterns are analyzed: polling, listening, interception and instrumentation in order to introduce the most common ways to collect data.

Polling

A performance data source (PDS) is a source of performance or availability data that is useful as a measurement to reflect a component’s relative health. For example, Java Management Extensions (JMX) services can typically provide a wealth of data about the health of a JVM.

When this collector pattern is applied, the collector is invoked on a regular frequency, and it retrieves and traces the current value of a metric or set of metrics from a PDS. For example, a collector might be invoked every minute to read a host’s CPU utilization or read the total number of committed transactions from a transaction manager through a JMX interface. The premise of a polling pattern is a periodic sampling of a target metric. So on a polling event, the metric’s value is supplied to the Application Performance Management (APM) system, but for the duration of the intermediate periods, the value is assumed to be unchanged. Accordingly, polling collectors typically use sticky tracer types: the APM system reports the value as unchanged in between all polling events. Figure 2.1 illustrates polling pattern.
D7.1: Global performance, security and privacy requirements

This pattern is implemented by statistical tracers; therefore, its premise, the periodic sampling, is a technique that can be applied in order to improve other monitoring techniques, as we can see in subsection 3.1.3

**Listening**

This general data pattern is a form of the Observer pattern. The collector registers itself as a listener of events with the target PDS and receives a callback whenever the event of interest occurs. The possible traced values issued as a result of the callback depend on the content of the callback payload itself, but at the least the collector can trace an incident for every callback. Figure 2.2 illustrates listening pattern.

Monitoring techniques like Hardware Breakpoints (subsection 3.1.1) implement this pattern.

**Interception**

In this pattern, the collector inserts itself as an interceptor between a target and its caller or callers. On each instance of activity that passes through the interceptor, it makes a
Figure 2.3: Interception collection pattern [3]

measurement and traces it. In cases in which the interception pattern is request/response, the collector can measure the number of requests, the response time, and possibly some measurement of the payload of the request or response. Figure 2.3 illustrates interception pattern.

In section 3.1.5 we can see an implementation of this pattern in order to monitoring web based applications client side.

Instrumentation

In this pattern, the collector is some code, which can be injected in source or binary code. Source code instrumentation is not recommended in FastFix, because this source code must be recompiled, and this is a very intrusive technique. Hence, in this subsection we will focus on binary instrumentation.

This pattern is used for monitoring both OS and applications. It is commonly used to collect information for many diverse analysis applications, such as dynamic slicing and alias analysis, software security checking, and computer architecture modeling.

There are two main approaches to binary instrumentation: static and dynamic. Static binary instrumentation inserts additional code and data into an executable and generates a persistent modified executable whereas the dynamic instrumentation inserts additional code and data during execution without making any permanent modifications to the executable. The static approach can be advantageous because it usually results in more efficient executables when compared to the dynamic approach. This is a result of the fact that static binary instrumentation introduces only the instrumentation code itself.

With dynamic binary instrumentation, additional overhead is introduced because the instrumentation tool must perform additional tasks such as parsing, disassembly, code generation, and making other decisions at runtime.

This is simply not an issue with static binary instrumentation tools because all decisions and actions are taken prior to runtime. The only cost taken at runtime is the
direct cost of performing additional instrumentation. However, static binary instrumentation has some disadvantages. It is not possible to instrument shared libraries unless the shared libraries are instrumented separately [30] and the executable is modified to use those instrumented libraries. Static instrumentation also provides less flexibility to tool developers since any instrumentation code persists throughout the application run while dynamic instrumentation provides the means to delete or modify instrumentation code as needed.

A mixed technique between static/dynamic source code/binary instrumentation will be presented in subsection 3.1.1 in order to minimize kernel instrumentation overhead.

2.2. Performance requirements, goals and testing objectives identification.

When we try to define performance requirements, the first we think is present some measurements about network latency, system overhead or application performance, in order to probe later that the application carries out them.

This could be a valid approach to “common” application, because probably other applications with similar functionality, technology or number of users are known, and we can extrapolate performance measurements.

However, this approach can’t be used to obtain FastFix performance requirements, because we don’t know yet all technical and functional requirements and, even if we knew them, we have not found any similar application to be compared with. In addition to that, in this kind of projects it is advisable to follow a conservative methodology while trying to quantify performance requirements. The selected methodology is based on the approach performed by Scott Barber, who is Chief Technologist for PerfTestPlus, Executive Director of the Associations for Software Testing and co-founder of the Workshop on Performance and Reliability. In Chapter 5 a summary of this approach is presented. It is based in distinguishing between performance requirements, performance goals and performance testing objectives.

- Performance requirements: Criteria that are absolutely non-negotiable due to contractual obligations, service level agreements or business needs. Only those criteria whose sub-par performance would unquestionably lead to a decision to delay a release are absolutely required.

- Performance goals: Criteria desired for release, but negotiable under certain circumstances. For instance, if a response time goal for a particular transaction is set at 3 seconds, but the actual response time is determined to be 3.3 seconds, it is likely stakeholders will choose to release the application and defer performance tuning of that transaction for a future release.

- Performance testing objectives: Items that add value to the team through the process of performance testing but are not intrinsically quantitative. For example, one objective might be to provide certain data to systems administrators to assist them in tuning systems under their purview. Another objective might be to determine peak application usage that the existing network can support.
3. Techniques.

In this chapter we present some lightweight techniques and some modifications to be done against the current monitoring techniques with the purpose of minimizing the performance impact in the target application (Operating System, Virtual Machine, Java Virtual Machine and Browser).

We also present other techniques which are independent of the execution environment, it says, some techniques for reducing or constraining the overhead produced by monitoring, regardless of the environment. Some Java programming tips are introduced in this chapter in order to reduce FastFix Client performance impact.

In addition to that, we will pay special attention to low impact application recording and obfuscation techniques, since they are also key in the performance of the system under monitoring.

3.1. Execution environment

As already said in the Introduction, application monitoring includes runtime execution environment monitoring. The runtime environment is mainly composed by an OS or a Virtual Machine, the Java Virtual Machine and the Browser (talking about web based applications). All of them must be monitored because their state and behaviour can affect the performance and ongoing of the application which is being monitored.

In order to monitor the OS we have found two approaches. The first one is based in some enhancements of the Linux kernel, in order to improve the impact of kernel instrumentation, and the second one is focused in hardware: the Hardware Breakpoints. Then, in Virtual Machine section, the use of an hypervisor to minimize the monitoring impact is discussed. Regarding JVM, two techniques about SNMP and JMX are analyzed. The last two sections are dedicated to application monitoring, focusing the first one in the browser, and the second one in Eclipse RCP applications.

3.1.1. Operating System

Current state of the art does not provide evolved techniques in order to monitor Windows with low impact. However, System Center Desktop Enterprise Monitoring (an agentless Microsoft tool that monitors and reports enterprise-wide desktop application and system crashes and hangs [38], ), ETW (Event Tracing Windows) and WPP is analized in Chapter 4. On the other hand, kernel Linux, thanks to its huge open source community, has been enhanced with components such as tracepoints, kernel markers and immediate values, in order to reduce the impact in kernel instrumentation. In Chapter 4 Linux tools such as DTrace, SystemTap or Lttng are analized. Finally, hardware breakpoints are described in order to present a special low impact technique for monitoring address memory locations.
Static instrumentation and dynamic activation

A tracer consists of a mechanism collecting an execution trace from a running system, while a trace is a sequence of event records, each identifying that the kernel executed a pre-identified portion of its code.

Mapping between execution sites and events is made possible by instrumentation of the kernel. Instrumentation can be either declared statically at the source code level before it’s running, or dynamically added to the running kernel. Although dynamically instrumentation has some advantages, the impact in the system is bigger than static instrumentation.

A mixed technique could be the best option: statically declared instrumentation which can be enabled dynamically.

A tracer probe is the tracer component called when enabled instrumentation is executed. This probe is responsible for fetching all the data to write in the event record, namely an event identifier, a time-stamp and, optionally, an event-specific payload.

To amortize the impact of I/O communication, event records are saved in memory buffers. Their extraction through an I/O device is therefore delayed. To ensure continuous availability of free buffer space, a ring buffer with at least two sub-buffers can be used. One is used by tracer probes to write events while the other is extracted through I/O devices. [14].

To manage the kernel static instrumentation and decouple it from the tracer, tracepoints and kernel markers infrastructures were created by Mathieu Desnoyers in 2007 [14]. These infrastructures are integrated into the mainline Linux kernel and used extensively by the kernel developers.

Tracepoints and kernel markers are hooking mechanisms providing static instrumentation that can be enabled at runtime with very small footprint when disabled. These tracepoints have been placed at carefully-considered locations which facilitate investigations into what the kernel is actually doing.

Statically declaring an instrumentation site for dynamic activation typically incurs a non-zero performance overhead due to the test and branch required to skip the instrumentation call. To overcome this limitation, Desnoyers created immediate values, which are used by LTtng.

Immediate values were created to statically activate compiled code in an efficient way, by dynamically modifying an immediate operand within an instruction. With these optimizations, Desnoyers has been able to show the Linux kernel community that a static instrumentation approach is viable and could have near-zero overhead when disabled. The better performing prototypes realized within this infrastructure replace a standard load, test and branch by a static jump which skips over the instrumentation unconditionally. Runtime code-patching allows to either activate or deactivate an unconditional jump to dynamically enable branches. This results in completely unmeasurable performance impact for the tracepoints. These prototypes initiated an effort at Redhat to implement the required compiler support for static jump patching.

Hardware Breakpoints

A hardware breakpoint register’s primary (and only) task is to raise a fault when the monitored location is accessed [28]. However these registers are processor specific (Intel 386/486/586/686 family of chips) and their diversity manifests in several forms—layout
of the registers, modes of triggering the breakpoint exception (such as exception being triggered either before or after the memory access operation) and types of memory accesses that can be monitored by the processor (such as read, write or execute).

The CPU has 8 debug registers: 4 for addresses (DR0 to DR3), 2 when debug extensions are enabled (DR4 and DR5), one for status (DR6), and one for control (DR7). Each of the first 4 can hold a memory address that causes a hardware fault when accessed. The control register holds some flags that specify the type of access (read, read-write, execute) and some other bits; the status register is helpful for determining which breakpoint was hit, when handling a system fault.

Three features can be highlighted:

- They can be set to monitor data as well as code.
- Possibly, the biggest convenience of using the hardware debug registers is that it causes no alteration in the normal execution of the kernel or user-space when unused, and has no run-time impact.
- Another key feature of hardware breakpoints is that you can use them to halt on non-execution accesses to memory locations, it means, you can use a hardware breakpoint to tell the processor to stop when a specific variable (address) is read or read/written.

You can also use hardware breakpoints to break in on code execution as well, although in the typical case, it is more common to use software breakpoints for that purpose due to the relaxed restrictions on how many breakpoints you may have active at once because the most notable limitation of this facility is the fewer number of debug registers on most processors [28]. For example, for x86, four hardware breakpoints can be activated at once. However, software breakpoints have the main drawbacks of being slow and intrusive. The debugged program has to be modified, and the debugger needs to memorize the original operation code (opcode) at the modified location, so that the debuggee’s code is restored when the breakpoint is removed. When a software breakpoint is hit, the instruction pointer needs to be decremented, and the original opcode restored. Then the debugged program has to be stepped out of the breakpoint. After the breakpoint is handled, the breakpoint opcode is reinserted [12].

Hardware Breakpoints are not available on some platforms (based on Windows 95) and their management require advanced knowledge or some external tools (like OllyDbg) to obtain the addresses which want to be monitored and to add the hardware breakpoints. However they are quicker than modifying code [13] and it can be used too in order to detect when a program crashed, when other typical techniques are not enough [16].

3.1.2. Virtual Machine

A virtual machine (VM) is a software implementation of a computer that executes programs like a real machine. Hypervisor (also known as virtual machine monitor VMM) is a layer of software (or a combination of software/hardware) that allows to run several independent execution environments in a single computer. The key difference between hypervisor technology and other kind of virtualizations (such as java virtual machine or software emulation) is the performance.
3.1.2.1. Hypervisor

There are two approaches to monitoring virtual machines: requesting each virtual machine (which is called “pooling”), or using a virtualization technology with an hypervisor. In this case you can ask the hypervisor in spite of asking every virtual machine.

Pooling is more flexible than using an hypervisor, however it requires a higher amount of requests, since hypervisor just need one, and then it can request directly to virtual machine. Besides some studies have demonstrated that the employment of a hypervisor-level exclusive buffer cache can allow transparent data access tracing and accurate prediction of the VM page miss ratio curve without incurring into significant overhead [45].

Hypervisor solutions have to introduce a very low overhead; the throughput of the virtual machines has to be very close to that of the native hardware [34].

Some experiences suggest that performing the trace collection at the hypervisor layer is minimally invasive to the collected trace while enabling tracing of the entire system (user/supervisor level, CPU, peripheral devices). The hypervisor is an ideal place to collect full-system execution traces, which can be stored as small deterministic log files and can be selectively expanded into a detailed full-system execution trace during the fast replay [62].

3.1.3. Java Virtual Machine

Java Virtual Machine (JVM) can be monitored and managed by using the interface provided by java.lang.management package [11]. The API provides access to information such as:

- Number of loaded classes and running threads
- Virtual machine uptime, system properties, and JVM input arguments
- Thread state, thread contention statistics, and stack trace of live threads
- Memory consumption
- Garbage collection statistics
- Low memory detection
- On-demand deadlock detection
- Operating system information

This information is represented by MBeans. A MBean has a management interface consisting of attributes that can be read and written, operations that can be invoked and notifications that can be emitted by the MBean. An Oracle overview [11] presents this example:
"An MBean representing an application’s configuration could have attributes representing the different configuration parameters, such as a cache size. Reading the CacheSize attribute would return the current size of the cache. Writing CacheSize would update the size of the cache, potentially changing the behavior of the running application. An operation such as save could store the current configuration persistently. The MBean could send a notification such as ConfigurationChangedNotification when the configuration changes”.

MBeans can be standard or dynamic. A standard MBean exposes the resource to be managed directly through its attributes (using “getter” and “setter”) and operations (the others MBean’s methods). All these methods are defined statically in the MBean interface. A dynamic MBean is an MBean that defines its management interface at runtime, for example, through an XML file. Dynamic MBeans can be used for monitoring both JVM and applications resources.

All MBeans (standard and dynamic) are registered in the MBeans repository, which is called Platform MBeanServer, where each MBean is registered with a unique name.

Since JavaTM SE 5.0, the JRE provides a means to access MBean: one is based on the Java Management eXtensions (JMX) and the other is a small Standards-Based Network Monitoring Technology (SNMP) agent.

