<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, Virtualisation</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>

---

**D3.1: State-of-the-art Context Elicitation, Context Modeling and User Modeling**

**Lead contractor:** TUM

**Author(s):**
Emitzá Guzmán, Sergio Zamarripa López, Joao Coelho Garcia, Nitesh Narayan, Benoit Gaudin, Walid Maalej

**Submission date:** September 2010

**Dissemination level:** Public

---

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

© S2, TUM, LERO, INESC ID, TXT, PRODEVELOP
Abstract: This report describes the state of the art of context awareness and user modeling, including the elicitation, processing and modeling of context. Specific attention is given to main issues in context awareness such as privacy, incompleteness and ambiguity. We discuss how context information relevant for software maintenance can be collected. We also discuss how context in general and users in particular can be modeled based on semantic technologies and machine learning and compare both approaches. Finally, we present first uses of context in the domain of remote software maintenance.
# Contents

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

2 Foundations ................................................. 8
   2.1 Terminology ........................................... 8
   2.2 Activities of Context Awareness .................... 9
   2.3 Main Issues of Context Awareness .................. 10

3 Approaches for Context Awareness ......................... 12
   3.1 Context Elicitation .................................... 12
   3.1.1 Application Execution ........................... 12
   3.1.2 User Interaction ................................... 15
   3.2 Context Modeling ...................................... 16
   3.2.1 Context Attributes ................................. 17
   3.2.2 Application Execution ............................ 18
   3.2.3 User Interaction ................................... 19
   3.2.4 Context Representation ........................... 20
   3.3 Context Processing .................................... 27
   3.3.1 Semantic-based Processing ....................... 28
   3.3.2 Machine Learning .................................. 30
   3.4 User Information Processing ......................... 33

4 Context-aware Frameworks .................................... 36

5 Context-aware Applications for Software Maintenance .... 41

6 Conclusion .................................................. 45

Bibliography ................................................... 47
List of Figures

2.1 Activities of Context Awareness ........................................ 9
3.1 Context Attributes .......................................................... 18
3.2 RDF relations between subject and object. .......................... 22
3.3 Instance, Class and Property representation in OWL. ............... 22
3.4 Aspect-Scale-Context model [60]. ....................................... 23
3.5 CONON context model [65]. ............................................. 24
3.6 (a) Application of crisp limits for data interpretation. (b) Application of fuzzy sets for data interpretation [35]. ....................... 29
3.7 Type inference. ............................................................... 30
List of Tables

3.1 Comparison of elicitation approaches. ........................................... 14
3.2 Summary of user interaction systems. ............................................. 17
3.3 Summary of context modeling approaches. ..................................... 25
3.4 Selecting a suitable machine learning technique [19]. ....................... 33
3.5 General characteristics of the machine learning techniques [19]. .......... 34

4.1 Summary of context-aware frameworks. ....................................... 40
1 Introduction

Software needs to be modified in order to correct its faults and to adapt to changes in the environment. Software maintenance is the process of modifying the software system or its components in order to correct faults, improve performance, and adapt to the environment [29]. It includes tasks such as the correction of errors, the improvement, addition and removal of functionality and the enhancements regarding performance and functionality.

In computer science, context awareness refers to applications that can sense, understand and react to changes in their environment [51]. Context awareness is important for software maintenance because it improves understanding of the circumstances under which an error occurred, making it easier to reproduce and correct. Errors occurring in an application can be reported by users through emails, forums, tickets or groupware systems. A significant gap between the perceived usefulness of information by users and software maintainers exists [14]. The automatic elicitation of context information related to the application execution and user interaction and the representation of this information represents the first step to solve this problem. Besides fault correction, context awareness is a valuable tool in varied software maintenance tasks. Examples of these tasks are: user adaptation, improvement, addition or removal of functionality and performance enhancement. The monitoring of context information related to users’ interaction with an application can help decide which application functionality is not being used or needs to be added. Moreover, the monitoring of context information related to the application execution environment can be monitored in order to control the application’s performance and make the necessary adjustments.

This document aims at discovering possible approaches to be taken when building context-aware systems for software maintenance, as well as at pointing out new research directions in this area. Context-aware systems consist of three main activities: context elicitation, context modeling and context processing. In this state of the art we analyze different approaches for context elicitation, modeling and processing, and identify their strengths and weaknesses.

The document is divided into 5 sections. Section 2 introduces the foundations of this research field including important concepts related to context awareness in software maintenance, as well as issues that need to be considered when building context-aware applica-
tions. Section 3 analyzes different approaches for context elicitation, context modeling and context processing. Due to the central role that users and their interactions with the application have in the software maintenance field special attention is given to user-interaction elicitation, user modeling and user processing. Section 4 summarizes of context-aware frameworks, whereas Section 5 summarizes context-aware applications related to software maintenance. Section 6 summarizes the major findings of the document.
2 Foundations

In this section we provide definitions for context and context awareness. The objective is to build a common vocabulary and to delineate the focus of the document. We also describe the main activities in context-aware applications and introduce the main challenges when building context-aware applications.

2.1 Terminology

The terms context, context awareness and user modeling are well supported in the literature, in the following we present commonly used definitions.

Context

Abowd et. al. [1] define context as any information used to describe a person, place, or object relevant to the interaction between a user and an application, including the user and applications themselves. Maalej defines context as the set of all events and information, which can be observed or interpreted in the course of the work, except those events and pieces of information that constitute the change (i.e. the main output of the work) [37]. In the software maintenance domain we define context as the aggregation of the application execution and the user interaction with the application. As Abowd et. al. we consider the user a part of context.

Context Awareness

Abowd et. al. [1] give a general definition of context-awareness as the ability of a system to use context in order to provide relevant information or services to the user. Hull et. al. [28] and Ryan et. al. [48] provide more specific definitions in which context awareness is defined as the ability to detect, interpret, and respond to aspects of the user’s and application’s environment. We define context awareness as the process of eliciting, representing and processing context in order to react or respond to the user, its environment and the application’s environment.
2.2 Activities of Context Awareness

Context-aware applications generally consist of three main activities: context elicitation, context modeling, and context processing.

**Context Elicitation**

In this component relevant information necessary for context awareness in software maintenance is sensed and recorded. We focus our attention in the elicitation of information related to application execution and user interaction. The elicitation of application execution is important for software maintenance because it enables fault replication and repair, whereas the elicitation of user interaction gives information about user’s tasks, goals and level of expertise.

**Context Modeling**

Context needs to be represented in a way that computers can read and understand it. The type of context model determines which context processing mechanisms can be applied, as some models are more suitable for some context processing approaches than others. The type of modeling approach also determines whether semantically meanings can be considered in the context model.

**Context Processing**

Context processing is concerned with interpreting and aggregating elicited information. Through context processing it is possible to extract implicit knowledge that can be incorporated in the context model and to detect errors in the context model.
2.3 Main Issues of Context Awareness

When building context-aware frameworks or applications, the following issues raise:

- **Privacy and Protection of Personal Data**: Context-aware environments collect numerous, possibly sensitive information about the user, including interaction histories and contents of private documents. Collecting this information brings issues about the user’s privacy. Such information can be abused, misinterpreted, or even sold for marketing agencies. Acceptable trade-offs need to be evaluated, as well as methods that protect user’s privacy while collecting their sensitive information.

- **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 [36].

- **Heterogeneity**: When building a context-aware framework context observation should be done independently from the monitored application and the usage domain. An abstract notion of context needs to be defined in a universal model, in order to be used in different scenarios. In order to do so, the answer to questions, such as, what is part of context? and what should be observed and what not? should be answered [36].

- **Scalability**: Depending on the type of context-aware application, different amounts of context information will be elicited, processed and modeled. Models and processing techniques should be able to handle high amounts of data.

- **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 [59].

- **Incompleteness and Ambiguity**: Information acquired by the sensors tends to be incomplete or of an ambiguous nature. The context model should include mechanisms in order to deal with this. Interpolation of incomplete data on an instance level is a possible solution to this challenge [59].
• **Level of Formality:** In context awareness a shared understanding allows for the interpretation of information by modules or context-aware applications. In order to enable a shared understanding, contextual facts and their interrelationships should be represented in a precise and traceable manner [59].
3 Approaches for Context Awareness

In this section the most representative approaches of the context awareness activities presented in Section 2.2 are summarized and compared.

3.1 Context Elicitation

In Context elicitation information is monitored and recorded. We focus on the elicitation of information related to the application execution and the user interaction with the application. Context can be monitored through so called sensors. The word sensor not only refers to sensing hardware, but also every other data source which provides context information [6]. Sensors can capture physical data, e.g. temperature and acceleration, or data from software applications or services.

3.1.1 Application Execution

A central part of acquiring context information for software maintenance is the monitoring of applications execution. Applications can be traced in order to enhance fault replication and repair. Tracing applications can be approached with different techniques, with different costs and overheads, ranging from the pragmatic and well-known to the experimental and esoteric.

State of Crashed Application

The most basic context elicitation technique for fault replication is gathering reports describing how the application at stake has crashed. Systems like Microsoft’s Dr. Watson\(^1\) and Mozilla’s Talkback\(^2\) implement this approach. These systems resort to the core dump mechanism provided by the operating system, which generate an image of the state of a process that has crashed. The file containing the application state at crash time has varied formats depending on the operating system. In general, it contains a snapshot of the processor state and the contents of the application’s process memory. The state information can be included in a report sent to the application developers who use these

\(^1\)http://oca.microsoft.com/en/dcp20.asp
\(^2\)http://wp.netscape.com/
reports to correct a possible bug. Sometimes, these reports do not include the full state of the crashed application (for privacy or brevity reasons). Instead the report only describes the point where the application’s process and its threads stopped. This approach does not involve direct communication from the crashed application to the context-aware application. Typically, the context-aware application would read the core dump of the crashed application (directly or via a specialized sensor) and would integrate it into the context model.

**Checkpointing**

Simply describing the state of an application after it crashes makes maintaining the application challenging. A more advanced technique is to periodically record the state of the application so that previous points of the application execution (during which it was executing correctly) are known. This is commonly known as *checkpointing*. This approach can be used for fault tolerance, in order to re-execute the application without having to start from the beginning. But it can also be used to partially replay an application [17], especially if several checkpoints are recorded and kept. In this case, the triggering of the checkpoint and the transmission of the resulting information, a form of application execution context information, to the context elicitation can be controlled either from the checkpointing infrastructure or periodically triggered by a module of the context-aware application.

The approaches presented up to here provide substantial information about the execution of an application but do not provide enough context information for the full replay of an application. In order to elicit all context information needed for fault replication, all operations performed by an application that can have non-deterministic results (values that vary between different executions) must be detected and its results must be recorded.

**Library-based Logging**

Library-based logging, involves modifying applications so that all non-deterministic operations are intercepted by a library that performs the logging, such as in Jockey [49] and liblog [20]. In these cases, the information generated by each relevant call is clearly delimited in the returned values that can be submitted to the rest of the context-aware application.

**Virtual Machine Tracing**

A simpler but very relevant topic is the tracing of application’s running within virtual machines, in particular user-level virtual machines. Virtual machines are currently very popular and the execution environment for a large part of recent applications due to the
### Approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Recorded Information</th>
<th>Replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>State of crashed application</td>
<td>State of the application when crashed</td>
<td>None</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>State of the application in different points of execution</td>
<td>Partial</td>
</tr>
<tr>
<td>Library-based logging</td>
<td>All results of non-deterministic operations</td>
<td>Full</td>
</tr>
<tr>
<td>Virtual machine tracing</td>
<td>Byte code triggering system calls and execution order of threads</td>
<td>Full</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of elicitation approaches.

The popularity of both the Java Virtual Machine and Microsoft’s .NET Common Language Runtime. Acquiring the needed information to replicate the execution of a virtual machine based application is simpler because the application is contained within an environment which is easy to instrument and control. Virtual machine applications are typically compiled from source language to a high-level assembly language, called bytecodes, which are then, in runtime, interpreted or compiled just-in-time. Therefore, in order to record the application execution, it is only necessary to intercept and record the execution of byte codes which trigger system calls, as well as the execution order of the threads within the virtual machine [?; 68].

The cost of tracing, can be adjusted if, during the re-execution, one is able to make up for missing information by experimenting with alternative re-execution paths, such as done in PRES [42]. Multi-core applications are traced by using points of reference of varying granularity which can be adjusted to limit the overhead of tracing. With fine grained traces, the original execution of the application can be quickly repeated, whereas with course grained tracing, an application has to be replayed several times until the correct re-execution is found.

### Comparison

In this section different applications for context elicitation where described. Table 3.1 summarizes the presented information. The approaches of recording the state of the crashed application and checkpointing are not enough for a full application replay, as not enough context information is given. In order to have a full replay all operations that have non-deterministic results need to be recorded. Library-based logging intercepts and records all non-deterministic operations through available libraries. The elicitation of information for the replay of application running on virtual machines is easier. It is only necessary to intercept the byte code which triggers system calls and the execution order of the threads.
3.1.2 User Interaction

User Interaction includes information about user activities such as mouse clicks, bar scrolls and data input. It is usually based on the monitoring of user interface events. Processing (interpreting and aggregating) user interface events is sometimes easier than processing the application execution trace. The elicitation of user interaction is important because it allows further understanding of what the user was trying to do when the error occurred. Important factors such as the cause of an error by a determinate sequence of user interface events can also be identified through user interaction. Besides, user interaction information is useful for determining the type of user and its characteristics. This information can be used in order to decide whether to aggregate, eliminate or modify the application’s functionality. In this section we present some user interaction elicitation approaches. These approaches are organized by the type of application from which the user interaction is monitored.

Desktop Applications

Hackystat [13] and ElectroCodeoGram [53, 52] are general frameworks for capturing developer context. They record empirical data about the processes developers pursue during the daily work [44]. Both applications monitor, therefore, a broad range of developer activities. Hackystat and ECG implement a client-server architecture, where sensors submit their data to a central analysis component. The sensor-API is modularized, allowing new sensors to be added. Besides a sensor for Eclipse, the Hackystat framework provides a dozen further sensors for other IDEs (JBuilder, Visual Studio, etc.). Sensors capture events mapped to fifteen categories (so called Sensor Data Types). These categories are tool-independent concepts with certain attributes. For each Sensor Data Type, several sensors can exist that create instances (e.g. Build-events from different IDE sensors). ECG primarily targets the Eclipse IDE, capturing several event types, such as window-activation, editor interaction, run and debug-actions, file changes, and tracking the focus of developers reading source code [67]. On top of this log data, modules implement algorithms to discover behavioral patterns and refine the basic interactions to certain episodes, such as duplicating code.

The INTI framework [37] monitors user interaction in order to detected user’s intentions and organize information across the different tools a user utilizes. INTI elicits information by instrumenting the user environment and the tools utilized by the user. INTI makes use of DTrace\(^3\), a tracing framework which is used in order to listen to the execution of particular functions, which reveal a type of interaction. It instruments user interfaces in order to make use of the similarity between the framework’s interaction ontology and

\(^3\)http://www.sun.com/bigadmin/content/dtrace
the labels of window menus. Furthermore, metadata about artifacts is sensed from file systems or from accessibility and user interface frameworks.

**Web-based Applications**

Some approaches concentrate on monitoring the activities of web-pages, as the ones described in the work of Atterer et al. [4]. In this approach an HTTP proxy modifies the HTML pages by adding Javascript code before delivering it to the client’s browser. The Javascript code allows for the collection of information such as keyboard input, mouse movement and scrolling. In order to track the interaction at the widget level the HTML layout is used. This allows to, e.g., map mouse coordinates to buttons, links, etc. This approach has the advantage that no software needs to be installed on the clients side and of being platform independent.

Some approaches use installed software in order to monitor user’s interaction in desktop and web-based applications. Fenstermacher and Guinsburg [18] describe a framework to study information access and usage within and across applications. They monitor and record the occurred events through the use of the Component Object Model (COM)\(^4\). Because of this the operating systems platform is limited to Windows and the monitored applications need to have a COM interface.