The JVM JMX agent is composed by a JMX RMI Connector Server which exposes the content of a the Platform MBeanServer (it means both standard and dynamic MBeans) to remote applications. When started, it registers all the platform MXBeans that compose the JVM Monitoring and Management API in the Platform MBeanServer, and expose them to remote JMX clients. One such client is JConsole.

The JVM SNMP Agent is a small SNMPv3 agent which exposes a single Management Information Base (MIB). In order to manage the JVM through SNMP, the use of an SNMP console (or an SNMP management API) is needed.

JVM-MANAGEMENT-MIB only exposes the information defined by JSR 163 - and none other. Thus, both JVM SNMP Agent and MIB are closed, it means if others MBeans are required, then the next steps are necessary:

1. Find a standard MIB that defines the information you want to expose, or write your own MIB.
2. Use such a tool as the Java Dynamic Management Kit to generate stubs for that MIB
3. Provide the application logic to implement those stubs
4. Start a new SNMP agent to expose your MIB.

Using MBeans-SNMP if just the JVM-MANAGEMENT-MIB MBeans information is required

The SNMP agent is essentially passive. This means that as long as it is not accessed by an SNMP manager, it will not consume any CPU.

The SNMP adaptor itself listens on a DatagramSocket - and uses an internal thread pool - so the cost of a passive agent in terms of resources is essentially that.

The MIB gets also registered by default in the agent - that is, as soon as the JVM starts, but it is a hollow shell - the actual data is computed 'on-demand' when SNMP
requests are received. When it needs to be accessed, the data is pulled from the M&M API \( \text{(java.lang.management MXBeans)} \) and stored in weak caches (this data can therefore be garbaged collected at the next gc invocation) \[20\].

An easy experiment to measure the overhead of JVM monitoring using SNMP can be done starting a JVM with an enabled SNMP agent. It shows:

- 1 more thread.
- Quasi identical memory consumption.
- Identical CPU consumption.
- Around 100 additional loaded classes.

Hence, the use of SNMP protocol would be an efficient option, when just the information required is provided by JVM-MANAGEMENT-MIB MBeans.

**Minimizing MBean-JMX Query Latency**

In order to monitor applications using JVM, more MBeans than JVM-MANAGEMENT-MIB MBeans will be required to gather some application bean data. For example, the JMX support in Spring provides the following features to easily and transparently integrate an Spring application into a JMX infrastructure:

- The automatic registration of any Spring bean as a JMX MBean
- A flexible mechanism for controlling the management interface of the beans
- The declarative exposure of MBeans over remote
- The simple proxying of both local and remote MBean resources

These features are designed to work without coupling an application components to either Spring or JMX interfaces and classes. To expose the properties and methods of a

---

**Figure 3.2.: Architecture of J2SE 5.0 Monitoring and Management Support[33]**
D7.1: Global performance, security and privacy requirements

bean as an MBean, an MBeanExporter class must be configured and referenced in the configuration file, as shown in Figure 3.3.

When the application is running on a container which has its own MBeanServer, Spring can attempt to locate the running MBeanServer and register the beans with that server. If no server is specified, the MBeanExporter tries to automatically detect a running MBeanServer. This works in most environments where only one MBeanServer instance is used, however when multiple instances exist, the exporter might pick the wrong server. In such cases, one should use the MBeanServer agentId to indicate which instance to be used [50] .

In the standard Java JMX implementation, a MBean query waits to return until the data are obtained, and uses CPU cycles while waiting. For example, querying 12000 MBeans, the wait time on an average Linux box could be as much as 60 or 120 seconds [32]. Three options are presented in order to minimize this wait time (using the standard Java JMX implementation):

- To avoid querying all of the MBeans at the same time or to query them only on demand.
- To query only a subset of some types of MBean.
- To poll them all, but only do it once every 5 minutes or so to minimize load on the system.

JConsole can be used to visualize JMX-MBean information. Most of common problems in Java SE applications can be detected using Jconsole, like we can see in Figure 3.4.

3.1.4. Desktop application monitoring

RCP applications can be monitored using the Observer software engineering pattern. The pattern defines a 1 to many dependency between objects. When an object, the subject,
### Common Problems in Java SE Applications

Typically, problems in a Java SE application are linked to critical resources such as memory, threads, classes, and locks. Resource contention or leakage may lead to performance issues or unexpected errors. Table 1 summarizes some common problems and their symptoms in Java SE applications and lists the tools that developers can use to help diagnose each problem’s source.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Symptom</th>
<th>Diagnostic Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insufficient memory</td>
<td>OutOfMemoryError</td>
<td>Java Heap Analysis Tool (jhat)</td>
</tr>
<tr>
<td>Memory leaks</td>
<td>Growing use of memory</td>
<td>Java Monitoring and Management Console (jconsole)</td>
</tr>
<tr>
<td></td>
<td>Frequent garbage collection</td>
<td>JVM Statistical Monitoring Tool (jstat)</td>
</tr>
<tr>
<td></td>
<td>A class with a high growth rate</td>
<td>Memory Map (jmap)</td>
</tr>
<tr>
<td></td>
<td>A class with an unexpected number of instances</td>
<td>See jmap -histo option</td>
</tr>
<tr>
<td></td>
<td>An object is being referenced unintentionally</td>
<td>jconsole of jconsole with jhat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See jconsole -deep option</td>
</tr>
<tr>
<td>Findbugs</td>
<td>Objects are pending for finalization</td>
<td>jconsole</td>
</tr>
<tr>
<td></td>
<td></td>
<td>jconsole with jhat</td>
</tr>
<tr>
<td>Deadlock</td>
<td>Threads block on object monitor or</td>
<td>jconsole</td>
</tr>
<tr>
<td></td>
<td>java.util.concurrent tasks</td>
<td>Stack Trace (jstack)</td>
</tr>
<tr>
<td>Leaking threads</td>
<td>Thread CPU time is continuously increasing</td>
<td>jconsole with JTop</td>
</tr>
<tr>
<td>High lock contention</td>
<td>Thread with high contention statistics</td>
<td>jconsole</td>
</tr>
</tbody>
</table>

Figure 3.4.: Common Problems in JavaSE Applications

changes its state it notifies all of its dependents automatically, the observers. Figure 3.5 presents the UML diagram of the observer pattern.

The pattern can be easily applied to the monitoring of RCP applications. Different RCP changes can be monitored through so called listeners. Different types of listeners exist. Listeners subscribe themselves to lists in order to be notified when specific changes occur in the RCP application. Listeners can be mapped to the observers in the above mentioned pattern. Packages involved with the monitoring of changes in RCP applications are the following:

- **org.eclipse.ui**: Provides listeners for different events related to RCP’s user interface elements, such as, activated parts of the windowbench, detection of the manipulation of windows belonging to the application.

- **org.eclipse.core**: Provides listeners for different events related to the RCP applications, such as, save, open, copy, paste, etc. executed on the application’s workbench.

The performance related to the monitoring of RCP applications can be enhanced by subscribing listeners in an intelligent matter, that is until certain event(s) trigger their subscription. This allows for the saving of resources.

Java also provides binary flags in order to enhance the performance of intensive sensing applications. The detection of certain events and its details sometimes needs several comparisons that are done in code typically through if statements where certain variables are compared. The Eclipse Java Development Toolkit (JDT) Project offers binary flags to encode the needed information. Binary flags provide a more efficient comparison with logical AND and OR operators, than the traditional comparison of non-binary values. An integer can be used to store 32 flags. The two flags F1 and F2 could be used in the following way:
Using an integer to store less than 32 flags, the OR operator can be used like this:

\[ X = (F_1 \lor F_2) = 001001 \]

To evaluate if the set \( X \) contains the flag \( F_1 \), the AND operator can be used:

\[ X \land F_1 = F_1 = 000001 \]

Using this method of comparison, a set of flags can be compared with one AND operation:

\[ X \land \text{SET} \neq 0 \]

### 3.1.5. Web application, monitoring the browser environment

In this section we will focus on the browser as the environment where monitoring is going to be performed, in other words, the client-side runtime environment of web applications. Looking at the surface of existing techniques used for remotely collecting client-side runtime information, we will look at the performance impact that these techniques include in the system, evaluating this impact and looking for alternatives or modifications that can minimize it.

Related work in this area has been filled with some projects instrumented to find performance bottlenecks in web applications, like ParaDyn\[40\], which uses dynamic, adaptive instrumentation for performance data monitoring. Other projects, like BrowserShield and CoreScript\[63\], have been oriented to enforce browser security. Most of these techniques are based on JavaScript rewriting, for example, BrowserShield is implemented with a JavaScript parser that is executed in the client browser in order to protect the application against malicious generated code. Other example of these applications is AjaxScope \[27\], and are based on the concept of instant redeployability, that is to say, the ability...
to dynamically serve new and different versions of code each time any user runs a web application.

The study performed by Kiciman et al.[27] with the AjaxScope platform, investigated the logging overhead of instrumented web applications. They assessed the critical path latency of logging a message within an instrumented program (injecting Javascript code). They measured the total execution time of a loop of 10,000 iterations of a function writing a log message, first without instrumentation and then with it, and with two different browsers: Internet Explorer 7.0 and FireFox 1.5. Finally, the results in figure 3.6 show that the overhead of logging a single message is approximately 0.01-0.02 ms.

<table>
<thead>
<tr>
<th>Browser</th>
<th>w/out Instrumentation</th>
<th>w/Instrumentation</th>
<th>Per-message overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>std.dev.</td>
<td>mean</td>
</tr>
<tr>
<td>IE 7.0</td>
<td>80</td>
<td>30</td>
<td>407</td>
</tr>
<tr>
<td>FireFox 1.5</td>
<td>33</td>
<td>14</td>
<td>275</td>
</tr>
</tbody>
</table>

Figure 3.6.: Overhead of message logging across browsers (times in ms)[27]

Choudhary et al.[9] introduced an interesting approach for remote monitoring of client-side code, that can be used in several scenarios, like error detection and debugging, client-side metrics collection and memory profiling (count of live variables, objects and arrays in the client code).

This technique is based, like CoreScript[63] or AjaxScope[27], on the injection of a Javascript Client-side Agent (CSA) into the code of the web application to be monitored. The injection of the CSA is implemented with the use of a rewriting reverse proxy.
The process is shown in Figure 3.7, where the client browser is just a JavaScript enabled web browser. When the client browser sends a request, the reverse proxy will direct it to the web server, as usually. After that, the web server sends a response, which is intercepted by the proxy, adding the client-side agent to the code to be sent to the client browser. After the code is instrumented with commands, the reverse proxy is configured to route HTTP requests containing monitoring data to the Command and Control Server (CnC), and all other traffic without monitoring data is routed to the Web Server.

All this process may slow down the client-side code of the web application under monitoring, first of all, because the proxy imposes an overhead of time while instrumenting pages of the application (inserting Client-side Agents), and second, because the CSA iterates through all members of the window object for collecting data, which also affects in the users of the application.

This impact has been measured and the results are available in the article written by Choudhary[9]. The evaluation of CSA injection has been done over 10 web applications characterized by having a large proportion of Javascript client-side code.

The first of the measurements gives response to the influence of the proxy's injection, focusing on the time needed to instrument one randomly chosen page of each of the applications (with and without injection). Each page has been loaded 100 times, and the measurements are averaged. The results shown in Table 3.1, reveal that the proxy takes up to 8 milliseconds long to inject the agent, which is negligible in the overall process.

Other interesting analysis is the cost of runtime monitoring, measuring the overhead imposed by the CSA that iterates through all members of the window object (in Javascript is the top level object) and inspects all global elements in the browser window. We can see in the table 3.2, the evaluation in terms of number of objects that can be accessed by the CSA and the absolute time required to access them. The results show that the agent
### D7.1: Global performance, security and privacy requirements

<table>
<thead>
<tr>
<th>Subject</th>
<th>Number of objects</th>
<th>Difference between with and without CSA injection (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive Test</td>
<td>147</td>
<td>2</td>
</tr>
<tr>
<td>Number Guess</td>
<td>144</td>
<td>3</td>
</tr>
<tr>
<td>Chat Client</td>
<td>147</td>
<td>2</td>
</tr>
<tr>
<td>Mail</td>
<td>1286</td>
<td>9</td>
</tr>
<tr>
<td>Showcase</td>
<td>4490</td>
<td>30</td>
</tr>
<tr>
<td>Joomla</td>
<td>229</td>
<td>3</td>
</tr>
<tr>
<td>Drupal</td>
<td>118</td>
<td>1</td>
</tr>
<tr>
<td>Wordpress</td>
<td>176</td>
<td>3</td>
</tr>
<tr>
<td>iGoogle</td>
<td>618</td>
<td>6</td>
</tr>
<tr>
<td>Amazon.com</td>
<td>314</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.2.: CSA accessing all the elements of the window (overhead)

```html
<!DOCTYPE HTML>
<html>
<head>
  <title>Worker example: One-core computation</title>
</head>
<body>
  <p>The highest prime number discovered so far is: <output id="result"></output></p>
  <script>
    var worker = new Worker('worker.js');
    worker.onmessage = function (event) {
      document.getElementById('result').textContent = event.data;
    };
  </script>
</body>
</html>
```

Figure 3.8.: Web Worker example (running the calculation of a prime number in a different thread)

was able to monitor the objects’ state and provide information to the CnC server in an efficient way (without affecting the experience of the users).

These results show that the performance impact of the web monitoring technique is quite satisfactory. Nevertheless, as Kiciman[27] concluded, if we use this method adding timestamp logging to the entry and exit points of every Javascript function defined in a program, it can reach a very high overhead, resulting in added CPU time as well as network bandwidth for reporting observations.

Hence, these results must be improved and one of the possibilities to optimize it is the use of one of the characteristics of the new HTML standard, particularly with Web Worker threads. These Web Worker threads can help to minimize the effect of the Client-side Agent, who slows down the client-side code of the web application performing a computation-intensive operation. Web Worker threads are long-running scripts in the background of the application runtime independently of any user interface scripts. These threads allow performing a computationally expensive task without interrupting the user interface. An example of Web Worker is shown in figure 3.8.

With the use of this independent thread technique, a CSA can be started as an independent worker thread, operating in parallel with the other client-side code and improving the performance to a higher extent. Event though in some cases this technique would not be necessary because of a low-impact of some CSA agents, it is strongly recommended to schedule the execution of these agents in a parallel thread, minimizing any influence over user interface execution or other client-side code of the application.