**Comparison**

User interaction elicitation is mainly done on specific applications, while focusing on certain types of users. Table 3.2 summarizes the most important aspects of each of the previously mentioned user interaction frameworks. The table summary takes into account the type of monitored application, the sensed data, and the type of user the elicitation approach focuses on. All of the elicitation frameworks and systems that were analyzed limited the set of applications where the user could be monitored. In this sense ElectroCodeoGram was the most limited one as it focused exclusively on elicitation developers interaction in an Eclipse environment. With the exception of Atterer’s et. al. approach, all analyzed frameworks and systems required the installation of software on the user’s side.

**3.2 Context Modeling**

In this section we summarize the most important attributes for context and user modeling in software maintenance, as well as representation mechanisms for context and its attributes.

\(^4\)http://www.microsoft.com/com/
<table>
<thead>
<tr>
<th>Framework / System</th>
<th>Monitored Applications</th>
<th>Sensed Data</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hackystat</td>
<td>Eclipse, JBuilder, Visual Studio, Microsoft Office, JIRA</td>
<td>User Interaction data: time editing, user input, window activation, etc.</td>
<td>Developers, Desktop application users</td>
</tr>
<tr>
<td>ElectroCodeoGram</td>
<td>Eclipse</td>
<td>User Interaction data: window activation, debug, file changes...</td>
<td>Developers</td>
</tr>
<tr>
<td>INTI</td>
<td>15 Mac OS applications (e.g. XCode and Mail)</td>
<td>Application execution, User Interaction data</td>
<td>Developers</td>
</tr>
<tr>
<td>Atterer</td>
<td>Web pages in HTML</td>
<td>User Interaction data: Keyboard input, mouse movement, scrolling</td>
<td>Web users</td>
</tr>
<tr>
<td>Fenstermacher</td>
<td>Applications with COM interface</td>
<td>User Interaction data: All types of COM events</td>
<td>Desktop and web users</td>
</tr>
</tbody>
</table>

Table 3.2: Summary of user interaction systems.

Section 3.2.1 gives an overview of the most important attributes to be modelled when considering context in any kind of domain. Afterwards it summarizes the most important context attributes in the software maintenance domain dividing them into two categories: those abstracted from the application execution and the ones that are obtained by monitoring the user’s interaction with the application. Section 3.2.4 summarizes several approaches for representing context according to the used data structure scheme.

### 3.2.1 Context Attributes

When building context-aware systems we need to decide which attributes concerning context will be modeled. We use the term context atom in order to designate a specific aspect of context to be modeled. Context atoms, can be represented with five basic attributes: type, value, time stamp, source and confidence [6].

- **Context type**: Refers to the category the context belongs to, e.g., temperature, time, speed, etc.

- **Context value**: Refers to the raw data gathered by the sensor. The unit depends on the context type and the applied sensor, e.g., degree Celsius, hours, miles per hour, etc.

- **Timestamp**: This attribute contains a date/time-value describing when the context was sensed. It is needed, e.g., to create a context history and deal with sensing conflicts.
• **Source:** A field containing information about how the data was gathered. In case of a hardware sensor it might hold the ID of the sensor and allow an application to prefer data from this sensor.

• **Confidence:** The confidence attribute describes the uncertainty of the context type. Not every data source delivers accurate information, e.g., location data suffers inaccuracy depending on the used tracking tool. This inaccuracy can be expressed through confidence labelling.

### 3.2.2 Application Execution

The context of the application execution is important to software maintenance in order to understand in which state the application was when an error occurred. When the necessary context information is recorded the error can be replayed. The application execution context necessary for enabling a replay during the software maintenance process can be classified as follows:

• **General Details:** This entails registering and verifying that the correct versions of the application, run time libraries, virtual machine (VM)/interpreter (if one is used), and operating system (OS) are used.

• **User Input:** The original user input has to be recorded. Only the user input that is decisive for the execution path of the application need be submitted. However, selecting only the critical input requires a complex analysis of the application.

• **Sources of Non-determinism:** The sources of non-determinism in the execution environment are all inputs to the application besides the user input. Since applications do not interface directly with the hardware or the external world, but rather interact with a VM or the underlying operating system, one must ensure that all calls to the VM/OS that may be non-deterministic must be recorded to be replayed. It is obvious that this work is highly specific to the particular environment being used. Examples of such calls are: reading the system clock, requesting a random...
number, reading the CPU load, requesting a process’s or thread’s identifier, etc. In summary, all calls to the system that return circumstantial values, as well as asynchronous events (e.g. signals) that generate application input without an explicit call. Regarding the input from inter-process communication such as file systems, sockets and pipes, these can be recorded when the corresponding system calls are performed. Finally, probably the least obvious source of non-determinism in application execution is scheduling. In single-threaded applications, scheduling does not alter the application execution path. However, in multi-threaded applications, varied scheduling of concurrent threads within an application can result in different execution paths. To control this non-determinism, the transition points between threads must be recorded. This problem is exacerbated in multi-core processors where concurrency is not delimited by the operating system scheduling but is enhanced by having an application’s threads each running on its own core. Therefore, in multi-core, additional information must be recorded to ensure that data races between threads on distinct cores can be detected. This may require recording the output of each and every memory operation but techniques exist to reduce the amount of recorded information.

### 3.2.3 User Interaction

Kass and Finin [33] identified several attributes for modeling users:

- **Goals and Plans**: Refers to what the user wants to achieve and how he wants to do it. This information needs to be inferred from the user’s actions or statements. Plan recognition is an active research field in Artificial Intelligence and is especially difficult when user’s goals are based on false beliefs [40].

- **Capabilities**: Refers to the level of expertise a user has for executing a certain task. This information is usually inferred or learnt and is specially helpful for applications seeking to present information to the user according to its level of expertise.

- **Preferences**: Detecting users preferences can be helpful for adapting applications. This information is used by for e.g. recommender systems.

- **Beliefs and Knowledge**: Adaptive systems usually record what a user knows or does not know about the system. This information is sometimes acquired explicitly or through inference rules.
3.2.4 Context Representation

Strang and Linnhoff-Popien [59] classified the following approaches for modeling context according to the data structure scheme used for representing and exchanging contextual information.

**Key-value**

The key-value pair is the simplest data structure to model context. It was used by Schilit et al. [51] in order to model context, such as location information. This approach is frequently used in distributed service frameworks, where services are described with a list of attributes in a key-valued matter. Service discovery is applied by using an exact matching algorithm on the key-valued attributes. These models are easy to manage but have the disadvantage of a lack of adequate structuring for efficient context retrieval algorithms.

**Markup scheme**

Markup schemes generally organize their information in a hierarchical way, making use of markup tags, attributes and content. One type of markup model schemes are *profiles*. Profiles are generally based on the Standard Generic Markup Language (SGML)\(^5\) and are generally extensions of the Composite Capabilities / Preferences Profile (CC/PP)\(^6\) and User Agent Profile (UAProf)\(^7\) standards, which propose the use of the Resource Description Framework (RDF)\(^8\) serialized in XML. One of these approaches is the one proposed by Indulskal et al.\([30]\) in which CC/PP and UAProf were extended in order to add context related concepts such as location, network characteristics, application requirements and session information. In their work the authors note that while their approach enables context awareness in applications it is difficult to capture complex contextual relationships due to the use of CC/PP. There are other existing markup scheme models, but as Strang and Linnhof-Popien [59] mention they are either partly/not available to the public and/or limited to few contextual aspects.

**Graphical**

The Unified Modelling Language (UML) can be used in order to model context, as done in the work of Sheng and Benatallah [56] where a UML-based language was used. Henricksen et al.[26] describe another approach based on Object-Role Modeling (ORM)\(^9\). Graphical

---

\(^5\)[http://www.w3.org/MarkUp/SGML/]
\(^6\)[http://www.w3.org/Mobile/CCPP/]
\(^7\)[http://www.wapforum.org]
\(^8\)[http://www.w3.org/RDF/]
\(^9\)[http://www.orm.net/]
approaches can be useful for deriving ER-models which can be used for structuring context related relational databases.

**Object oriented**

The main benefit of utilizing an object oriented model is the use of object oriented attributes such as encapsulation, reusability and inheritance. In this approach various objects are used in order to represent different context types (location, network traffic, etc.) and encapsulate the details of context processing. Access to context information is provided through well-defined interfaces. Bouzy and Cazeneve [9] use object oriented mechanisms in order to represent temporal, goal, spatial and global contextual knowledge about the Go game in a computer. They justify the use of an object oriented approach due to the inheritance and reutilization capabilities. The GUIDE project [11] also uses an object oriented approach in order to model context. It encapsulates the details about context collection and fusing (e.g. context adaptive composition of HTML fragments) in objects, hiding the information from other system modules.

**Logic-based**

Logic-based models are formal models which make use of facts, rules, and expressions. These type of models allow to make inferences in order to derive new facts from existing facts and rules. McCarthy [38, 39] was one of the first to use logic in order to model context. In his work he introduces the ist$(c,p)$ predicate. This predicate means that proposition $p$ is true in context $c$. In McCarthy’s approach contexts can be used to define other contexts, expressed as $c' : ist(c,p)$ which means that proposition $p$ is true in the context of $c$, itself asserted in an outer context $c'$. McCarthy’s proposed formalism allows for the inference of contexts and propositions and for the establishment of relationships between propositions of different contexts, which can be expressed by the $ist(c,p) \iff ist(c',p')$. This can be clarified with the next example [55]

$ist(\text{context:: Holmes' stories, "Sherlock Holmes lives close to Victoria station"})$

$\iff$

$ist(\text{context:: real London, "Baker street is situated close to Victoria Station"}).$

Numerous logic-based approaches can be found in the literature, some of them are summarized in [3, 59].

**Ontology-based**

Ontologies can be understood as models that describe sets of objects, their relationships, and constraints in a domain of interest. This domain is either a part of reality or an entirely fictitious environment. The universe of objects and relationships is expressed in
a declarative, formal vocabulary that collectively constitutes the knowledge about the domain [21].

RDF and OWL (Web Ontology Language)\textsuperscript{10} the most popular ontology representation languages. RDF provides a structure for describing different domains. RDF representation of knowledge is very similar to natural languages. In RDF, ontologies are represented as a set of statements in the form of subject-predicate-object expressions. The subject denotes the resource, while the predicate denotes properties or aspects of the resource and expresses a relationship between the subject and the object. Figure 3.2 illustrates an example where the notion “John has the role of project manager” is represented as a triple with the subject “John”, the predicate “has the role of ” and the object “Project Manager”.

OWL extends RDF and consists of the following abstractions:

- **Instances**: Represent objects in our domain of interest.

- **Classes**: Represent concepts and group sets of instances.

- **Properties**: Represent types of associations between concepts or between instances.

Figure 3.3 illustrates an example of classes, instances and properties. Instances are represented with diamond shapes, whereas classes contain instances and are represented with circles. Properties are represented with arrows, joining either classes or instances.

Knowledge in OWL ontologies is specified through logical formalisms. According to their level of expressiveness they are divided into: OWL Lite, OWL DL and OWL Full. OWL Lite intends to support classification hierarchies and simple constraints. OWL DL includes all OWL language constructs under some restrictions to preserve decidability.

\textsuperscript{10}http://www.w3.org/TR/owl-features/
and is based on a sublanguage of Description Logic [5]. OWL Full is based on different semantics from OWL Lite and OWL DL. There are no systems which support complete reasoning for OWL Full. OWL DL is often preferred [64], since it provides maximum expressivity, while retaining computational completeness, decidability and the ability of practical reasoning algorithms.

Due to the fact that OWL uses logical formalisms for representing knowledge, a reasoner can be used to infer new implicit knowledge on both terminology and instances [55]. Reasoners can, e.g. derive a hierarchy of concepts, infer new types, and add new relationships.

Ontologies are well suited to define semantic mappings for heterogeneous data [45, 57, 63]. Different knowledge representations of the same domain can be incompatible even when using the same meta-model for implementation. For example, the concept “release” might be modeled and named differently in two systems that both use SQL to represent knowledge. To deal with such situation, ontologies offer constructs such as “inverse of”, “equivalent to”, “language label”, etc.

One ontology-based approach for context modeling is Aspect-Scale-Context (ASC) model [60] which uses ontologies defined in the Context Ontology Language (CoOL) defined by the designers of the model. CoOL uses OWL and F-Logic as ontology languages and OntoBroker\(^\text{11}\) as an inference and querying mechanism (in order to query ontologies with OntoBroker ontologies need to be expressed in F-Logic). ASC models context through the core concepts aspect, scale and context information. Each aspect aggregates one or more scales and each scale aggregates one or more context information, as illustrated in Figure 3.4. The ASC model is used in order to enable context awareness and contextual interoperability during service discovery and execution in a distributed system architecture.

Figure 3.4: Aspect-Scale-Context model [60].

Another similar approach is the one developed by CONON [65], which uses the OWL-DL language and proposes 4 basic entities for modeling context: Computational entity,

\(^{11}\)http://ontobroker.semanticweb.org/
location, user and activity. In order to reuse information among different types of context they propose the separation of general and specific domain concepts into two different ontologies. Figure 3.5 illustrates the general ontology, denominated the upper ontology of the CONON context model.

C-OWL [8] is another ontology-based approach which modifies the OWL semantics in order to allow the interrelationship of concepts, roles and individuals in different ontologies. This needs to be done in order to relate contexts from different domains.

Comparison

We evaluate the suitability of each of the previously presented approaches for modeling context based on the following criteria [59].

- **Partial Validation:** Context models should offer the possibility to validate contextual knowledge on the structural and instance level against a context model. Partial validation is important for context awareness because it allows to detect error-prone context models.

- **Richness and Quality of Information:** Models should be able to support richness and quality indication as the quality and richness of the information acquired by the sensors varies over time.

- **Incompleteness and Ambiguity:** Models should be able to handle the incomplete and ambiguous information acquired from the sensors, with the use of, e.g. an interpolation function on the instance level [59].
<table>
<thead>
<tr>
<th>Approach</th>
<th>Partial Validation</th>
<th>Richness, Quality</th>
<th>Incompleteness, Ambiguity</th>
<th>Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-value</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>No scheme to match against.</td>
<td>No complex context information because of lack of possibility to represent relationships. Quality indication is not supported because the addition of meta-information is not possible.</td>
<td>Incompleteness can only be handled on instance level. Ambiguity is not considered.</td>
<td>No formal mechanism.</td>
</tr>
<tr>
<td>Markup-scheme</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Scheme can be defined and set of validation tools are available for type and range checking. Complex representations can also be validated.</td>
<td>Meta-information can be added. Structure of markup scheme helps to handle richness of information.</td>
<td>Have to be handled on the application level.</td>
<td>Development of comprehensive scheme is step towards high formality.</td>
</tr>
<tr>
<td>Graphical</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Possible.</td>
<td>Complex context information can be modelled. Quality of information may be considered through meta-information.</td>
<td>Does not have mechanisms to cope with incompleteness and ambiguity.</td>
<td>Used mainly for human structuring purposes.</td>
</tr>
<tr>
<td>Object oriented</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Using a compiler on the structure level and a runtime environment on the instance level.</td>
<td>Complex context information can be modelled. Quality of information can be modeled.</td>
<td>The quality information can help deal with incompleteness and ambiguity.</td>
<td>Through the definition of well defined interface Drawback: Invisibility through encapsulation.</td>
</tr>
<tr>
<td>Logic-based</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Difficult to maintain, making the model error-prone.</td>
<td>Complex context information can be modelled. Quality of information can be modeled.</td>
<td>Handling of incompleteness is possible through rules. Ambiguity is not addressed.</td>
<td>Logic is high level formal language.</td>
</tr>
<tr>
<td>Ontology-based</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Several of tools enable this.</td>
<td>Complex context information can be modeled in hierarchical structure. Quality of information can be modeled.</td>
<td>Able to deal with incompleteness and ambiguity when inference mechanisms are incorporated into the ontology.</td>
<td>Ontologies can be built using formal languages.</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of context modeling approaches.
• **Level of Formality:** Context atoms and their relationships should be formally represented, enabling a shared understanding among all entities using the model.

With the exception of partial validation, the above mentioned criteria is part of the main issues when building context-aware systems mentioned in Section 2.3. We added partial validation due to importance of having consistent context models.