3.2. Execution environment independent monitoring

In this section, we present some techniques for reducing or constraining the overhead produced by monitoring, independently from the execution environment where this monitoring has to be done. In that sense, general approaches will be shown with different techniques for controlling the impact in CPU or memory usage, among other controllable resources. These techniques, that can be applied to any system interface or API, will show that it is possible to control the overhead due to software monitoring while achieving high accuracy in the monitoring results.

In addition to these techniques, some Java programming tips are introduced to reduce FastFix Client performance impact. These suggestions improve the performance of the garbage collector and the run-time execution environment overhead; and they can be apply to any Java application.

3.2.1. Software Monitoring Controllable Overhead

Software Monitoring Controllable Overhead (SMCO) is a general monitoring technique that can be applied to any system interface or API. It is based on temporarily disabling monitoring of selected events for as short time as possible, with the constraint of a user-supplied target overhead.

These techniques first started with the name of software monitoring with bounded overhead, particularly, with a general technique called High-Confidence Software Monitoring[5] (HCSM). Callanan et al. [5] described HCSM as a general technique for limited overhead software monitoring. First, it is general since its instrumentation can be attached at any system interface or API.

In order to achieve this, a generic controller implements an optimal control strategy for controlling the overhead. The instrumentation of the system also maintains an estimate of the rate of accesses to the interface, as well as the time spent handling these accesses.

The control theory is based on finite state machines (Figure 3.9). In the figure, P represents a process (a.k.a plant), with controllable input \( v \) and output \( y \), and a reference input \( x \). The controller design problem is that of designing a controller \( Q \), with inputs \( x \), \( y \) and output \( v \), such that the composition of \( Q \) and \( P \) follows the reference input (i.e., \( y \) is nearly equal to \( x \)) with good dynamic response and small error. This control theory is applied in the SMCO technique for runtime monitoring with controlled overhead, since it can be beneficially stated as the previously explained controller design problem, where the controller is the runtime monitor, the plant is a software application and the reference input \( x \) is the target overhead \( O_{\text{target}} \). To ensure that the plant is controllable, one typically instruments the application so that it emits events of interest to the monitor. The monitor catches these events, and controls the plant by enabling or disabling event sig-
D7.1: Global performance, security and privacy requirements

The main motivation of this technique is using control theory to avoid an ad hoc approach to overhead control, making it general and achieving good results in runtime monitoring of software systems.

The structure of SMCO is illustrated in more detail in figure 3.10. The key of the system is located in the SMCO controller, which is in charge of periodically send enable/disable monitoring commands to the monitor associated to an instrumented program, the target of the monitoring. The controller will enable/disable monitoring in such a way that the monitoring overhead never exceeds the target overhead $O_{\text{target}}$. Current observed overhead feeds the controller back, so it has the information to disable monitoring when the observed overhead is close to $O_{\text{target}}$ and enable monitoring when the difference between the observed overhead and the target is enough.

The way the overhead is measured in runtime monitoring is based on monitoring percentage, in other words, the percentage of execution time spent monitoring events. Overhead is related with this monitoring percentage, since it is the measure of how much longer a program takes to execute because of monitoring. If an unmodified program executes in time $U$, and executes in total time $U + M$ with monitoring, then we say that monitoring has overhead $M/U[23]$. Monitoring percentage is then derived from overhead with the following equation: $m_{r} = \alpha_{t}/(1+\alpha_{t})$ [23].

The results of HCSM technique shows that the most obvious limitation is saturation: regardless of the kind of instrumentation, once the instrumented system hits a bottleneck (usually the processor), it will reach a peak service level. More instrumentation will simply reduce the service level, and as a result we will not be able to observe more than a particular threshold of events. Figure 9 shows observed overhead versus target overhead.
for our load benchmark of the Lighttpd server, with both bounds checking and memory-underline-utilization detection turned on. Huang ran Lighttpd with target overheads from 5% to 100% in increments of 5%. The solid line shows the observed percent overhead (y axis), which should ideally adhere to the y = x line. The dotted and dashed lines show the number of processed function call events and memory access events respectively, in millions of events (y2 axis). Huang uses the curl-loader tool to hit the server with one request per second from 25 simulated clients. He observes roughly linear growth for target overhead settings between 5% and 60%, with the saturation point lying at 60%. Huang observes that saturation accompanies a rapid increase in the number of memory access events observed.
In the last two years, this technique has evolved, improving results with the study of SMCO, and it has been improved with a cascade controller\cite{23}.

The architecture of the cascade controller is shown in figure 3.12. It consists of a set of secondary controllers $Q_i$, each of which directly control a single plant $P_i$, and a primary controller $PQ$ controlling the inputs $x_i$ to the secondary controllers.

The difference between global and cascade controllers is the distribution of overhead across different event sources. The results of observed overhead with the cascade controller
are closer to the ideal, as we can check in figure 3.13.

In all cases, SMCO results show that in all cases, it was able to track the target overhead for a wide range of values, even when there were too few events during execution.

This technique can be especially helpful when setting a low overhead, which will be commonly our case, but results of the whole range of target overhead shows how it can be controlled, no matter the number of events being monitored.

3.2.2. Java Performance Tuning

Another approach of Execution Environment Independent Monitoring is the suggestion of some ways to improve the performance of FastFix’s Client [48]. This tips can be applied to any Java Application.

In order to minimize execution environment overhead:

- Avoiding growing files
- Clustering files together
- Minimizing CPU contention
- Timing out processes

In order to reduce object creation and garbage collector overhead:

- Avoiding repeated string concatenation, or using StringBuffer instead of String concatenation
- Pre-size collection data structures
- Keep caches of frequently-used values rather than reallocating them
- Avoiding temporary objects
- Eliminate references to objects promptly when you no longer need them.

- The usage threshold [33] is a manageable attribute of memory pools. It enables the monitoring of memory use with low overhead. Setting the threshold to a positive value enables usage threshold checking for a memory pool. Setting the usage threshold to zero disables usage threshold checking. The default value is supplied by the JVM. A JVM performs usage threshold checking on a memory pool at the most appropriate time, typically during garbage collector and sometimes at allocation time. If the JVM detects that the current memory usage exceeds the usage threshold, it will set the UsageThresholdExceeded attribute to true. Some memory pools may not support the usage threshold.
You can use the `UsageThresholdSupported` attribute to determine whether a memory pool supports a usage threshold. For example, in a generational garbage collector (such as the HotSpot virtual machine), most of the objects are allocated in the young generation, from the "eden" memory pool, which is the pool from which memory is initially allocated for most objects. The eden pool is designed to be filled up; performing garbage collection on the eden memory pool will free most of its memory space since it is expected to contain mostly short-lived objects unreachable at garbage collection time. So, having the eden memory pool to support the usage threshold is not only not useful but also might not be implemented efficiently.

These attributes can be seen and setted in MBeans tab of J2SE Monitoring and Management Console, as we can see in Figure 3.14.

**In order to reduce or eliminate unnecessary run-time overheads:**

- Avoiding speculative casts, system paging and unnecessary assignments
- Converting recursion to iteration
- Cutting dead code
- Eliminating null tests, prints and unnecessary variables
- Initializing variables only once.
- Moving loops to native routines
- Threading slow operations, if it’s possible.

**Others**

- Altering process priorities to increase the amount of time allocated by the CPU to selected processes
- Using HashMap instead of Hash table to improve performance
- Using prepared statements to speedup SQL queries
3.3. Application Recording

Replaying software execution is a requirement for providing remote software maintenance. If a software application was a completely predictable automaton, there would be no need to record their execution in order to replay it. Replaying software requires monitoring the application to be replayed because some parts of most programs are non-deterministic. A non-deterministic operation (e.g., generating a random number) is an operation that, every time it’s repeated, may have a different result and therefore its re-execution may be different than the original execution. However, if one is replaying an application in order to debug it, it is essential that the re-execution leads to the same bug as the original execution. The main sources of non-determinism in software are: user input data, some system calls, thread execution order, signals, data read from uninitialized locations and some processor specific operations.

Recording application execution for debugging has been approached in many ways. The main reason for the diversity of approaches is the high performance cost of recording each operation a program does. It is easy to see that if all memory operations are repeated as writes to a log, this will lead to overheads of more than 100%. There have been four general directions taken in attempting to reduce the monitoring overhead:

- Implement monitoring in specialized hardware [61, 41, 42]. This approach has the significant downside of requiring major modifications to the platform where the application is running and is clearly outside the scope of FastFix.

- Recording only non-deterministic operations: If the monitoring system is able to distinguish deterministic from non-deterministic operations clearly this can be a significant way of reducing recording overhead.

- Recording the least information needed to allow the replication of the fault: Most error replication methods assume that the error must be replayed at the maintenance site at the first attempt. PRES [44] is able to monitor applications, in particular, thread scheduling at different precision levels which later imply different numbers of replay executions until the correct execution is replayed.

- Statistical debugging is a recent approach [31, 25] that aims at isolating bugs by analyzing information gathered from a great number of users. This idea is based on the notion that software applications are usually executed by a large user community. Therefore, instead of trying to detect the bug relying only in data from executions from a single user, statistical debugging attempts to speed up the bug tracking process by distributing the monitoring across different clients. By doing this it is possible to extract afterwards patterns of similarity among the universe of executions collected that could lead to the failure.
4. Related projects and tools

4.1. Tools

4.1.1. Windows

System Center Desktop Enterprise Monitoring

System Center Desktop Enterprise Monitoring, also called DEM [38], is a Microsoft Desktop Optimization Pack (MDOP) solution, easy to use, which monitors and reports enterprise-wide desktop application and system crashes and hangs.

DEM is oriented to desktop applications to solve the problem that when an application experiences an error it is usually resolved by rebooting or reopening the application, hence IT is not aware of the issue, and any data that IT could have used to help debug the issue is usually lost. These reboots causes a productivity loss too, as users have to wait for the PC reboot and get back to the point in the application where they left off.

The most special feature of this tool is the method that uses to collect data, because it is an agentless method. It means that is not necessary install anything on the desktop or visit any machines to set it up. DEM uses Group Policy and Windows Error Reporting, which is built into Windows, to send reports to a designated server within the organization for diagnosis.

Using Group Policy, an IT administrator can capture these errors without ever having to deploy an agent to the client computer.

Windows Error Reporting causes the pop-up message that appears on end users computers when applications hang, asking them if they want to send an error report about the problem to Microsoft. DEM leverages this technology to redirect these reports to the designated server.

If desired, the system can be also configured so that the designated server passes crash data on to Microsoft’s Error Reporting servers as well as to have DEM automatically download the latest troubleshooting data based on specific errors. DEM can help you find out if a particular driver is causing issues with desktops in your enterprise and help you find an updated driver.

WPP Software Tracing

WPP Software Tracing simplifies the use of WMI event tracing to implement efficient software tracing in drivers and applications that target Windows 2000 and later operating systems.

It is an efficient mechanism to log real-time binary messages for this trace provider:

- Kernel-mode driver.
- User-mode application or dynamic-link library (DLL).
WMI event tracing is supplemented and enhanced by WPP software tracing in kernel-mode drivers by adding conventions and mechanisms that simplify tracing a driver’s operation.

Logging messages with WPP software tracing is similar to using Windows event logging services. The driver logs a message ID and unformatted binary data in a log file. Then, a postprocessor converts the information in the log file to a human-readable form. However, WPP software tracing supports message formats that are more capable and flexible than that supported by the event logging services. For example, WPP software tracing has built-in support for IP addresses, GUIDs, system IDs, time stamps, and other useful data types. In addition, users can add types relevant to their application.

In order to use the default form of WPP in a trace provider, we must define a control GUID that uniquely identifies the trace provider, then add the required WPP-related C preprocessor directives and WPP macro calls to the provider’s source files, build the driver and then use other tools, such as TraceView, Tracelog, Tracefmt and Tracepdb to configure, start and stop tracing sessions and to display and filter trace messages. These tools are included in the Windows Driver Kit (WDK).

4.1.1.1. ETW (Event Tracing Windows)

Event Tracing for Windows (ETW) [37] provides application programmers the ability to start and stop event tracing sessions, instrument an application to provide trace events, and consume trace events.

It provides holistic view of the system with high speed: 1200 to 2000 cycles per logging event and low overhead: less than 5% of the total CPU cycles for 20,000 events/sec.

Trace events contain an event header and provider-defined data that describes the current state of an application or operation. You can use the events to debug an application and perform capacity and performance analysis.

ETW works for both user mode applications and drivers. Tracing sessions and event provider are separated. ETW can be dynamically enabled or disabled and it’s designed to allow tracing of production code.

Both, ETW and WPP support most types of trace providers, however, they use types that are not available for certain types of drivers, such as miniport drivers.

4.1.2. Linux

LTTng

LTTng (Linux Tracing Toolkit, next generation) is a high performance tracing tool for Linux that efficiently handles large amounts of trace data. Initially aimed at the Linux kernel, its technology has been extended to support user space tracing (UST). LTTng uses technologies like Tracepoints, Markers and Immediate Values, which are described in section 3.1.1

Apart from LTTng’s kernel tracer and userspace tracer, viewing and analysis tools are part of the project. The LTTV viewer permits to analyze and show extremely large traces (>10GB) [46], both in text format and graphically.

The LTTng Architecture can be examined in Figure 4.1

Some interesting features are now under development, for example, the remote control of tracing and the integration of LTTng in a Eclipse project.
D7.1: Global performance, security and privacy requirements

Today, we are be able to control remotely using Target Communication Framework (TFC) with a LTNg client library and the LTNg agent on the other side, the tracing behaviours such as list tracepoints and start trace. According to LTNg developers, they are improving the integration side with both the LTNg and UST tools and are working on streaming traces over the network using TCF so we'll be able to collect trace data in real time.

LTNg belongs to an Eclipse project called Linux Tools Project-LTNg Integration. The scope of the project is to provide an Eclipse integration of LTNg, in particular its LTTv component [51].

The project will be delivered in two components:

- A Tracing and Monitoring Framework (TMF), which will be described in section 4.2
- An LTNg reference implementation based on TMF.

Some of the main features of the LTNg integration, from a user perspective, are the control of the LTNg tracer running on a target node, efficient retrieval and handling of LTNg trace files, support for distributed, multi-core and multi-processor traces synchronization, standard LTTv trace visualization, correlation and analysis views and support for kernel and user space tracing.

Dtrace

Sun’s DTrace facility can be used to examine the behaviour of user programs and of operating system [18]. It is a dynamic tracing facility, meaning that it can be used to insert tracepoints at any location in the kernel. DTrace can be used on live production servers with often negligible impact on performance [10].
D7.1: Global performance, security and privacy requirements

The DTrace framework provides instrumentation points that are called probes. A DTrace user can use a probe to record and display relevant information about a kernel or user process. Each DTrace probe is activated by a specific behaviour.