Table 3.3 summarizes the suitability of each of the modeling approaches for representing context in context-aware applications. From the previously presented modeling approaches the key-value modeling approach is the one with the largest disadvantages. Because there is no scheme representing the structure of the information partial validation is not possible. The lack of a scheme causes that incomplete or ambiguous information only be handled at the instance level and the interrelationships between contextual facts can not be represented. Because the addition of meta-data is not possible, there is no way to add a quality label to the context information.

Markup-schemes fulfill the partial validation requirement as the structure of context models can be validated through a previously defined scheme. Meta-data can be added allowing for the addition of labels concerning quality. The hierarchical structure of markup-scheme languages helps for the representation of complex information. The effort to develop a comprehensive scheme is a step towards formality. However, this approach has the disadvantage that the management of incomplete and ambiguous information has to be done at the application level.

Graphical models are specially useful for helping humans structure context information and then generating code or ER-models. Partial validation of certain graphical models, e.g. UML, is possible through exising tools. Incompleteness and ambiguity can be handled by the addition of quality labels, as done by Henricksen [25] in a revised version of his model. The formality of graphical models is low.

In the object oriented approach partial validation is possible: a compiler can be used in order to evaluate at the structure level, whereas a runtime environment can be used in order to evaluate at the instance level. The inclusion of quality information can be done through the inclusion of this field in the object oriented context model. Formality is achieved through well defined interfaces.

In logic-based context models partial validation is possible, but difficult to maintain. Meta-data can be added in logic-based context models, making it easy to add a quality indication for the sensed data. Even though not addressed in any of the logic-based model approaches mentioned previously, the incompleteness of information can be addressed through rules. To our knowledge there is no mechanism for handling ambiguity in logic-based models.

Analyzing the fulfillment of the previously defined evaluation criteria, ontologies are the most effective way to represent context. The hierarchical structure on ontological
languages, e.g. RDF and OWL, enables the representation of complex information. The use of logical languages in ontologies allows for the expression of context interrelationships through rules and for a shared understanding between application and modules using the context information. Reasoners can be incorporated into ontologies enabling partial validation of the information contained in the context model; this leads to the detection of inconsistent context models. SWRL rules are also useful for dealing with ambiguity. Furthermore, ambiguities can be handled through the ontologies’ hierarchical structure and through the URI\(^{12}\) used by ontology languages, e.g. RDL and OWL. To sum up, ontologies are an attractive way to model context if the appropriate formal language is used. In the next section we will see that multiple tools can be incorporated into ontologies order to realize context processing. Reasoners and rule languages which can also be incorporated into the ontology, e.g. SWRL\(^{13}\), are able to infer incomplete knowledge and perform aggregation tasks (which will be further explained in Section 3.3). For example, reasoners and rule languages which can also be incorporated into the ontology, e.g. SWRL, are able to infer incomplete knowledge and perform aggregation tasks (which will be further mentioned in Section 3.3). This incorporation does not need a large programming overhead, thus making it an attractive option to model context.

As will be further discussed in Section 3.3 the type of modeling approach determines which context processing approaches can be used.

**User-Interaction Representation**

User-interaction information is elicited and the data resulting from this process is used to create a *User Profile*. User Profiles contain information about the user that is important for the application using them. For example, a context-aware application for software maintenance will include information about the user’s goals, plans, capabilities, preferences, beliefs and knowledge. Golemati et. al. [22] and Trajkova and Gauch [62] use ontologies for representing model profiles.

**3.3 Context Processing**

In this section we review approaches for processing context information. Context processing activities include the following:

- **Interpretation:** Sensors collect information which needs to be mapped into context model concepts. Interpretation deals with the generation of these concepts from the

\(^{12}\)http://www.w3.org/Addressing/

\(^{13}\)http://www.w3.org/Submission/SWRL/
key-value pairs offered by the sensors. Interpretation is sometimes considered a pre-processing step. An example for interpretation when speaking about temperature is mapping the value -3 Centigrade into the value low and 100 Centigrade into the value high.

- **Aggregation:** Combinations of fine-grained sensed information can be used as indicators for higher-level information. For example, a sequence of button clicks, scrolling and reading activity can help identify when a user is performing e.g., a search, text editing, etc.

- **Type Inference:** Sensors are only able to detect a limited set of information. Reasoning facilities can be used in order to infer information about the type of user interactions or artifacts [36]. For example, if a sensor detects the interaction CreateMethod for a user in a software development environment then it can infer that the artifact must be a Method. Type inference is useful because not all information has to be collected from the sensors.

- **Inconsistency Detection:** When sensing information and building the context model, erroneous information can be incorporated into the model. The detection of these errors is also a part of context processing.

We divide the approaches taken in order to complete these activities into two categories: semantic-based and machine learning approaches.

### 3.3.1 Semantic-based Processing

The approaches that can be taken when realizing semantic-based processing are affected by the type of context model. Models with a high level of formalization enable inference of types and inconsistency detection, which cannot be done with non-formal models. In semantic-based processing it is important to consider the desired context processing activities, in order to choose an adequate model with which such activities can be realized.

**Interpretation**

The Context Management Framework [35] uses fuzzy sets and crisp limits for generating context atoms that applications can use or that can be further utilized to obtain higher-level concepts. The application of crisp limits result in a true-false labeling, as can be seen in Figure 3.6(a), where the membership to the Silent, Moderate and Load attributes, denoted as \( \mu(x) \), is either 1 or 0. The application of fuzzy sets results in continuous valued fuzzy labeling, with attributes assigned values between the continuous range of zero and one. An example is: \( \mu'(x) = 0.7/\text{Silent} + 0.3/\text{Moderate} + 0/\text{Loud} \) [35]. Figure 3.6(b) illustrates a graph of values an attributes when applying fuzzy sets.
Aggregation

The INTI framework [37] uses aggregation in order to generate user intention descriptions. It implements this functionality by using a relevance model and the semantics included in the ontologies (INTI’s context models). The relevance model orders the different interactions. Most frequent, long lasting, less aged interactions of the same type are the most important ones, and give an indication of the activities performed in the sessions. Each activity type that has a calculated relevance $REL(i)$ which is greater than the minimum threshold $minG$ is used to describe the user’s intention. The threshold $minG$ is set to a default value at the beginning and updated during the use of the framework. This update takes into consideration user feedback and uses naive Bayes classification in order to calculate the new value of $minG$. CORTEX [7] is a framework using an object-based context model. This framework uses CLIPS\(^{14}\) in order to infer higher level information based on the sensed information. CLIPS contains an inference engine, which given a number of facts decides which set of predefined rules it fires. CLIPS has the advantage that it can be integrated and further extended in languages such as C++ and Java. Gaia [47], a framework which represents context through 4-ary predicates in DAML+OIL, uses first order logic operations in order to create additional higher level knowledge. One example of an aggregation rule in this framework is demonstrated in below [6]:

\[
\text{Context(Number of people, Room 2401, >, 4)} \\
\text{AND} \\
\text{Context(Application, Powerpoint, is, Running)} \\
\Rightarrow \\
\text{Context(Social Activity, Room 2401, Is, Presentation)}.
\]

\(^{14}\text{http://clipsrules.sourceforge.net/}\)
Inference of types

INTI uses SWRL rules incorporated in their OWL ontology in order to deduce knowledge, they create and extend their rules manually or with association rule mining techniques [43]. Figure 3.7 shows an example of inference types. The elements marked in gray are the ones being inferred. The CORTEX and Gaia frameworks use CLIPS and first-order logic respectively in order to infer types.

Detecting inconsistencies

The detection of inconsistencies is generally done in ontology-based or markup scheme context models, due to the possibility to define a scheme with which the structure can be validates. Ontologies make use of logic-based reasoners. The CoBra framework [10] uses the Flora-2 logic-based reasoner in its F-OWL language in order to not only detect, but also correct inconsistencies in the context-model. OWL-based ontologies can make use of reasoners, such as FACT\textsuperscript{15} and Pellet\textsuperscript{16} in order to detect inconsistencies in their context models.

3.3.2 Machine Learning

Several machine learning techniques have been used in order to process context information from the sensed data. The processing activities that can be done with machine learning include: interpretation, aggregation and inference of types, besides the prediction of user’s actions and preferences, as well as the relevance of user’s actions. We distinguish between supervised and unsupervised learning techniques. Supervised techniques are those in which the training data is previously classified (previously classified data is called labeled data). The learner deduces the characteristics of each class used in the labeling of the training data. Unsupervised learning techniques do not need that the training data be pre-classified. They define clusters of training data. In unsupervised learning the learner defines the classes in which data will be classified, whereas in super-

\textsuperscript{15}http://owl.man.ac.uk/factplusplus/
\textsuperscript{16}http://clarkparsia.com/pellet/
Unsupervised learning

This approach consists of clustering and association rules.

- **Clustering:** Clustering consists of grouping unclassified data together. The groups in which the unclassified data are classified are created based on similarities found in the training data. Clustering techniques can have a hard or fuzzy nature. In hard clustering data points belong to exactly one cluster, whereas in fuzzy clustering data points can belong to more than one cluster, and each data is associated to membership degree values. Clustering algorithms are generally used in order to cluster users that establish similar behavior patterns (stereotypes) and in order to group applications, such as web pages, which have similar content. Clustering for user modeling is usually done with non-numerical data. The main problem with using clustering techniques is the definition of distance to be used. Distance is the parameter which indicates the similarity/difference between two data points. This is especially difficult in user modeling because the data is non-numerical. The works of Joshi et. al. [31] and Mobasher et. al. [41] are examples of how web-user behavior can be translated into numerical data.

- **Association rules:** Association rules discover frequent patterns among a group of transactions. Agrawal et. al. [2] introduced the problem, when trying to detect the frequent items in a market basket and the associations between them. Association rules identify actions that have a causal relationship. One common use of association rules for user modeling is the detection of pages that are accessed together, or predicting user’s intention. Association rules have the disadvantage of omitting the temporal (when an action is going to happen) and sequential (when A happens, B follows) aspects.

Supervised learning

The main supervised learning techniques are decision trees, classification rules, k-NN and support vector machines. The paragraphs explain each of these techniques, describing its applications, advantages and disadvantages.

- **Decision trees and classification rules:** Decision trees are used in order to implement classify users and documents. They have the advantage of being able
to handle imprecise and noisy data. Classification rules are an alternative representation of decision trees and as such are also used in order to classify users and data. They have the advantage of having a human-readable representation. Both approaches have the limitation of being highly dependant on the quality of the available information. Decision trees are also inefficient when dealing with high dimensional data and hard to manually modify when necessary.

- **k-NN**: k-NN is a technique used for classification in which the training data represents the model. In user modeling k-NN algorithms are used for recommendation based on the users past behavior. For example, the recommendation of interesting documents. The main disadvantage of this approach is the response time and the need to choose some values for metrics, as experimentation and knowledge about the system is needed.

- **Support vector machines**: Support vector machines (SVM) try to find an hyperplane which divides a set of datapoints into two classes. It is a classifier derived from statistical learning theory. They have the advantage of being able to work with high dimensional data. In the area of user modeling they are used for classifying information. So far, it has not been used to classify users [19].

**Neural networks**

Neural networks imitate the way biological nervous systems function. They can be used for supervised and unsupervised learning problems. Neural network consist of neurons, weighted interconnections, activation rules and learning rules in order to adjust the weight of the interconnections. Neural Networks have the advantage that no metrics need to be calculated (opposed to clustering, where the *distance* needs to be found). As a consequence no initial knowledge about the application is required. Neural networks are usually applied in classification and recommendation tasks. They have the inconvenience that a large amount of information and training time is needed in order to create the learnt model. Unlike previously discussed approaches, the models generated from neural networks cannot be interpreted and therefore no human readable model can be generated.

**Comparison**

The determination of an appropriate machine learning technique for context processing depends on the desired task (e.g. classification or recommendation), the amount of data that needs to be handled, the availability of labeled/unlabeled data, the need of generated (by the machine learning technique) human readable models, and the need for dynamic modeling support (models that change over time cause the learning algorithm to retrain
Tables 3.4 and Table 3.5 can be used as guides when deciding what type of machine learning technique to use. Table 3.4 shows a summary of the appropriate machine learning technique, based on the desired task: recommendation or classification, the need for a human readable model and the availability of labeled/unlabeled data. Table 3.5 shows a comparison of the presented machine learning techniques based on their complexity (an indicator of scalability), support for dynamic modeling, need for labeled/unlabeled data and generation of human readable models.

### 3.4 User Information Processing

User models can be acquired in an implicit or explicit way. Explicit acquisition refers to the process where the model is formed on hand of information provided by an external source, which can be the user itself or an administrator of the application. Implicit information acquisition refers to the process where the information provided by the external source is processed in order to be added to the user model [40]. For example, raw user interactions provide information of little use for the elaboration of a user model, but when grouped into a sequence of interactions, they can be useful for determining if a user is an expert or novice. The tasks of grouping the sequence of interactions and determining the user’s level of expertise are considered a part of processing. The processing of information for the creation of user models can be divided into three main types: rules of inference, stereotypes and machine learning techniques. In this section we will briefly describe each of these methods.

**Rules of inference**  Rules can be used to make inferences about user behavior and knowledge. For example, if a user is familiar with an action it can be assumed that
<table>
<thead>
<tr>
<th></th>
<th>Off-Line Complexity</th>
<th>Dynamic Modeling</th>
<th>Labeled/Unlabeled</th>
<th>Readability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clustering</strong></td>
<td>Depends on clustering techniques. See [19]</td>
<td>No</td>
<td>Unlabeled</td>
<td>No (SOM, K-means)</td>
</tr>
<tr>
<td><strong>Association Rules</strong></td>
<td>NP complete exponential with number of items</td>
<td>No</td>
<td>n.a.</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Decision Trees</strong></td>
<td>for single attribute, multi-way splits on $A$ discrete variables and data size of $N$: $O(A^2N)$</td>
<td>Yes</td>
<td>Labeled</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Classification Rules</strong></td>
<td>Same as decision trees + Rule generation</td>
<td>Yes</td>
<td>Labeled</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>k-NN</strong></td>
<td>$O(t \cdot c)$ with $c$ number of clusters $t$ number of training samples</td>
<td>Yes</td>
<td>Labeled</td>
<td>No</td>
</tr>
<tr>
<td><strong>Neural Networks</strong></td>
<td>NP complete for 3 layers Polynomial for 2 layers</td>
<td>Yes</td>
<td>Both</td>
<td>No</td>
</tr>
<tr>
<td><strong>SVM</strong></td>
<td>Highly dependant of the kernel used</td>
<td>No</td>
<td>Labeled</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.5: General characteristics of the machine learning techniques [19].
the user knows about the preconditions and effects of the action. KNOME [12] uses this approach. It catalogs the level of expertise and knowledge a user possesses based on user’s actions and user’s answers to questions from the system.

**Stereotypes**  
Stereotypes are clusters of characteristics [46]. In the area of user modeling they could be considered as sets of characteristics that can be assigned to a determinate type of users. Stereotypes reduce the overhead of modeling each individual user. When a user is assigned to a certain stereotype, it then inherits the characteristics of that stereotype. This enables applications to acquire implicit information about users [40]. In order for a user to be assigned to a certain stereotype, the user needs to activate certain triggers or events. For example, when the user types the a complicated command on the shell it is designated as an expert shell user. The GRUNDY system [46] uses stereotypes in order to initialise and maintain user’s models in the system. Stereotypes are activated by the execution of certain user events. Each stereotype consists on a set of tuples of attribute and value. For example if a user describes herself as a doctor, a stereotype which includes characteristics such as well educated, fairly affluent and well respected in the community are triggered.

**Machine Learning**  
The machine learning techniques mentioned in Section 3.3.2 can be used in order to learn about user characteristics, preferences and level of expertise.
4 Context-aware Frameworks

In the following we summarize existing context-aware frameworks. We focus on the used approaches for context elicitation, modeling and processing, as well as the mechanisms used in order to assure security and privacy.