Probes are implemented by providers. A probe provider is a kernel module that enables a given probe to fire. For example, the function boundary tracing provider fbt provides entry and return probes for almost every function in every kernel module.

A DTrace consumer is any process that interacts with the DTrace framework. While dtrace(1M) is the primary DTrace consumer, other consumers exist. These additional consumers mostly consist of new versions of existing utilities such as lockstat(1M). The DTrace framework has no limit on the number of concurrent consumers.

The behaviour of DTrace can be modified with the use of scripts that are written in the D language, which is structured similarly to C. The D language provides access to kernel C types and kernel static and kernel global variables. The D language supports ANSI C operators [39].

DTrace’s schema can be examined in Figure 4.2.

**SystemTap**

The SystemTAP project from Redhat, first made available in 2005, aims at letting system administrators run scripts connected at specific kernel sites to gather information and statistics about the system behaviour and investigate problems at a system-wide level [14].

It’s a dynamic method of monitoring and tracing, because instead of building a special kernel with instrumentation SystemTap allows you to install that instrumentation dynamically at run time. It does this with an application programming interface called Kprobes, for dynamic probes on addresses inside the kernel; and uprobes, for dynamic probes in user space programs. This provides a unified way of probing and then collecting data for observing the whole system. To dynamically find locations for probe points, arguments of the probed functions and the variables in scope at the probe point, SystemTap uses the debuginfo (Dwarf) standard debugging information that the compiler generates [57].

The stap program is the front-end to the SystemTap tool. It accepts probing instructions written in its scripting language, translates those instructions into C code, compiles this C code, and loads the resulting kernel module into a running Linux kernel to perform the requested system trace or probe functions. You can supply the script in a named file, from standard input, or from the command line. The SystemTap script runs until one of
the following conditions occurs:

- The user interrupts the script with a CTRL-C.
- The script executes the exit() function.
- The script encounters a sufficient number of soft errors.
- The monitored command started with the stap program’s -c option exits.

Figure 4.3 presents the basic flow of the SystemTap process.
ReTrace

It’s a trace collection tool based on VMware’s virtual machine monitor and deterministic replay. ReTrace has extremely low run-time overhead and high trace file compression ratio [62].

ReTrace operates in two stages: capturing and expansion. ReTrace capturing accumulates the minimal amount of information necessary to later recreate a more detailed execution trace. It captures (records) only non-deterministic events resulting in low time and space overheads (as low as 5% run-time overhead, as low as 0.5 byte per thousand instructions log growth rate) on supported platforms. This reduces the necessary trace file size and trace distortion, because the capturing overhead is minimal.

ReTrace expansion uses the information collected by the capturing stage to generate a complete and accurate execution trace without any data loss or distortion.

Figure 4.5 shows the run-time impact of ReTrace capturing. Run-time in the VM without and with ReTrace are compared. The run-time overheads of ReTrace capturing range from 0.7% to 31% with a geometric mean of 5.09%. This shows that ReTrace has extremely low overhead for CPU intensive workloads.

4.2. Projects

4.2.1. Health Center

The IBM Monitoring and Diagnostic Tools for Java - Health Center (Health Center) is a very low overhead monitoring tool which enables to assess the current status of a running Java application. Health Center continuous monitoring provides information to identify and help resolve problems such as identifying if memory is leaking, discovering the methods which are taking most time to run and viewing any lock contentions. Health Center is also used to optimize application performance, improve application stability and uptime, reduce system resource usage and the time to resolve problems.
D7.1: Global performance, security and privacy requirements

It uses a sampling method profiler, without recompilation or binary instrumentation. It provides information, such as:

- Information relative to garbage collector and if it’s causing performance problems.
- A full call stack information for all sampled methods, showing where the application is spending its time.
- If the application is using more memory than seems reasonable, including where memory leaks occur.
- Class loading information to detect an excessive class loading.

When a trouble is detected, Health Center suggests how you can solve it. For example, if a garbage collector performance problem is detected, it suggests the most appropriate command line options in order to improve the performance.

Health Center uses an 'environment perspective' to provide details of the Java version, Java classpath, boot classpath, environment variables, and system properties. This is particularly useful for identifying problems on remote systems or systems where you do not control the configuration. If the Health Center detects misconfigured applications, it will provide recommendations on how to fix it.

Health Center can save the data obtained from monitoring an application and load it again for analysis at a later date.

4.2.2. Tracing and Monitoring Framework (TMF)

The Tracing and Profiling Tools Project is a project in the Eclipse Test and Performance Tools Platform (TPTP) Top-Level Project (TPTP project provides an open platform that allow software developers to build unique test and performance tool, both open source and commercial, that can be easily integrated with the platform and with other tools), it addresses the tracing and profiling phases of the application lifecycle.

The Tracing and Profiling Tools Project provides frameworks (TMF) for building tracing and profiling tools by extending the TPTP Platform. The framework contains views, dialogs and action items that support the capability of collecting and analyzing application performance information.

TMF provides extensible support for [6]:

- Tracing and Monitoring tool discovery and control, which purpose is to identify the available trace providers and their capabilities, and use this information to generically control the tools (start/stop/pause/resume/...). Two main features of tool control, in order to minimize the monitoring impact, are the tracing rate regulation (throttling) and the budget policy, to avoid congestion on the target, host and transport and constraint target resource usage such as CPU, memory and bandwidth.

- Data retrieval, storage and visualization, which purpose is to collect and store tracing/monitoring data from the tool and provide a set of standard data visualization tools (such as a generic log file interface), generic monitoring views (such as event logs, CPU/Memory/Heap/Network usage) and generic graphical widgets (charts, histograms, etc). Data visualization will be extensible to allow for application-specific contents.
Analysis and correlation tool integration whose purpose is to provide basic analysis functions and support host-based and external analysis tools and libraries. A key feature is the causal dependency analysis, which includes event dependency tree, critical path, correlation of event data, reconstruction of event sequences from related traces and execution replay. Some external tools integration, such as tools to send the analysis results to UI views or widgets are included too at this point.

The project includes exemplary profiling tools for both single-system and distributed Java applications through monitoring agents that collects trace and profile data. A generic tool kit for customizable probe insertion is also available.
5. FastFix Global Performance Requirements

5.1. Introduction

Before we can determine the desired performance characteristics of an application, two things have to be understood: what the application does and how we expect it to be used. Then the most critical business transactions must be identified in terms of frequently used transactions, performance-intensive transactions and business-critical transactions.

Once most of the business transactions are identified, we can begin the process of verbalizing performance requirements, goals and performance-testing objectives.

Using Scott Barber's terminology, in order to capture performance requirements, in general, we just must review contracts and legally binding agreements related to the software under development and get executive stakeholders to commit to any performance conditions that will cause them to hold up release of the software into production. All the business transactions that are related with a performance requirement must be included in performance testing. Apart from these business transactions, the easiest way to capture performance-testing objectives is to ask each member of the team what he or she would like tested.

However, Barber says that performance goals are tricky to both capture and quantify, and so capturing and quantifying goals should be treated as separate activities, first table capturing performance goals in qualitative terms, and then quantifying them. He recommends capturing performance goals for both critical business transactions and the application as a whole in subjective. For example, a goal in qualitative terms could be “no slower than the previous release”, to quantifying it we just have to execute the equivalent performance test against the previous release. Another closely qualitative term is based in end-user satisfaction and/or frustration. To quantify it just a prototype or demo and some representative users are needed.

According with the gathered information in Munich workshop, the most critical business transactions of FastFix could be:

- Frequently used transactions: Gathering data from run-time execution environment, visualizing traced data (application, user and context information).
- Performance-intensive transactions: Recording data, event correlation (pattern recognition and possible causes and symptoms).
- Business-critical transactions: Reporting the error to IT, patch generation, fault replication: availability for replay.

This list is used to identify the high level global performance requirements, which must be reviewed every time that functional requirements are reviewed.
### 5.2. Requirements

The following list of requirements is established based on the concept of FastFix usability, which must be first understood as the accomplishment of a specified Service Level Agreement, if available. In other cases, usability level will be established according to the end user satisfaction and acceptance, following Barber recommendations.

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Type</th>
<th>Global Performance Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gathering data from run-time execution environment</td>
<td>Frequently-used transaction</td>
<td>Must be achieved without penalizing users workflow and experience. Once an abnormal situation appears, all the useful information for the software maintainer will be gathered granting FastFix usability.</td>
</tr>
<tr>
<td>2</td>
<td>Recording data from run-time execution environment</td>
<td>Performance-intensive</td>
<td>Regular application execution must not be impacted. Data associated to recording must be gathered granting FastFix usability.</td>
</tr>
<tr>
<td>3</td>
<td>Reporting the fault to maintainers</td>
<td>Business-critical transaction</td>
<td>The software maintainer must be aware of the fault or abnormal situation at least simultaneously to the end user.</td>
</tr>
<tr>
<td>4</td>
<td>Visualizing traced data (application, user and context information) of fault</td>
<td>Frequently-used transaction</td>
<td>Fault related data will be available for the software maintainer granting FastFix usability.</td>
</tr>
<tr>
<td>5</td>
<td>Visualizing possible symptoms/causes of the error</td>
<td>Performance-intensive</td>
<td>Identified possible causes and symptoms associated to the fault must be visually presented to the software maintainer granting FastFix usability.</td>
</tr>
<tr>
<td>6</td>
<td>Recognition of associated patterns during abnormal situations</td>
<td>Performance-intensive</td>
<td>During event correlation process, some patterns will be recognized during abnormal situations. In case of pattern recognition, these patterns will be available according to FastFix usability.</td>
</tr>
<tr>
<td>7</td>
<td>Fault replication: availability for replay</td>
<td>Business-critical transaction</td>
<td>Once all recording data has been gathered, replay action will be available according to FastFix usability. Any delay introduced by the fact that the environment is not the original, must not affect the fault replication.</td>
</tr>
<tr>
<td>8</td>
<td>Patch generation</td>
<td>Business-critical transaction</td>
<td>When a case is well-known and a solution is available to be applied, patch generation will be performed allowing the application of the patch according to FastFix usability.</td>
</tr>
</tbody>
</table>

Table 5.1.: Global Performance Requirements
D7.1: Global performance, security and privacy requirements

In order to implement requirements 1 and 2, a summary from Chapter 3 are presented and must be followed, when possible.

- Gathering data:
  - Java performance tuning tips explained in section 3.2.2 will be properly followed, when possible.
    * Autocontrol policies will be evaluated and implemented, when possible, at monitoring level and hence at overhead level, with two different approaches:
      - When the environment is overloaded, the monitoring level will be reduced, with the minimum necessary amount of agents to ensure monitoring.
      - When a situation of possible error is detected, the monitoring level will be increased, as long as the environment performance allows for it.
    * When using instrumentation techniques, these must be as little intrusive as possible, relying on static instrumentation of binaries and their dynamic activation.
    * When monitoring the JVM, SNMP should be used instead of JMX because with JMX more objects must be loaded and more threads are necessary in order to monitor JVM. This applies if the only information required is the default MIB of the JVM.
    * When monitoring Java applications or even the JVM itself using JMX, the proper recommendations listed in 3.1.3 will be followed in order to reduce the overhead:
      - To avoid querying all of the MBeans at the same time or to query them only on demand.
      - To query only a subset of some types of MBean.
      - To poll them all, but only do it once every 5 minutes or so to minimize load on the system.
    * In the case of monitoring a web application, we recommend the use of Web Worker threads discussed in section 3.1.5 to monitor the browser environment, because this kind of threads are long-running scripts in the background of the application, so they allow performing computationally expensive tasks, like monitoring tasks, without interrupting the user interface.
  - Recording data.
    * Recording the least information needed to allow the replication of the fault. For example, recording only non-deterministic operations if deterministic and non-deterministic operations can be distinguished.
    * The use of statistical debuggers.
Part II.

Security and privacy requirements
Introduction

There has been an exponential increase in new technologies usage, and a growing dependence on information systems. This causes that the services offered to the citizens mainly rely on the availability of their information systems, and the integrity, privacy and confidentiality of the data administered by them. These are the three mainstays of actual definition of information security. A security incident altering these pillars can be critical to users and service providers, because it can provoke a drop in confidence. The number of incidents reported to the CERT has grown exponentially since the birth of modern networks. From around 250 in 1990 to 2500 in 1995, and to 22000 in 2000, to more than 50000 in 2001 and 80000 in 2002. Regarding software vulnerabilities, in 1995 were reported 171 vulnerabilities, in 2000 1090 were reported, and up to 7300 vulnerabilities reported in 2007.1

High assurance systems are computer systems where compelling evidence is required that the system delivers its services in a manner that satisfies certain critical properties such as safety, security and privacy [35]. Security requirements on an ideal development process for such systems are derived more or less trivially from some well-recognized evidence related to this definition. The impact of a security incident affects not only information theft or lack of services. Vulnerability correction poses an economic problem to the developers, because the latter the defects are found, the more expensive and critical they are likely to be. These corrections because of the already found errors are not the only way new errors are introduced. Systems where confidential and private user information is handled also need a high grade of security to avoid legal problems. Likewise, the main goals in security and privacy must be carefully identified using a set of security requirements, specific for the developed platform [29]. This way, the first step towards identifying security requirements is to determine a topology of threats that can affect systems.

<table>
<thead>
<tr>
<th>Year</th>
<th>Total Vulnerabilities Catalogued</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1-Q3 2008</td>
<td>6,058</td>
</tr>
<tr>
<td>2007</td>
<td>7,236</td>
</tr>
<tr>
<td>2006</td>
<td>8,064</td>
</tr>
<tr>
<td>2005</td>
<td>5,990</td>
</tr>
<tr>
<td>2004</td>
<td>3,780</td>
</tr>
<tr>
<td>2003</td>
<td>3,784</td>
</tr>
<tr>
<td>2002</td>
<td>4,129</td>
</tr>
<tr>
<td>2001</td>
<td>2,437</td>
</tr>
<tr>
<td>2000</td>
<td>1,029</td>
</tr>
<tr>
<td>1999</td>
<td>417</td>
</tr>
<tr>
<td>1998</td>
<td>262</td>
</tr>
<tr>
<td>1997</td>
<td>311</td>
</tr>
<tr>
<td>1996</td>
<td>345</td>
</tr>
<tr>
<td>1995</td>
<td>171</td>
</tr>
<tr>
<td>Total</td>
<td>44,074</td>
</tr>
</tbody>
</table>

Table 5.2.: Reported vulnerabilities

1www.cert.org/stats/ Accessed in October 2010
Security requirements
6. Foundations

Since the beginning of The Internet, security concept has had an increasing interest. Software and applications are getting more heterogeneous, availability has become very critical and code-inherent vulnerabilities increase. Attackers are using more sophisticated techniques [26], making possible theft and fraud in the digital World. The consequences are devastating to the point of breaking severely citizen’s digital security. As an example, there are reports of Denial-of-Service attacks in medical environments that caused a dramatic interruption of these services working conditions. Surprisingly, the main reason for application code vulnerabilities is the low quality of the development process [19]. Part of this work package tries to reduce the risk of a security incident, covering the main security aspects of project development. In first place, it presents the results of a study in the threats topology, because it’s necessary to know compromising agent behaviour in the FastFix platform to be able to execute protecting actions against them. In second place, there is the compilation of security requisites that were set out for each one of the modules or elements taking part in the project architecture.