CoBra

CoBra [10] possesses an agent based architecture. A context broker is in charge of managing the contextual model which is used by the agents. The context broker has 4 main components: Context Knowledge Base, Context Inference Engine, Context Acquisition Module and the Privacy Management Module. It uses reusable sensor units called context acquisition components in order to make sensing transparent to the agents. CoBra uses an OWL ontology-based approach, COBRA-ONT[10], in order to model context. Context processing is possible through the F-OWL language, which is a Prolog-based language implemented through Flora-2 [69]. F-OWL allows for the detection and solution of inconsistencies in the ontology and for the interpretation of sensing inputs. In order to do this the context broker is in charge of providing the additional rules to reason about the domain-specific knowledge. CoBra uses the Rei policy language [32] to express security policies about contextual information. Rei is modelled through the concepts of rights, prohibitions, obligations and dispensations. It controls access to context information through modifiable domain dependent rules.

Context Management Framework

The Context Management Framework [35] has a centralized architecture in which a context manager stores context data and provides this data to client applications through a blackboard. Similarly to CoBra it uses reusable sensor units called resource servers. This approach uses RDF-based ontologies as its context model. In order to quantify the information obtained from the sensors crisp limits and fuzzy sets are used. Quantization allows to assign sensed values to ontological attributes (e.g. setting a -10 Centigrade temperature to low). A naive Bayes classifier is used in order to recognize higher-level contexts from the lower-level context atoms obtained after quantization. The Context Management Framework contains a security module, but information about its implementation
is not available.

**Context Toolkit**

The Context Toolkit [50] has a centralised module where distributed sensors, interpreters and aggregators are registered in order to be discovered and used by client applications. This framework makes use of *context widgets* in order to reuse and encapsulate the sensing of different types of information. The framework uses XML coded value-attribute tuples as its context model. It allows for the aggregation and interpretation of information. The context aggregators are responsible for creating the context of an entity by subscribing to the necessary widgets. Context interpreters are responsible for transforming context, e.g., returning the corresponding e-mail address to a given name. In order to assure the protection of user privacy the Context Toolkit introduces the *context ownership* concept. Users are assigned as owners of their context data and can control through rules (based on situations, e.g., authentication and time) which other users can access their information. In order for context ownership to work, records of which context data is owned by which user are required.

**Hydrogen**

Hydrogen [27] tries to avoid taking a centralized approach by making a distinction between *remote* and *local* context. The local context is the information that a device knows about and that was acquired by itself, whereas the remote context information was acquired from another device which was in physical proximity in a peer-to-peer matter, such as WLAN and Bluetooth. Hydrogen uses an object-oriented approach to modeling context. The superclass ContextObject encapsulates different types of context e.g. LocationContext, UserContext, NetworkContext, etc. Each of these context types has methods to convert data from and to an XML stream. In terms of context processing only raw data is aggregated and interpreted.

**SOCAM**

SOCAM [24] was designed to enhance the development and rapid prototyping of context-aware mobile services. It uses a centralized server in charge of processing the context data, which is made available by distributed context providers. Context-aware mobile applications use the information given by the centralized server in order to adapt their behavior. SOCAM uses *context providers* in order to make the sensing transparent to the consumer and reusable. SOCAM uses OWL ontologies as its context model. They divide their application focus, pervasive computing, into several subdomains, e.g., home domain, office domain, etc. and define an ontology for each subdomain. Having an ontology for
each sub-domain helps reduce the complexity of context processing. Context processing is
done by the Context Reasoning Engine which is responsible of inferring deduced contexts,
maintaining the consistency of the context model and resolving context conflicts. The
interpreter is built with help of the semantic web toolkit Jena\(^1\).

**WildCAT**

WildCAT [15] is an extensible Java framework which provides underlying mechanism for
realizing context-aware applications by providing APIs to discover, interpret and monitor
the events occurring in an execution context. It provides a dynamic data model to represen-
t the execution context and offers access to context information through synchronous
(pull) or asynchronous (push) requests. The data model representing the execution con-
text considers context to be made of several domains, which separates different aspects of
the execution context. Examples of context domains include: local hardware resources,
network performance and topology, geophysical information, user preferences and charac-
teristics.

WildCAT does not implement any of the mentioned context domains but provide inter-
faces to realize them and means to share the gathered data across different applications.
Finally each context domains are modeled as an XML tree of named resources. Each
resource being described by key-value pairs. The context monitoring and processing is an
event driven mechanism and records every single change occurring on the context model.

The WildCAT framework consists of several sub-frameworks with there underlying roles
in the overall architecture. Among all the data acquisition framework is the one which
is built around the sensors which monitor the execution environment. The framework
follows the notion of active and passive sensors. It does not provide means to gather the
data (sensors itself) but rather provides methods to monitor the sensors, at a slightly
higher level. WildCAT possesses the ability to reason about context but no details of how
this is done are given.

**Construct**

Construct [16] is framework for pervasive computing. It uses distributed modules, called
nodes, in order to discover context providers, elicit information, manage context models
and process information. Nodes communicate with each other through the Zeroconf
protocol [58] in order to update their models. Context information is stored in RDF triples
and can be retrieved with the SPARQL query language. Information is aggregated and
inferred through the Jena framework. Due to their non-centralized approach Construct
is robust to individual node failure, However the communication between nodes can lead

\(^1\)http://jena.sourceforge.net/
to substantial communication traffic.

**Comparison**

Table 4.1 summarizes each of the previously mentioned context-aware frameworks, while taking into consideration the most important activities for context awareness: context elicitation, context modeling, and context processing. While each framework uses different elicitation mechanisms, it is important to note that all of them encapsulate the elicitation process and provide access to contextual data through well defined interfaces. The election of the context modeling approach has direct impact on the processing techniques that can be applied. As mentioned in Section 3.2.4 ontologies allow for the inclusion of semantic meanings, enable knowledge sharing and allow for the incorporation of reasoners, with which inference and aggregation are possible. Because the Context Toolkit uses attribute-value tuples as its context model, it is in a disadvantage in comparison to the other frameworks, as information can not be inferred and no semantic meanings are available. Concerning security and privacy, the frameworks which include mechanisms to protect the users information are CoBra, the Context Toolkit and the Context Management Framework.
<table>
<thead>
<tr>
<th>Framework</th>
<th>Context Elicitation</th>
<th>Context Modeling</th>
<th>Context Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoBra</td>
<td>Context acquisition module</td>
<td>OWL ontologies</td>
<td>Interpretation: ontologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation: Flora-2 inference engine</td>
</tr>
<tr>
<td>Context Management Framework</td>
<td>Resource servers</td>
<td>RDF ontologies</td>
<td>Interpretation: ontologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation: naive Bayes classifiers</td>
</tr>
<tr>
<td>Context Toolkit</td>
<td>Context widgets</td>
<td>Attribute-value tuples</td>
<td>Interpretation: tuple search</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation: tuple search</td>
</tr>
<tr>
<td>Hydrogen</td>
<td>Adapters for various context types</td>
<td>Object-oriented</td>
<td>Interpretation: object model, only on raw data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation: object model, only on raw data</td>
</tr>
<tr>
<td>SOCAM</td>
<td>Context providers</td>
<td>OWL ontologies</td>
<td>Interpretation: ontologies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aggregation: Jena2 semantic web toolkit</td>
</tr>
<tr>
<td>WildCAT</td>
<td>Provide no sensors, just interfaces</td>
<td>XML markup scheme</td>
<td>Implemented. No explanation.</td>
</tr>
<tr>
<td>Construct</td>
<td>Sensors</td>
<td>RDF triples</td>
<td>Aggregation and Inference: Jena semantic web toolkit</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of context-aware frameworks.
5 Context-aware Applications for Software Maintenance

Software maintenance is a task that involves the integration, abstraction, and analysis of different knowledge resources and artifacts. Context-aware applications used in software maintenance focus on adapting or reconfiguring applications with varying context and on enabling direct access of the applications to context information [54]. Other platforms use context information in order to benefit from knowledge sharing.

Several EU context-aware projects in the scope of software maintenance and self-adaptive applications, are related to mobile applications, due to the fact that this domain deals with issues that require configuration, operations, maintenance and change management. The current range of mobile platforms and the variety of infrastructures makes the maintenance of the applications more difficult.

Mylyn

Mylyn [34] tries to reduce the information overload in IDEs by filtering source code elements inside development environments, which are irrelevant for the current task. It highlights and blurs out some of the artifacts in large software projects that appear to be less relevant for the current task of the developer. The core hypothesis of Mylyn is that the artifacts in a large software project differ in their relevance for the actual developer in a given context. Mylyn tackles this by analyzing two dimensions of context. First, it assigns interest values for each artifact in the IDE, based on the developer’s interactions with the IDE (i.e., when clicking or editing artifacts). Those values decay over time. Second, Mylyn computes a degree-of-separation model among all artifacts, which is used to estimate the interest of related items. The degree-of-interest model created this way can then be used to modify or filter the appearance of information inside the IDE. One of Mylyn’s design goals is to ensure predictability and transparency. For identifying task context, its creators for the traceable approach of recording actual user-driven events. Furthermore, all filter mechanisms offered by Mylyn (regarding the Eclipse package explorer, outline view and task list) can be easily controlled and switched off by the user. Since Mylyn monitors the developers’ interactions with the IDE, it provides a view of those usage statistics to reveal the captured information. Those statistics, as well
as all information regarding context and tasks are available in a dedicated API. Mylyn is currently developed as an Eclipse technology project\(^1\).

**MUSIC**

MUSIC\(^2\), acronym of self-adapting applications for Mobile Users In ubiquitous Computing environments, is funded in the Sixth Framework Programme. Its goal is to develop an open source platform for the implementation of self-adapting, reconfigurable software that adapts to the dynamic user and to the execution context. Context awareness is one of the most important ingredients for achieving the ubiquitous computing paradigm as it gives information about the changing environment and users. MUSIC allows self-configuring the applications with context information in an indirect and autonomous way.

As part of the MUSIC architecture, one of the most remarkable components is the Context Middleware, which is responsible for collecting, organizing, managing and sharing context information. With the help of this middleware, context providers and context consumers can be plugged at runtime. This feature is enriched with a context distribution mechanism based on the Session Initiation Protocol (SIP)\(^3\). This protocol provides instant messaging and presence functionality and also be used for sharing context information in a SIP-enabled network.

MUSIC can adapt its behavior, using both parameter and compositional adaptation. In the case of parameter adaptation, a set of variables is defined so that they can be dynamically modified at runtime. On the other side, compositional adaptation enables the structural and algorithmic modification of a system, thus enabling, for example, the introduction of new behavior after deployment or the correction of software without having to bring the system to a halt.

**MIDAS**

MIDAS [61, 23] was funded under the Sixth Framework Programme, with the aim of providing a context-aware platform that simplifies and speeds up the creation of developing and deploying mobile services and applications. It is based in the provision of middleware building blocks, for critical services in situations where time and infrastructure availability are often critical factors, like emergency situations or major events. This middleware is used to help distribute applications to mobile devices, and to configure and update them before and during operation.

MIDAS achieves this with the use of middleware building blocks. Services are realized by distributed software components running on nodes owned by the service provider.

\(^{1}\)http://tasktop.com/mylyn/

\(^{2}\)http://www.ist-music.eu/

\(^{3}\)http://tools.ietf.org/html/rfc3261
and on devices operated by the end-users. MIDAS middleware components are installed on all of these nodes. The middleware realizes a Distributed Data Management System (DDMS). All service functionality will be realized by entering, retrieving or responding to changes in data stored in the DDMS, which contains all the context-aware information (location, access-rights, bandwidth). Nodes maintain the DDMS by asynchronously exchanging short messages. These are exchanged using one or more communications mechanisms, depending on what nodes are present and the available communications. Service developers do not need to access low-level functionality provided in specific mobile networks. MIDAS middleware automatically adapts to changes in network topology. This allows to compensate for problems (e.g. failure of particular links).

**Collaborative tools**

Other research projects, e.g. CodeLink, and PATTERNS, have put their efforts in developing context-aware applications that can facilitate the tasks of software maintenance, allowing developers to explore analogies between a current working situation and past experiences, and using them in solving a new problem by recognizing its similarity with a specific known problem.

**CodeLink**

CodeLink [70] is a prototype of the Department of Mathematics and Computer Science of the Drexel University (Philadelphia, USA). The goal of CodeLink is to automatically extract the software development context associated with email-based project collaboration. This application enriches the communication within engineering teams and builds repositories that detail collaborative decisions made in the development and maintenance processes. These repositories can then be used to make software maintenance easier by enabling context-aware email collaboration among software developers. It structures the information provided in the email messages interchange between the members of software development teams. Context information can be searched in a variety of ways to improve the software development, management and maintenance process.

**PATTERNS**

PATTERNS\(^4\), with the same purpose of CodeLink, has the following slogan: Patterns to adopt knowledge-based solutions to software management. This European Project of the Fifth Framework Programme tries to provide an inter-networked software application that allows users to obtain a rapid solution to a context-based problem. This solution

\(^4\)http://www.esi.es/Pat terns
should be provided according to the knowledge available locally at each Knowledge Centre available in the network.

In order to achieve this purpose, PATTERNS perform the modelling of management practices of the Knowledge Centres domains, to determine common characteristics, the rules to be applied and the action to be taken. It also defines a learning and dynamically adaptive architecture that allows users to explore analogies between a working situation and past experiences, and to use them in solving a new problem by recognizing its similarity with a specific known problem. Finally, as a result it creates an inter-networked structure supported by intelligent agents, to spread the learning process across different organizations.

By means of the Context Extractor, the relevant source code information is encoded with a message ontology based on the DARPA Agent Markup Language (DAML) and it is inserted into the email message as a MIME attachment of type DAML/code-link. At the same time, a snapshot of the source file is taken and sent to the CVS access control module using CVSPUT request. All the email messages and the code-link attachments are archived in a repository, which is finally accessed using a web browsing/search interface. In that sense, the search of archived messages can be performed based not only on text parsing, but also on context.

**Comparison**

Context-aware applications used in software maintenance include collaborative software engineering tools, e.g. CodeLink and Patterns, and context awareness frameworks. Collaborative software engineering tools are considered part of software maintenance because a better collaboration among software maintainers enhances the maintenance process. Example of context awareness frameworks are MUSIC and MIDAS. Both have focused on developing architectures that can make the acquisition of context information easier, and that can provide agile software maintenance with only reconfiguring some parameters. While MUSIC provides context information with a wide range of applications (any context consumer), MIDAS is focused on ensuring the proper configuration of services according to the context changes, by means of a Distributed Data Management System. Mylyn provides an example of how context information can help in the software maintenance process by eliminating unnecessary information, which is irrelevant to the maintenance task.
6 Conclusion

This state of the art summarizes the most important approaches of context-aware activities: context elicitation, context modeling and context processing. In this work, context-aware frameworks and applications are described and analyzed according to the approaches they use for eliciting, modeling and processing information. This section analyzes the most important findings and open research questions for each context-aware activity.

Context Elicitation

The survey on application execution elicitation showed that in order to have a complete replay of the application execution (in order to have more information about the error) all information concerning non-deterministic operations has to be collected. It was found that this can be done through library logging. The elicitation of context information of application running on Virtual Machines is easier, as only the bytecode that triggers system calls and the order of the execution threads has to be recorded.

In the survey regarding user interaction elicitation no system or framework was found which could monitor any type of application. The question of the feasibility of monitoring user interaction independently of the type of application and operating system remains an open question.

Context Modeling

The evaluation of different modeling approaches summarized in Table 3.3 shows that ontologies are the modeling approach that fulfills to the most complete extent the defined criteria of: partial validation, representation of rich information, management of incomplete and ambiguous information and level of formality. Through the summary of different context-aware frameworks in Section 4 it can be concluded that the type of context modeling approach heavily affects the type of semantic-based approaches that can be used. Ontologies using logic-based languages have the advantage of enabling an easy incorporation to inference engines. Important challenges when modeling context is the representation of large amount of information (scalability), the representation of ambiguous and incomplete information, as well as the quantification of human factors, e.g.
Experience.

**Context Processing**

Semantic-based approaches are based on rules and inference engines, i.e. reasoners. These approaches present the challenge that manual rules can be difficult to maintain and rule-mining may be needed.

The survey of different machine learning approaches showed that the choice of the adequate approach heavily depends on the type of task (e.g. classification, recommendation, etc.), the availability of labeled/unlabeled data, the amount of data and the need for dynamic and human understandable models. Some challenges for context processing with machine learning are: need for large data sets, need for labeled data, concept drift and computational complexity [66].
Bibliography