6.1. Definitions

To understand the whole of the presented measures we need to define a series of concepts [49]:

- **Integrity**: The property that data has not been altered or destroyed from its intended form or content in an unintentional or an unauthorized manner.
- **Confidentiality**: The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.
- **Availability**: The extent to which information is operational, accessible, functional and usable upon demand by an authorized entity (e.g., a system or user).
- **Asset**: Resource of the organization which has a physical, logical or material value. It is needed for the correct running of the organization, and the achievement of the proposed objectives.
- **Event**: Occurrence of a group of circumstances.
- **Attack**: Group of actions whose objective is to compromise any of the security aspects of a resource (these are integrity, confidentiality or availability).
- **Threat**: Undesirable event that can potentially cause damage on the assets.
- **Incident**: Any adverse event that threatens the confidentiality, integrity or availability of information resources.
- **Potential safeguard**: Procedure or device, either physical or logical, which reduces the risk.
- **Risk**: Probability that a consequence is produced over an asset. It’s divided into inherent risk, assessed before considering introduced safeguards, and residual risk, known to be the risk still existing alter the introduction of the safeguards.
Malicious Code: Malicious code refers to code that is written intentionally to carry out annoying, harmful actions or use up the resources of a target host. They sometime masquerade as useful software or are embedded into useful programs, so that users are induced into activating them. Types of malicious code include Trojan horses and viruses.

Information Security Architecture: A framework designed to ensure information security. Principles are defined and integrated into business and IT processes in a consistent manner.

6.2. Security requirements methodology

The security requirements definition process of the FastFix project has required basically 3 focused areas of work [55]. The first thing to consider is on the one hand, the necessity to define a threat catalogue using a topology of the common pattern attacks against the FastFix assets, thus we have obtained threats families 1. On the other hand, we have determined all the architecture components, identifying a taxonomy that includes all the FastFix assets, this categorization has grouped all server internal modules, the agents and sensors deployed and all the communications needed among their elements [1]. Last but no least, it is necessary to obtain the misuses cases based on the uses cases, enabling the possibility of determine the specific security requirements. The next picture illustrates an example of the applied methodology.

As a result and using the information above is possible to generate global security requirements based mainly on the taxonomy of threats and the catalog of the platform systems. The definition of specific security requirements must be defined over the life of the project as these are based on use cases, part of the iterative cycle of software development. Thus, we have determined the set of global security requirements taking as input the list of threats and the taxonomy of the platform.

7. Threat topologies

Here we discuss the threats affecting the platform development according to a topiology [2] that tries to cover all possible risk scenarios. It’s shown below together with the most significant threats to the FastFix environment:

7.1. Authentication

Authentication can be defined as the process by which the system verifies the user’s identity, trying to determine that the user is who he says he is, generally through credentials such as user and password. But in FastFix, we can identify the authentication process as an authorizer to some private areas of the platform, such as management interfaces [2].

- **Identity impersonation**
  - **Description**: This threat arises when a user, either internal or external, logs into the system with other user’s credentials (user and password). This can be a potential attack if the identity impersonation has evil intentions or a traceability and responsibility problem if the impersonator has accessed the system in collaboration with the legitimate user.
  - **Potential safeguard**: It’s necessary to make the users aware of best practices in password selection and to introduce expiration and syntax control techniques that prevent the use of easy or predictable passwords. It’s also necessary to introduce full and periodical checks of the accesses to production systems, controlling that they are logical and correct in duration and timeline.

- **User credentials obtaining**
  - **Description**: The arousal of this threat, which consists on an illegitimate obtaining of the user’s access credentials, is related to the use of sniffing tools, bad practices in credentials selection and management, and more sophisticated attacks as the man-in-the-middle ones.
  - **Potential safeguard**: Expiration and syntax controls must be introduced, so password choosing is improved and brute-force attacks avoided. This must be applied together with awareness campaigns about best practices in security, to improve effectiveness. The use of hub-like devices should be avoided, substituting them with switch-like devices, to mitigate sniffing over the network nodes.

- **Other user’s session theft and manipulation**
  - **Description**: User session theft can be considered similar to other user identity impersonation, because the system cannot distinguish between the attacker...
D7.1: Global performance, security and privacy requirements

and the legitimate user. With this kind of attacks it’s possible, as well as to obtain the legitimate user profile data at runtime, to modify it.

- Potential safeguard: The application must free from memory all session objects once the user session has been closed, as well as other related temporary data.


Authorization is the process that validates the user and allows him (or not) to access the application, giving him the privileges granted by the administrator [2]:

- Privilege escalation.
  
  - Description: this threat consists of obtaining illegally a higher privilege level than the one granted by the user’s profile. This aspect is applied to the FastFix platform environment, for internal users as well as other users or administrators.

  - Potential safeguard: To avoid this kind of actions, it’s necessary to control and limit the use of privileged users. Likewise, it’s important to apply best practices in password choosing and check periodically their accesses to the system. On the other hand, as most of systems and applications vulnerabilities are directed to privilege obtaining (to be able to execute specific and unique commands or as a start point for more elaborated attacks), it’s essential to keep all systems and applications fully updated, preventing the attacks over publicly disclosed vulnerabilities.

- Access to privileged actions.
  
  - Description: This threat is very similar to the previous one. It’s related to the sporadic use of some functionality or logical access which usage is limited to privileged users.

  - Potential safeguard: The safeguards explained before are also valid here.

- Unauthorized access to development and preproduction environments.
  
  - Description: Third party unauthorized access to environments other than production implies that people having this kind of accesses could introduce unauthorized modifications in the final product. They could also access data managed by or stored in the application (this is the main reason for the database isolation between production and preproduction environments), or even steal source code.

  - Potential safeguard: Development and preproduction environments must be correctly updated in security matters (patches, updates, open ports, etc), and its use must be strictly restricted to authorized personnel (system and databases administrators in preproduction environments, and developers or administrators in development environments). This way code integrity will be secured, and damage caused by malicious programs will be avoided.
7.3. Data input validation

In FastFix platform, exploitation of data input validation threats is closely linked with the modules that realize that kind of functions. These modules are located in the generic components of the system architecture, user interface and administration part [2]:

- Introduction of invalid input parameters.
  - Description: This threat is related to the modification of application’s input parameters. Through this attack, it could be possible to obtain information about other models and access non-public or restricted areas of the application, either in client or server side.
  - Potential safeguard: Validation module must deal with parameters introduced in input boxes and handle them coherently with the rest of the information introduced. This handling can be as simple as discarding that parameters, in most cases.

- SQL Injection attacks.
  - Description: This threat resides in exploiting the user interaction provided by web applications (for example in data entry forms), to execute manipulated SQL queries. Being incorrectly filtered by the system, these queries can have undesirable effects, including revelation of system information, database structure, or even allowing modification or removal of stored data. This kind of threats could affect FastFix platform in the parts offering web services.
  - Potential safeguard: All of the inputs in forms and URL parameters being used must be validated and correctly handled before their processing by other modules of the system, or their storage in the database.

- Cross-Site Scripting attacks.
  - Description: A Cross-Site Scripting (or XSS) attack consists of the inclusion of some code, often JavaScript or VB Script, in web forms or URL parameters, for it to be stored in or reflected by the web server, and eventually executed in any user browser viewing the set in information. This kind of attacks is usual in forums, blogs or digital media, where there exist a high level of user interaction. As in the previous threat, this menace could affect the part of FastFix platform using web services.
  - Potential safeguard: Refer to the previous threat.

- Buffer Overflow attacks.
  - Description: A buffer overflow attack is based on the overrun of a buffer’s space, with the aim of overwriting other memory zones with malicious code, which can lead to unauthorized code execution in the system.
  - Potential safeguard: This kind of attacks is directed mainly to desktop applications, but here can produce denial of service, connections slow down and bad performance. It is important the usage of alternative functions limiting
the amount of memory being copied between variables, instead of using functions that don’t control this aspect (for example, by using `strncat()` instead of `strcat()` in C).

- Malformed input denial of service (DoS) attacks.
  - Description: This threat joins together the previous attacks, trying to make the system work unexpectedly when processing an unexpected, and thus not considered in the code, input parameter. These attacks can cause a high slowdown in the system, faulty operation or even a system hang.
  - Potential safeguard: Refer to previously pointed out safeguards related to input data validation.

### 7.4. Configuration.

In this point we are about to discuss about general configuration threats [2] that can allow attackers to access and manipulate it, or make it work unexpectedly or incorrectly.

- Unauthorized access to application configuration data.
  - Description: This threat uses different attacks, or exploits faulty access permissions to obtain illegitimate access to the configuration values, containing IP directions, access credentials or database connection parameters. Among others, the existence of directories allowing listing, extensive error reports or information excess, are very common.
  - Potential safeguard: In order to fight against this kind of threats, it’s desirable the execution of periodical security audits in the production environment, and in minimum impact level. Additionally, application’s components execution permissions must be revised, as well as visibility restrictions in web directories or adequate error handling.

- Faulty operation because of lack of permissions.
  - Description: Here the threat is produced by a bad setup or installation of the application, in which read or write permissions of files, folders or databases aren’t well defined. This causes the system to have an erratic behaviour, if any.
  - Potential safeguard: Application permissions in production environments must be identical to those in preproduction environments, either at a server (and web server) side or at any other involved component side, assuring the right behaviour of the application.

- Vulnerability by excess of application permissions.
  - Description: Opposed to the previous threat, the excess of permissions in application installation or setup raises the impact of a successful attack over the system, allowing the attacker to obtain higher privileges and visibility over the application than the strictly needed for the app to work properly. This threat raises the probability of other systems affectation if a vulnerability is exploited.
D7.1: Global performance, security and privacy requirements

- Potential safeguard: Periodic security audits must be run in the production environment, to make sure that it’s not possible to gain access to system or environment resources that shouldn’t be accessible through the application.

- Bad behaviour or disinformation because of lack of time synchronization between systems in the application environment.

  - Description: In environments which systems communicate among others and the time factor is relevant (data exchange, cross validation, etc.), it’s mandatory that all of the systems are synchronized by means of time synchronization systems like NTP. In addition, this aspect takes special significance when investigating security incidents, where knowing the temporal order of the events and transactions that took place is mandatory.

  - Potential safeguard: NTP time servers must be deployed, and NTP clients must be installed in every system in the environment. In this case, this measure has already been adopted.

- Uncontrolled software or operative system update.

  - Description: Updates application in any software component (such as software, operative system, or database version, for example) involved in the application operation and motivated by security problems or new functionality, must be thoroughly analyzed and previously tested in preproduction environments. This avoids that the solving of irrelevant problems for the actual environment ends up in real problems.

  - Potential safeguard: Patch application, version changes or modifications of any kind in application’s systems must undergo a previous stage of implications analysis, and must be scheduled in a minimal impact schedule. If the change is significant, the person responsible for the application must authorize the change. In FastFix platform, all the patches or updates must be applied in a preproduction environment, undergo a full set of performance and functionality tests and work correctly during a reasonable time lapse before being applied to a production environment.

7.5. Sensitive data.

Sensitive data appearance, regarding application, system or environment information, must be limited, so users or potential attackers cannot obtain it. Also, public information contained at any part of the application must be managed and approved in a procedure way.

- Passwords, IP directions or access paths embedded in the source code.

  - Description: Under some circumstances, due to commodity or speed, developers embed configuration data in the code, such as database IP directions, access paths or even passwords. If that is the situation, and strict controls are not implemented in the moving into production process, these problems are also moved to the production environment, exposing the application and
hardening the modification of these parameters in the future. This threat is especially critical if that information is embedded in JavaScript code, if the app is a web one, or in the source code of interpreted languages.

- Potential safeguard: General configuration files must be used, without public paths and access rights, avoiding the embedding of any value in the code. Source code must be audited periodically looking for this kind of information.

- Eavesdrop on the sending of sensible data across the network (Logical sniffing).

  - Description: This threat represents the ability of a system connected to the electronic transmission systems of FastFix platform to listen and copy the information transmitted by the system.

  - Potential safeguard: In order to mitigate this kind of risks, switched devices must be used, instead of hub-like devices, making the passive sniffing difficult. Thus, the communication between systems must be ciphered with SSL or TLS (https in web services and ciphering libraries otherwise).

- Unauthorized view or manipulation of sensitive data.

  - Description: This threat consists of the manipulation, publication or view of sensible data by unauthorized personnel. In FastFix environment, this risk is tightly related with enterprise information, and the fact that this data could be accessed by external third parties, either through an interface (client or server side) error, system vulnerability or data loss.

  - Potential safeguard: Backend systems must reside in a network range isolated from the frontend, and its access from the frontend or DMZ must be limited to the strictly necessary ports to obtain a correct operation. This way Internet accesses to the backend are avoided, and ports that could lead to any type of attacks remain invisible to the Internet. The isolation of presentation and database layers in different environments with controlled traffic between them is essential in any proposed security architecture; if this isolation cannot be achieved, compensatory controls are highly recommended, and so the total security is reinforced. Examples of compensatory controls are: host intrusion detection systems, firewalls, exit traffic control, etc.

- Source code theft.

  - Description: Event though the development process is carried out mainly by consortium members, the developers own a development and preproduction environment where they can deploy the application. Its analysis by a malicious user could lead to ways of exploiting the application externally.

  - Potential safeguard: As it has been pointed out in previous thefts shown there, access to development and preproduction environments, having source code on them, must be allowed only to developers and authorized personnel. When the development is done by a third party, confidentiality and exclusivity agreements must be set up, to assure the code is neither advertised nor shared with external third parties that are not in collaboration with the FastFix project.
7.6. Exception handling.

The application must handle all errors and warnings in an appropriate way, avoiding the disclosure of too much information or sensitive information, and also avoiding the suffering of unexpected effects [2].

- **Application stop.**
  - **Description:** An incorrect error handling in the application can cause a faulty operation of the system, causing unexpected behaviour, system slow down and, eventually, system shutdown.
  - **Potential safeguard:** All of the application raised errors must be correctly handled, masking the error to the user or generating a friendly error page. These errors must generate internal reports that must be revised and managed by the people responsible for the application.

- **Relevant data obtaining from exceptions.**
  - **Description:** An incorrect error handling can show the attacker valuable information about the system, that the attacker can use to perform other attacks. This information includes versions of all the components of the system (application servers, databases, etc), IP addressing, full access paths, etc.
  - **Potential safeguard:** Whenever it is possible, error screens and their associated information must be previously defined. This behaviour is configurable in the application or web server.

7.7. Audit and logging.

In order to detect security and operational problems, every component must have a logging system, allowing it to establish the traceability of the detected errors or incidents, and be able to carry out a reliable analysis of its origin [2].

- **Masking and removing traces.**
  - **Description:** The base of this threat resides in the probability that application logs and traces can be removed, either by chance or on purpose, hiding possible attacks, application errors or operational revisions.
  - **Potential safeguard:** System logs (databases, application servers, access logs) must be accessed only by privileged users. Whenever it is possible, an external and centralised logging system should exist. This system, if existing, must receive all the relevant information to allow its revision and eventually saved in a centralized way.

- **Lack of logging records.**
  - **Description:** The lack of activity records of the components of the system can pose a great problem in many ways: the review of operational errors, potential attacks or fraudulent use trials, overload detection, application sizing, or analysis of use problems, etc.
D7.1: Global performance, security and privacy requirements

- Potential safeguard: The systems must be configured to register the relevant actions of the system, to a level that doesn’t interfere in the system performance and the volume of generated data is easy to use (debug levels in database, for example). A centralised logging system is recommended, as in the previous safeguard, allowing the compilation and eventual analysis of logs.

7.8. Application deployment control.

The application building process through the development stages (development, preproduction or mirror, and production) must be subject to a change management process. This process must determine formal authorizations and requirements [2], such as responsible personnel, reverting procedures and time schedule.

- Error propagation and persistence because of lack of validation in the moving to preproduction stage.
  - Description: The lack of a formal validation protocol for the application to move from development stage to preproduction, poses a potential risk of errors in the system, inherited from the development stage. This can also lead to the release of some unapproved functionality.
  - Potential safeguard: A change in the management process must be developed and introduced. This process must consider the formal approbation of the development stages’ changes from development to preproduction, and also determine the duties and responsibilities of the involved actors. Direct modifications in the preproduction system must be forbidden, and periodical security audits and modified source code revisions must be performed.

- Error propagation and persistence because of lack of validation in the moving to production stage.
  - Description: The lack of a formal validation protocol for the application to move from preproduction stage to production, poses a potential risk of errors in the final system, the one the user will access. This can also lead to the release of some unapproved functionality.
  - Potential safeguard: As it is said in the previous safeguard, a change in the management process must be developed and introduced. This process must consider the formal approbation of the development stages’ changes from preproduction to production, and also determine the duties and responsibilities of the involved actors, the schedule, estimated migration time, previous and new requirements, and reverting procedures.

- Error appearance because of application version or configuration differences in the development, preproduction and production environments.
  - Description: Version or configuration differences between different components of the system used in preproduction and production environments (as database engines, or operating system or application servers versions, for example) can lead the application to exhibit a different behaviour to those of the preproduction environment, either aesthetical or functional.
• Potential safeguard: Components in both environments must have the same versions, configurations and updates, and any modification made in the pre-production environment must be included into the production environment, after a functional verification has been held.

• Lack of reverting procedures when moving to the production environment.
  • Description: The lack of a reverting procedure can risk the application normal operation, leaving it on an inconsistent state at application level and database level.
  • Potential safeguard: The moving to production stage must have a reverting mechanism just in case there is any problem with the application deployment. Every modification to the production system that requires a service stop or a major application modification must have this mechanism, approved by a responsible person.

• Existence of development tools in production environments.
  • Description: The existence of development tools in production environments, such as debuggers, compilers or other development tools can provoke its use to compile exploits, trojans or viruses, besides favouring the application modification by the developers.
  • Potential safeguard: Final environments must have the minimal set of tools needed for the correct operation of the application, avoiding the indicated threats and potential library or compatibility problems.

7.9. Version control.

The source code of an application, its non buildable components, the databases and any other element of the application must be correctly saved in backups, assuring its availability if necessary [2].

• Unavailability of backups.
  • Description: The unavailability of backups, caused by theft, loss or bad conservation can risk the application continuity in the worst cases, or make the recovery of lost or erased information more difficult.
  • Potential safeguard: System backups must be periodically scheduled and saved in a restricted access area, controlling the usage limit date of the CD, DVD or tapes used.

• Impossibility to recover previous versions of the system.
  • Description: The lack of version control tools makes the recovery of previous versions of the developments notably difficult, avoiding the comparison of changes, and the return to consistent application states if it is needed.
  • Potential safeguard: A version control application must be introduced, that makes the source code management easier, and allows for the efficient management of previous versions of the application.
8. FastFix Global Security Requirements

Next, we will introduce the global security requirements that FastFix platform must apply, with the purpose of ensuring an appropriate level of privacy, availability and integrity. Actions are focused towards four different lines: ensuring server environment security, sensor level requirements in the client, maintenance of confidentiality and integrity of data while information flows and finally, aspects concerning security on Java environments.

8.1. Server security requirements

8.1.1. Global requirements

Efforts have been focused on identifying the security items that can affect the FastFix project and its users separately from communication, server or client specifications, where the items are based on identified security threats.

8.1.2. Administration interfaces

One of the prominent concepts concerning the security of the FastFix platform is the covering of all the security aspects of the administration interfaces of its architecture. These interfaces can be delivered through web-based environments. If the proper measures are not adopted, this kind of systems can present multiple security deficiencies. Hence, security requirements have been defined with five criteria: authentication, session management, authoring, input data validation and configuration management [24].

- Authentication
### D7.1: Global performance, security and privacy requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All systems supporting server environment</td>
<td>Host environment (where the platform core will be deployed) must have all OS security patches updated(^1).</td>
</tr>
<tr>
<td>2</td>
<td>All systems supporting server environment</td>
<td>Applications to be used must be updated to its latest version. If it is not possible, installed versions must be updated with the latest security patches [36].</td>
</tr>
<tr>
<td>3</td>
<td>All the platform assets</td>
<td>Server must use services to keep time and other elements updated (e.g. NTP). [60]</td>
</tr>
<tr>
<td>4</td>
<td>Test and production assets</td>
<td>Maintenance of development and pre-production environments, separately from production environment, with the same security controls [1].</td>
</tr>
<tr>
<td>5</td>
<td>Network infrastructure</td>
<td>Link layer communications environment must be implemented with a switch, instead of a hub [52].</td>
</tr>
<tr>
<td>6</td>
<td>Administration system</td>
<td>A password expiration policy (3 months) for administrator access must be applied [1].</td>
</tr>
<tr>
<td>7</td>
<td>All systems supporting server environment</td>
<td>Server platform must offer only strictly necessary services for FastFix use, avoiding problems derived from unnecessary services [22].</td>
</tr>
<tr>
<td>8</td>
<td>Network infrastructure</td>
<td>Services offered by the server must be published in a DMZ network segment, allowing filtering through firewall [1].</td>
</tr>
</tbody>
</table>

**Table 8.1.: Global security requirements**

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>All server administration systems</td>
<td>The server administration interface must have an authentication system (through user-password) [2].</td>
</tr>
<tr>
<td>10</td>
<td>All server administration systems</td>
<td>Authentication system must not allow listing users, in other words, in case that access credentials are incorrectly entered, it will show a general message: “Application access error” [2].</td>
</tr>
<tr>
<td>11</td>
<td>All server administration systems</td>
<td>Placing back-up copies or temporary files inside the application folder tree must be avoided [2].</td>
</tr>
<tr>
<td>12</td>
<td>All server administration systems</td>
<td>The authentication system must use passwords with 8 alphanumeric characters minimum, including 1 symbol at least [2].</td>
</tr>
<tr>
<td>13</td>
<td>All server administration systems</td>
<td>Authentication system must not include trivial management accounts, like admin, root or test [2].</td>
</tr>
<tr>
<td>14</td>
<td>All server administration systems</td>
<td>If a password recovery system is available, it must send an email to the alternative account, introduced in the registering process of the user, delivering a temporary link that allows to re-establish the password. The password must never be directly sent through email [2].</td>
</tr>
<tr>
<td>15</td>
<td>All server administration systems</td>
<td>Critical operations as password reset, email or SMS notification or create new user must be protected through a Captcha system [2].</td>
</tr>
</tbody>
</table>

**Table 8.2.: Authentication requirements**
D7.1: Global performance, security and privacy requirements

- Session management

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>All server administration systems</td>
<td>A close session link must be permanently working and visible [2].</td>
</tr>
<tr>
<td>17</td>
<td>All server administration systems</td>
<td>Session token must be always sent through POST method or using a cookie, never through GET parameters [2].</td>
</tr>
<tr>
<td>18</td>
<td>All server administration systems</td>
<td>Session token generation must be robust (using a 256 bits AES algorithm with a charset of 50 elements at least [2]).</td>
</tr>
<tr>
<td>19</td>
<td>All server administration systems</td>
<td>Critical operations must avoid CSRF attacks, in other words, a temporary token must be generated as an access link to perform the operation. Such token will be sent through POST parameters and it must expire after use [2].</td>
</tr>
<tr>
<td>20</td>
<td>All server administration systems</td>
<td>Session token must be checked for each client request [2].</td>
</tr>
<tr>
<td>21</td>
<td>All server administration systems</td>
<td>Session token’s length must be checked by the server, preventing from buffer overflow while being handled [2].</td>
</tr>
<tr>
<td>22</td>
<td>All server administration systems</td>
<td>If the session is handled through a cookie, it must expire with the session [2].</td>
</tr>
<tr>
<td>23</td>
<td>All server administration systems</td>
<td>Session fixation attacks must be avoided, in other words, session token must change after server’s authentication [2].</td>
</tr>
<tr>
<td>24</td>
<td>All server administration systems</td>
<td>Session must expire after an abrupt closure of the administration environment [2].</td>
</tr>
</tbody>
</table>

Table 8.3.: Session management requirements

- Authorization
### D7.1: Global performance, security and privacy requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>All server administration systems</td>
<td>Application server configuration must not allow operations performed out of webroot’s path [2].</td>
</tr>
<tr>
<td>26</td>
<td>All server administration systems</td>
<td>Once the user has closed the session, he must not be able to access to the application resources [2].</td>
</tr>
<tr>
<td>27</td>
<td>All server administration systems</td>
<td>The user must not be able to modify application profiles or grants, and they must not be sent through parameters in cookies or GET or POST methods [2].</td>
</tr>
</tbody>
</table>

Table 8.4.: Authorization requirements

- Input data validation

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>All server administration systems</td>
<td>Each parameter value (GET, POST, cookie) that can be modified by the user must not be returned through HTML format without filtering the information to be sent, avoiding reflected XSS attacks [2].</td>
</tr>
<tr>
<td>29</td>
<td>All server administration systems</td>
<td>Each parameter value (GET, POST, cookie) that can be modified by the user must not be persistently included in HTML data, without filtering the information to be sent, avoiding persistent XSS attacks [2].</td>
</tr>
<tr>
<td>30</td>
<td>All server administration systems</td>
<td>Each parameter value (GET, POST, cookie) that can be modified by the user, which could be part of a SQL query must be previously filtered [2].</td>
</tr>
<tr>
<td>31</td>
<td>All server administration systems</td>
<td>Each parameter value (GET, POST, cookie) that can be part of a repetitive sentence must be previously checked [2].</td>
</tr>
<tr>
<td>32</td>
<td>All server administration systems</td>
<td>Each parameter value (GET, POST, cookie) that can be part of a disk write sentence must be controlled in order to prevent possible DoS attacks, filling the whole disk storage system [2].</td>
</tr>
<tr>
<td>33</td>
<td>All server administration systems</td>
<td>An appropriate resource management must be carried out, once these resources are not in use [2].</td>
</tr>
</tbody>
</table>

Table 8.5.: Input data validation requirements

- Configuration management
Table 8.6.: Configuration management requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>Database servers</td>
<td>Database component must handle exceptions correctly, preventing from revealing internal configuration information [2].</td>
</tr>
<tr>
<td>35</td>
<td>All server administration systems</td>
<td>File extensions must be hidden, avoiding the retrieval of the language or technology currently used.</td>
</tr>
</tbody>
</table>

8.1.3. Event correlation

One of the components of special importance in the FastFix general architecture is Event Correlation, which is in charge of receiving the whole set of notifications coming from each one of the deployed agents. The application of security requirements and restrictions in the Event Correlation design will provide it with an appropriate strength to cope with possible security threats, reducing the risk of falling into a security incident.

Table 8.7.: Event correlation security requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>Event correlator and interacting modules</td>
<td>It must ensure that only authorized parties are allowed to be event producers or event consumers. For such purpose these agents must be authenticated [17].</td>
</tr>
<tr>
<td>37</td>
<td>Event correlator</td>
<td>It must ensure that incoming events are filtered so that producers cannot introduce malicious data to produce a crash, denial of service or code injection [17].</td>
</tr>
<tr>
<td>38</td>
<td>Event correlator</td>
<td>It must ensure that consumers only receive information to which they are entitled [17].</td>
</tr>
<tr>
<td>39</td>
<td>Event correlator</td>
<td>It must ensure that unauthorized entities cannot add new Event Processing Agents to the system or make modifications to the Event Processing Network [17].</td>
</tr>
<tr>
<td>40</td>
<td>Event correlator</td>
<td>It must keep auditable logs of received and processed events or other activities performed by the system [17].</td>
</tr>
<tr>
<td>41</td>
<td>Event correlator</td>
<td>It must ensure that correlation system databases do not offer public services or interfaces that can be accessed by third-parties. In that sense, it must offer authentication capabilities and a physical or logical interface only accessible by the correlation engine and entitled systems [17].</td>
</tr>
</tbody>
</table>

8.1.4. Java security requirements

One of the already taken decisions in the project is that FastFix platform development is going to use in some way the Java programming language. Fault Replication (WP5) and Self Healing (WP6) components are not an exception. Hence, we proceed not to identify a set of security requirements, that must be applied in subsequent stages of the project.
8.1. Java code security requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Java code</td>
<td>Java permissions for accessing to the file system must use the FilePermissions directive, avoiding the use of wildcards * or -, and granting access with minimum privileges (read/write) only to the required information [58].</td>
</tr>
<tr>
<td>59</td>
<td>Java code</td>
<td>Authentication and authorization of users must be handled in a suitable way, using SecurityPermission and AuthPermission classes, always with a minimum privileges policy [58].</td>
</tr>
<tr>
<td>60</td>
<td>Java code</td>
<td>With regards to the access control, the mediator for access control operations is the JavaSecurityManager, that has to be installed in the Java runtime. Without the SecurityManager it isn’t possible to perform access control checks. Hence, the use of this service must be configured in the JVM properties(^2).</td>
</tr>
<tr>
<td>61</td>
<td>Java code</td>
<td>The SecurityManager works with predefined permissions and policies or uses the default permissions respectively default policy. Therefore, a file access policy must be established (file java.policy) and it must be based on default DENY command, and granting permissions to classes as it is required on time(^3).</td>
</tr>
</tbody>
</table>

Table 8.8.: Java security requirements

8.2. Client and sensors security requirements

Client and sensors security requirements are described in Table 8.9

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Application sensors</td>
<td>A previous obfuscation of data must be performed in the Fault Replication component, with the aim of ensuring the privacy of the user information(^1).</td>
</tr>
<tr>
<td>43</td>
<td>Application sensors and FastFix server</td>
<td>With a view to ensure the integrity of the information sent by the sensors to the server, the SHA-512 hash of the sensor itself must be sent along with the information, providing a system to ensure that the code have not been manipulated.</td>
</tr>
<tr>
<td>44</td>
<td>Application sensors and FastFix server</td>
<td>Each log file generated on the client side must not contain sensitive data like passwords, IP addressing or any personal information data [7].</td>
</tr>
</tbody>
</table>

Table 8.9.: Client and sensors security requirements

8.3. Communications

Communication between FastFix components and between server and sensors deployed in the client must ensure that security requirements are upheld.
8.3.1. External communications

External communications, between sensors or agents deployed in the application and the FastFix server, must grant a suitable security level, disabling any attacker from interfering or influencing on the availability, confidentiality or integrity of the interchanged data.

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>Application sensors and FastFix server</td>
<td>It must use cyphering communication protocols, like SSL or TLS (updated versions) [54].</td>
</tr>
<tr>
<td>46</td>
<td>Application sensors and FastFix server</td>
<td>It must use AES or CP4 as cyphering algorithms [54].</td>
</tr>
<tr>
<td>47</td>
<td>Application sensors and FastFix server</td>
<td>It must use SHA512 or SHA265 as hash algorithms [54].</td>
</tr>
<tr>
<td>48</td>
<td>Application sensors and FastFix server</td>
<td>Communication between FastFix components must provide an authentication system to use its offered services. In case of using Web Services, these must only grant access to its functionality under previous authentication [1].</td>
</tr>
</tbody>
</table>

Table 8.10.: External communications security requirements

8.3.2. Internal OSGi communications

Once the external communication requirements are set, internal communications between FastFix server modules becomes necessary. Data interchange between modules will be performed by means of the OSGi services architecture. OSGi security is mainly based on the Java solution. The OSGi security layer must run in parallel to the functionality layers. With the currently newest version of OSGi, Release 4, the way of enabling security changed. In the earlier versions of the framework, a class called PermissionAdmin was responsible for security operations. This class is still available in OSGi Release 4, but only to provide backward compatibility. Therefore, we must use the new class to handle security issues, the ConditionalPermissionAdmin.

With AdminPermission granted, bundles can perform privileged administrator functions or get sensitive information about bundles. OSGi security requirements are focused on achieving two targets: ensuring a correct use of permissions and establishing suitable access conditions.

8.3.3. OSGi permissions

OSGi permissions are described in Table 8.11
### D7.1: Global performance, security and privacy requirements

#### Figure 8.1.: OSGi architecture

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>Internal Modules</td>
<td>One of the four OSGi special permissions, AdminPermission, allows performing privileged administration functions. Hence, this permission must be properly configured, restricting execute, lifecycle and context actions to bundles. If not, malicious bundles might stop or uninstall all bundles, and even worse, updating some bundles and replace the existing ones with his own probably malicious bundles. Only the minimum necessary permissions will be granted [58].</td>
</tr>
<tr>
<td>50</td>
<td>Internal Modules</td>
<td>PackagePermission has two actions, export and import, which allow to export or import a package from another bundle. Only the required services will be exported/imported [58].</td>
</tr>
<tr>
<td>51</td>
<td>Internal Modules</td>
<td>BundlePermission is similar to PackagePermission, with the require and provide actions, allowing to a bundle to get access to a package from another bundle, but offering special functions to expose only a fragment of the services of a bundle. Only the minimum required services access will be granted [58].</td>
</tr>
<tr>
<td>52</td>
<td>Internal Modules</td>
<td>ServicePermission allows actions to get or register another bundle. This permission isn’t needed by application designers, but while building a whole OSGi implementation, this is a must. It must only be offered in this second case (the minimum required) [58].</td>
</tr>
</tbody>
</table>

Table 8.11.: OSGi permissions requirements
8.3.4. OSGi conditions

OSGi conditions are described in Table 8.12.

<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>Internal Modules</td>
<td>With the purpose of ensuring the authenticity of the used bundles, the BundleLocationCondition and BundleSignerCondition directives must be used in the OSGi environment [58].</td>
</tr>
<tr>
<td>54</td>
<td>Internal Modules</td>
<td>As its name says, the BundleLocationCondition checks if the bundle comes from a specified location. The bundles locations must be included, denying access to any bundled considered as not trustworthy [58].</td>
</tr>
<tr>
<td>55</td>
<td>Internal Modules</td>
<td>Communication between external bundles, if necessary, must be performed through HTTPS protocol [58].</td>
</tr>
<tr>
<td>56</td>
<td>Internal Modules</td>
<td>The use of general wildcards must be avoided, like * in the BundleLocationCondition definition [58].</td>
</tr>
<tr>
<td>57</td>
<td>Internal Modules</td>
<td>All bundles must be able to prove its authenticity, by means of security signatures of BundleSignerCondition [58].</td>
</tr>
</tbody>
</table>

Table 8.12.: OSGi conditions requirements

8.4. Summary

Covering all the security aspects of a technological environment must start with the identification of potential security threats, able to produce security incidents with a negative impact affecting the development and exploitation of the FastFix project. The security part of the present document gathers all the actions to be taken into account, as a result of the carried out investigation of Work Package 7 (Task 7.3 Security Requirements), in the first 6 months of the project.

This way, 31 security threats have been identified and classified in 9 families, associated to its taxonomy, as a result of the analysis of security methodologies and other public sources of attack patterns, internationally well-known. On the basis of the previous information and after the study of possible measures that can mitigate the inherent risk in its materialization, we have proceeded to determine the security requirements focusing in three environments: server, client and communications.

Only 8 security requirements have been identified as not applicable to the previous categorization. These requirements refer to specifications affecting the whole set of systems.

Concerning the FastFix server environment, 37 requests have been determined focusing on three categories: administration interfaces, event correlation and programming in Java environment. This will allow to acquire an adequate level of security, enlarging the robustness against possible attacks.

Communications among the different modules of the platform must be covered in this security analysis, for both internal and external communications. Security must be ensured applying the 13 security requests presented in the corresponding section.
Last but not least, we have identified some security specifications that will be applied to the deployed sensors in the application client, permitting to obtain a high degree of integrity and privacy of the data property of the users.

To sum up, this WP7 document covers the security aspects that can be applied to the different FastFix elements, obtaining a suitable level of robustness against security threats.
Privacy requirements
9. Foundations

Establishing an optimal level of security in an organization, appropriate to the assets being protected and to the damage that could result from its loss or malfunction is a task that implies homogeneity in the various aspects of security: logical, physical and organizational security.

In addition, the need for adequate levels of security within the organization, sometimes is not a matter of choice, but an obligation under the regulatory framework. Moreover, sometimes a security incident involves not only undermining the institution itself and its image, but also a possible financial penalty associated.

The security framework of an organization should ensure the data privacy and proper use of treated personal data. Not only because it is regulated, but also because of the existing level of social sensitivity. Any new project or service that is being designed must include among its requirements compliance with the legislation on personal data protection.

Fastfix is not an exception.

9.1. Definitions

At first, some definitions that will be needed later on [15]:

- 'personal data': any information relating to an identified or identifiable natural person ('data subject'); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.

- 'processing of personal data' ('processing'): any operation or set of operations which is performed upon personal data, whether or not by automatic means, such as collection, recording, organization, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, blocking, erasure or destruction.

- 'personal data filing system' ('filing system'): any structured set of personal data which are accessible according to specific criteria, whether centralized, decentralized or dispersed on a functional or geographical basis.

- 'controller': the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data; where the purposes and means of processing are determined by national or Community laws or regulations, the controller or the specific criteria for his nomination may be designated by national or Community law.
D7.1: Global performance, security and privacy requirements

- 'processor': a natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller.

- 'third party': any natural or legal person, public authority, agency or any other body other than the data subject, the controller, the processor and the persons who, under the direct authority of the controller or the processor, are authorized to process the data.

- 'recipient': a natural or legal person, public authority, agency or any other body to whom data are disclosed, whether a third party or not; however, authorities which may receive data in the framework of a particular inquiry shall not be regarded as recipients.

- 'the data subject’s consent': any freely given specific and informed indication of his wishes by which the data subject signifies his agreement to personal data relating to him being processed.

9.2. Context

The Directive 95/46/CE [15] of the European Parliament and of the Council, of 24 October 1995, on the protection of the individuals regarding the processing of personal data and on the free traffic of this information constitutes the text of reference, to European scope, as for protection of personal data. Its publication constituted the creation of a regulatory framework with the purpose of establishing a balance between a high level protection of the private life of individuals and the free movement of personal data inside the European Union, fixing strict limits for the personal data processing and requesting the creation, in every Member State, of a national independent organism in charge of the protection of the personal data privacy. The Directive shall apply to the information treated by automatic means and to the information processed in not automatic information systems, and with exception of the processing of personal data effected by a natural person in the exercise of exclusively particular or domestic activities and of the exercise of activities not in the scope of the Community law (such as the public safety, the defense or the national security), the Directive is applied to any processing of personal data, having as aim the protection of rights and freedom of individuals regarding the processing of their personal data, and establishing principles to determine the legality of the personal data processing.

These are some of the European Directive 95/46 CE principles:

- The quality of the information: personal data will be treated fairly and lawfully, and collected with certain, explicit and legitimate purposes. In addition, it will be adequate and, when necessary, kept up to date.

- The legitimization of the processing: the processing of personal data will only be able to be effected with the unambiguous data subject consent or if the processing is necessary for some of the following suppositions:
  - The performance of a contract between the data subject and the controller.
  - The performance of a juridical obligation between the controller and a third party.
• To protect the vital interest of the data subject.
• The processing is required on public interest grounds.
• For the purposes of the legitimate interest of the data processing controller, except where such interest is overridden by the interests for fundamental rights and freedoms of the data subject.
• Members States shall prohibit the processing of personal data revealing the racial or ethnic origin, the political opinions, the religious or philosophical beliefs and the trade-union membership, and the processing of data concerning health or sex life, unless the data subject has given express consent for the data processing.
• Data subjects must be informed of the controller identity, treatment purposes, recipients of the personal data, etc.
• The controller shall guarantee to every data subject the right of access, rectify, erase and block of his personal data.
• The right of access of the data. All data subject should have the right to obtain from the controller:
  • Confirmation of whether or not data relating to him are being processed and information at least as to the purposes of the processing, the categories of data concerned, and the recipients or categories of recipients to whom the data are disclosed.
  • The guarantee of the rectification, erasure or blocking of data the processing of which does not comply with the provisions of this Directive.
• The right to object the treatment: the data subject shall have the right to oppose, for legitimate reasons, to the data concerning him are being processed. The data subject must be informed before personal data are disclosed to third parties for purposes of commercial prospecting, and have the right to object to such communication.
• Confidentiality and security of processing: persons acting under the authority of the controller or processor, may process personal data only to those who have access, and only when charge by the controller. Moreover, the controller must implement appropriate measures to protect personal data against destruction, accidental loss, alteration, disclosure or unauthorized access.

The Directive intended to facilitate the development of national codes of practice that contribute to a successful implementation of Community and national rules.

The Directive also stated that each Member State would appoint one or more independent public authorities responsible for monitoring the application within its territory of the provisions adopted by Member States in their implementation.

Since 1995, the transposition of the Directive to the legislative framework of each Member States has resulted in the existence of national laws with considerable interpretive differences, mainly related to:

• Interpretation and application issues: in some countries not only data on individuals, but also legal persons are considered within the scope of the law. On the other hand
the extensive, yet ambiguous definition of personal data means that there is a lack of definition of what is and is not a personal data, an interpretation that also varies from one Member State to another. Finally, we will always find the eternal discussion about what kind of data can make a person identifiable through it.

- Perception of risk and consequences of compliance failure. The long period of time transposing the Directive into the laws of the Member States has resulted in a relaxation by the private and public organizations in compliance and inspection activity was still low and punitive national institutions responsible for ensuring compliance with legislation.

- Differences in implementation. Regulatory and policy developments of national laws vary in their degree of specificity, permissiveness and level of "hardness" of the security measures implemented on the basis of personal data processed, differences that result in both organizational (mandatory or voluntary maintenance of the security document) and legal (differences in deadlines and registration obligations) and technical (mandatory or voluntary maintenance of audit records).

As a sample of these evident interpretive differences, and of the consistent relaxation that the doubts and the differences as for scope and level of implantation of the security measures are provoking, the Workgroup of the 29th article, European organism with consultative and independent character for the personal data privacy created by virtue of the article 29 of the Directive 95/46/CE, in his document on "The Future of the Privacy" (WP 168) of December 2009, it was demonstrating already his opinion that the in force juridical frame had not managed to guarantee that the protection requirements of information were being translated into effective mechanisms that were contributing an authentic protection of the personal data.

In fact, the 29th Group published a document in July 2010 denouncing that if the personal data protection does not form a part of the shared values and the practices of the organizations and responsibilities are not assigned expressly, there are serious risks about the guarantee of an adequate treatment of personal data.

This opinion even proposes the modification of the Directive, introducing the principle of responsibility of the processors and the controllers of the personal data processing in applying the appropriate and effective security measures that guarantee the observance of the Directive principles and obligations.
10. A first approach to privacy in FastFix

Understanding "personal data processing" as any operation or set of operations performed upon personal data, whether or not by automated procedures, such as collection, recording, organization, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, blocking, erasure or destruction, and keeping in mind FastFIX's general exposition and the aims that the project chases, later the implications that some of the principles regulated by the Directive 95/46 CE are going to be analyzed.

Data quality  "Personal data will be processed fairly and lawfully, and collected for specified, legitimate and explicit purposes".

Therefore, it is necessary that FastFIX guarantees that personal data that could be collected is treated with the only and exclusive purpose described and declared in the scope of the project or service. Only in that case a legitimate treatment will be guaranteed.

Legitimization of the data processing  If the data subject has not previously given his unambiguous consent, the personal data processing in FastFIX's scope will have to comply with at least one of these conditions:

- Processing is necessary for the performance of a contract to which the data subject is party or in order to take steps at the request of the data subject prior to entering into a contract.

- Processing is necessary for compliance with a legal obligation to which the controller is subject.

- Processing is necessary in order to protect the vital interests of the data subject.

- Processing is necessary for the performance of a task carried out in the public interest or in the exercise of official authority vested in the controller or in a third party to whom the data are disclosed.

- Processing is necessary for the purposes of the legitimate interests pursued by the controller or by the third party or parties to whom the data are disclosed, except where such interests are overridden by the interests for fundamental rights and freedoms of the data subject.

In a first approach, FastFIX seems to achieve with the last condition. Although it could be interpreted as a legitimate interest of the controller that its applications suffer the fewest possible run-time bugs, and therefore any initiative to prevent, detect and correct any error situation could be understood as an initiative in line with the satisfaction of
the legitimate interest, it is not very clear that just this legitimate interest allows the controller to proceed with the application users’ personal data processing.

Therefore, a priori it seems that at least there is a need for having the consent of the subject (the user) whose personal data could potentially be collected in the runtime environment monitoring to analyze or predict a possible failure of execution of an application.

This assertion is further reinforced by the fact that there are other principles covered by the European Directive 95/46 EC which specify the need to inform the person who collected the data on the identity of the controller, on the treatment purposes or potential recipients of the data, as well as the existence of a number of rights in the processing of such data.

10.1. Application recording and user privacy

Once a monitoring mechanism is in place, privacy and security issues arise. The first one is the issue of protecting the trace log, the file containing the result of all program operations (or only the non-deterministic ones) including user input. This is an easy problem to solve. Most operating systems provide protection of file access between user accounts and, if needed, the trace file may be encrypted. The next privacy issue is the risk that the application trace may be leaked from the device. This is a more serious issue. If a device is disconnected from the network, there is little risk that the user data will be propagated outside the device and the device’s users’ privacy be put at risk. However, once the device running the monitored application has a network connection, the information leak becomes more difficult to control. With the protection of adequate security software (firewall and anti-virus) we may reasonably put aside the possibility that malicious software (virus, trojans, etc.) may transmit the trace file to the outside. This leaves us with the problem of trusting the developers of the monitored application. Developers have to be trusted not to include malicious mechanisms to leak user information. Possible ways to leak information without the user noticing are:

- Transmitting information to external devices without notifying the user.
- Transmitting more data than what the user is told.
- Providing access to the trace information to third parties.
- Using personal information in log traces for other purposes than fault replication.
- Designing malicious code that indirectly discloses personal information just by the places where it crashes, e.g. inserting faulty code in points of the application code that can only be reached when certain personal data has been input.

There is little that users who buy an application can do and, in the limit, this is an issue between users and developers whose final decision is legal. If we assume that the error reporting functionality is well behaved, another important but generally trivial problem is securing the transfer of the error report to the development site.

This leads us to the core issue in application monitoring for fault replication which is determining what information may reasonably be sent to the development site. There is a tension between helping the remote debugging process and protecting user privacy.
D7.1: Global performance, security and privacy requirements

If all user input data is sent to the development site for debugging, private data may be exposed. If users’ private data is withheld, execution replication may be impossible. Obviously all information pertaining to non-deterministic operations that is not critical information for the user may be sent to the software maintenance site without restrictions.

The reference in terms of privacy techniques in replication of faults is the work of Castro et al. [8]. This technique assumes that the application is being continuously monitored, and that a complete record of its execution is made so that it can be replayed in the future. When there is a fault, the program is re-executed symbolically - traversing the same paths of execution that led to the error - in which the input is replaced by symbols. These symbols are being constrained in each branch of the program, which enables the system to calculate, as the final result, all the conditions that the input data must meet in order for the execution path to be the same as that which led to the failure. Thus, using a SMT (Satisfiability Modulo Theories), you can create a new input that travels the same path of execution - and thus lead to the same flaw – as the original input, but that is independent from the original input. However, for paths that have a narrow field of possibilities or with only one value, there is a loss of privacy of user data. This loss is measurable through an entropy measure of the new input, which is presented to the client when it is asked for permission to send the error report with new input. The main disadvantages of this technique has the great disadvantage are the large computational burden and the fact that it may not guarantee the privacy of user data.

If we want to further reduce or filter the information that is leaked from the user to the developer there are some additional techniques that can be used:

- Labeling of sensitive information: With access to application code, the application can be modified so that users indicate which data may or not be transmitted to the developers so that critical data is never transmitted.

- Gradual use of information: Another technique that can be used is the gradual transmission of personal data [56]. If the client device is running when the developers are debugging a fault, we can avoid sending the whole application trace to the maintenance site. As the application is being debugged and more information is needed, the debugging environment may request from the client more trace information (including obviously parts of user data we aim to protect).

- Policy support: Finally, we can support the process of transmitting the trace to the developer with policy mechanisms that specify what types of information may be communicated [56].

10.2. Fulfillment of the in force legislation

In the analysis of the legal requirements for guaranteeing the suitable use of the personal data of the users in the monitoring process of the run-time environment applications, detailed analysis of in force data privacy legislation in every Member State is not going to be developed.

The big differences in application and interpretation aspects from one Member State to another, and the different perceptions of risk, have been commented previously. For example, in some States the development of a technical security document that describes the information systems and the security and organizational measures applied to personal
data processing is not mandatory, whereas in other States not only the existence of the security documentation is required, but with a high level of detail as well.

Therefore, the detailed and specific national security measures are not going to be analyzed, but the common and main security measures recommended not only for the regulation developments of every State but also for the best practices' guide defined in the ISO 27002 standard.

**Information access control** It is necessary to guarantee the existence of an authentication mechanism to control the physical and the logical access to the collected information of run-time environments according to his level of security (for personal data and for corporate information, financial information, of production, etc.). It is necessary to guarantee the nonexistence of user's generic identifiers. The users' management policies must consider not only the users with access allowed to replay and analyze the runtime bugs but the administrator users of the information systems as well.

The users' management policy must define a robust password policy and provide details about password expiration, syntax and strength.

Information access control must be based on the “need to know” principle, and must provide a procedure for application request, acceptance and authorization, with a suitable segregation of duties.

**Backup Policy** The current model legislation on protection of personal data requires that if the data used in simulation and/or preproduction environments is real data that has not undergone any obfuscation or "anonymity" procedure, it should be protected by the same policy of backup as if it was in a production environment. The media containing the backups should be stored in safe places, and if it could contain personal data requires special protection encrypting the information contained and stored in an alternate location.

There should also be a test plan and documented periodic restoration of existing backups, to ensure recovery of all the necessary elements of different managed environments in case of contingency.

**Data transmission** Regardless of the type of information processed, and the level of security of personal data that can be managed, the transmission of information on the events of the run-time environments of the monitored systems and applications will be made through secure communications networks to the Support Center, conducting a pre-encryption process if there are no adequate safeguards.

**Incident management** The current model legislation on protection of personal data requires that if the data used in simulation and/or preproduction environments is real data that has not undergone any obfuscation or "anonymity", a search, monitoring and management of security incidents process should be performed, in the different monitored environments and in the information systems management as well.

**Contractual considerations** If the Support Center that receives events of the runtime environments that occur in situations of potential runtime error is a third party legally independent of the organization in which the system is deployed for monitoring
the application, it will be necessary to address the signing of a legal annex, complements the commercial contract previously signed between the two companies, which regulates aspects related to the protection of personal data and writing that reflects the obligations that both figures (controller and processor) are, including an appropriate non-disclosure agreement.

The current legislation on personal data protection does not understand the concept of "group of companies" as understood from the business perspective. Resource and information services sharing between legally independent companies must be regularized from contractual terms, especially if in the "shared" information there is personal data.
11. The private use of corporate resources

In the current job environments, and in real life, the use of the corporate resources for users’ private purposes is generally accepted.

There are companies that provide independent email accounts to their employees (one for exclusively corporate use and other one for exclusively private use). This totally differentiated use contributes undoubted advantages, facilitating the work of IT departments at the moment of discerning of which mail accounts it is necessary to guarantee a suitable backup level or at the moment of measuring the size of the mail accounts. But at the same time, not having doubts about the possibility of accessing the corporate mail account of an ex-employee. Nevertheless this one is not a usual practice.

In the real world, employees use the corporate resources (laptop, mail account, internet access, etc.) mainly for a professional use, but also for a private use, and every time it’s getting more complicated to find the borders between both uses.

The “reasonable use” of the corporate resources for particular purposes is accepted, and it even relies on jurisprudential support in some Member States, judgments that support and protect the moderated use of these resources with private purposes by employees, guaranteeing for example the inviolability of the e-mail as a constitutional right.

But the accepted and consented coexistence of corporate and private information in the same environments of work doesn’t allow the companies to treat the above mentioned environments in a relaxed way. If the use of the corporate mail for private purposes is accepted, the employees mail accounts could hold personal specially protected information about their health, or about their relatives’ health, for example.

However, this mixed use should not limit the right of the organizations to monitor or control, for example, the use of the internet access by the users, in case of performance degradation or bandwidth saturation. But it does.

If the company is monitoring its employees’ internet access use, and controlling the web sites they are visiting, it may be putting itself in a serious problem, because before carrying out this monitoring, employees should have been clearly and explicitly informed.

Though a moderated use of the corporate resources with private purposes is accepted, the company also has the right and the legal authority, in case of need, of controlling the use of these resources.

To do this, policies of use of the corporate resources must be developed, with specific regulations detailing his use. The security and monitoring measures the organization might use in case of need must be clearly detailed in these regulations.

And publishing the regulations in the intranet or in a general bulletin board is not enough. The organization must have the evidence of informing clearly the employees about their rights and obligations when using the resources put to their disposition, and its conditions of use. The evidence might consist in receiving the proper acknowledge of these regulations, physically or electronically.
FastFIX is directly affected by the existing conflict between the private use of the corporate resources and the right of the organization to monitor the use of his resources with the purpose of optimizing its use, its functionality and its performance.

FastFIX is based on the monitoring of the applications run-time environment to reach better performance levels of the monitored applications, and to obtain this way applications that learn from their errors and are increasingly proactive before the possible execution bugs. Evidently the application run-time environment is influenced by other coexisting environments, such as the operating system, virtual systems, other applications run-time environments, etc. The analysis of these external influences cannot be based on information obtained from test or pre-production environments. This analysis and this feedback learning mechanism need the run-time environment information.

The runtime monitoring and the environment analysis implies the capture of the external influences that can run into an application bug, and the recording of the processed information as in the runtime environment as in other coexisting environments. But neither these other environments nor the processed information are necessarily corporate.

It’s quite probable to be monitoring an application executed remotely from a private equipment, or a corporate application that, besides treating corporate information, can treat particular information, which in addition can be relative to natural persons.

Capturing, analyzing and processing the runtime environments of an application executed in a private environment, as well as of a corporate application that can treat personal information, will mean having informed beforehand the application users about the implications of the monitoring and of the information type that could be treated.

In the first case this previous information should have carried out by means of the application conditions of use, while in the second one it should be with the previous delivery of the policies and use regulations of the corporate resources. And in both cases, with the previous "acceptance" of the application users.
12. FastFix Privacy Requirements

Finally, some requirements are presented in Table 12.1 to ensure user privacy. These requirements are built according to S2 experience on privacy management and the European Commission Directive 95/46/CE [15].
<table>
<thead>
<tr>
<th>ID</th>
<th>Target</th>
<th>Privacy Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User of monitored applications</td>
<td>The user of monitored applications must be informed in an explicit way of the fact that his job environment is being monitored, specifying that the “job environment” includes not only the monitored application, but any other user application as well. This information should be given with the most adapted method in every case (conditions of use of an application, policies and use regulations of corporate resources, etc.). The organization must have an evidence of having informed the users about the possibility of personal data getting involved in the process of monitoring of the affected run-time environments. The monitoring system must guarantee that the captured run-time environments and, in general, the information monitored (especially if it contains personal data) is controlled inside secure systems, with controlled and authorized access profiles.</td>
</tr>
<tr>
<td>2</td>
<td>External services</td>
<td>The legal situation of all the external services that can be received from third companies (in the phases of capturing, communicating, processing or analyzing the information) should be regularized. The corresponding not disclosure agreements and the contractual annexes that regularize the personal data legal situation of the obligations of every part must be formalized.</td>
</tr>
<tr>
<td>3</td>
<td>Personal data</td>
<td>The personal data compiled in the process of monitoring and later analysis must be dedicated only and exclusively to the mentioned and declared purpose, not being able to be used for any other purpose nor by any other premeditation (interpretation of personal tastes or affinities, analysis of labor productivity, efficiency, market researches, etc.). The dedication of the information to any other not declared purpose will be considered to be illegal if the data subject’s consent has not been obtained before in an informed and unequivocal way.</td>
</tr>
<tr>
<td>4</td>
<td>Personal data</td>
<td>When the personal data compiled for the analysis, replaying and debugging process does not contribute relevant information to the analysis of the bugs, the personal data will be submitted to a dissociation processes, in order to avoid the identification of the user.</td>
</tr>
<tr>
<td>5</td>
<td>Personal data</td>
<td>Exclusively in the cases in which it is absolutely necessary to capture information of coexisting runtime environments to the environment of the monitored application - and specially if in these environments information of private type could exist - the personal data capturing will be carried out with the necessary security measures to guarantee its confidentiality: use of obfuscation technologies of the information, encrypting the information on its flow through communications networks as well as in its storage, etc.</td>
</tr>
</tbody>
</table>

Table 12.1.: Privacy requirements
Bibliography


[10] CORP;ORACLE, I. A migration guide commissioned by Intel Corp. and Oracle.


[16] EDUARDO. debugme-4 Checksums.


[28] KRISHNAN, P. Hardware Breakpoint (or watchpoint) usage in Linux Kernel.


D7.1: Global performance, security and privacy requirements


[37] MICROSOFT. Event Tracing.

[38] MICROSOFT. Overview of System Center Desktop Error Monitoring.


D7.1: Global performance, security and privacy requirements


