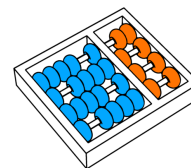Felipe Henriques da Silva

# "Serial Annotator: Managing annotations of time series."

# "*Serial Annotator: Gerenciando anotações em séries temporais.*"

**CAMPINAS**

**2013**

i

University of Campinas
Institute of Computing

*Universidade Estadual de Campinas*
*Instituto de Computação*

## Felipe Henriques da Silva

## "Serial Annotator: Managing annotations of time series."

Supervisor:
*Orientador(a):*  **Prof.ª Dr.ª Claudia Maria Bauzer Medeiros**

## *"Serial Annotator: Gerenciando anotações em séries temporais."*

MSc Dissertation presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a Mestre degree in Computer Science.

*Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.*

THIS VOLUME CORRESPONDS TO THE FINAL VERSION OF THE DISSERTATION DEFENDED BY FELIPE HENRIQUES DA SILVA, UNDER THE SUPERVISION OF PROF.ᴬ DR.ᴬ CLAUDIA MARIA BAUZER MEDEIROS.

*ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA POR FELIPE HENRIQUES DA SILVA, SOB ORIENTAÇÃO DE PROF.ᴬ DR.ᴬ CLAUDIA MARIA BAUZER MEDEIROS.*

_____

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2013

iii

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Informações para Biblioteca Digital

**Título em inglês:** Serial Annotator : managing annotations of time series
**Palavras-chave em inglês:**
Databases
Time-series analysis
**Área de concentração:** Ciência da Computação
**Titulação:** Mestre em Ciência da Computação
**Banca examinadora:**
Claudia Maria Bauzer Medeiros [Orientador]
Renato Fileto
Luiz Fernando Bittencourt
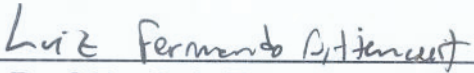**Data de defesa:** 10-06-2013
**Programa de Pós-Graduação:** Ciência da Computação

# TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 10 de Junho de 2013, pela
Banca examinadora composta pelos Professores Doutores:

**Prof. Dr. Renato Fileto**
**INE / UFSC**

**Prof. Dr. Luiz Fernando Bittencourt**
**IC / UNICAMP**

**Profª. Drª. Claudia Maria Bauzer Medeiros**
**IC / UNICAMP**

# Serial Annotator: Managing annotations of time series.

## Felipe Henriques da Silva

June 10, 2013

**Examiner Board/*Banca Examinadora*:**

- Prof.ª Dr.ª Claudia Maria Bauzer Medeiros (Supervisor/*Orientadora*)

- Prof. Dr. Luiz Fernando Bittencourt
  Institute of Computing - UNICAMP

- Prof. Dr. Renato Fileto
  Department of Informatics and Statistics - UFSC

- Prof.ª Dr.ª Maria Cecilia Calani Baranauskas
  Institute of Computing - UNICAMP (Substitute/*Suplente*)

- Dr.ª Carla Geovana do Nascimento Macario
  CNPTIA - EMBRAPA (Substitute/*Suplente*)

# Abstract

Time series are sequences of values measured at successive time instants. They are used in several domains such as agriculture, medicine and economics. The analysis of these series is of utmost importance, providing experts the ability to identify trends and forecast possible scenarios. In order to facilitate their analyses, experts often associate annotations with time series. Such annotations can also be used to correlate distinct series, or look for specific series in a database. There are many challenges involved in managing annotations - from finding proper structures to associate them with series, to organizing and retrieving series based on annotations. This work contributes to the work in management of time series. Its main contributions are the design and development of a framework for the management of multiple annotations associated with one or multiple time series in a database. The framework also provides means for annotation versioning, so that previous states of an annotation are never lost. Serial Annotator is an application implemented for the Android smart phone platform. It has been used to validate the proposed framework and has been tested with real data involving agriculture problems.

# Resumo

Séries temporais são sequências de valores medidos em sucessivos instantes de tempo. Elas são usadas em diversos domínios, tais como agricultura, medicina e economia. A análise dessas séries é de extrema importância, fornecendo a especialistas a capacidade de identificar tendências e prever possíveis cenários. A fim de facilitar sua análise, especialistas frequentemente associam anotações com séries temporais. Tais anotações também podem ser usadas para correlacionar séries distintas, ou para procurar por séries específicas num banco de dados. Existem muitos desafios envolvidos no gerenciamento destas anotações - desde encontrar estruturas adequadas para associá-las com as séries, até organizar e recuperar séries através das anotações associadas a estas. Este trabalho contribui para o trabalho em gerenciamento de séries temporais. Suas principais contribuições são o projeto e desenvolvimento de um arcabouço para o gerenciamento de múltiplas anotações associadas com uma ou mais séries em um banco de dados. Este arcabouço também fornece meios para o controle de versão das anotações, de modo que os estados anteriores de uma anotação nunca sejam perdidos. *Serial Annotator* é uma aplicação desenvolvida para a plataforma Android. Ela foi usada para validar o arcabouço proposto e foi testada com dados reais envolvendo problemas do domínio agrícola.

# Acknowledgements

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction and Motivation

Time series are sequences of observations of an object along time. Such series are used in various domains of knowledge. They can be used to represent, for instance, the amount of rainfall measured by a sensor, the heart rate measured on electrocardiograms, or the value of shares in the stock exchange. The graphical representation of a time series (of a single numeric measure) usually has the observation timestamps on the x axis and the measured values on the y axis. Figure 1.1 provides some time series examples.
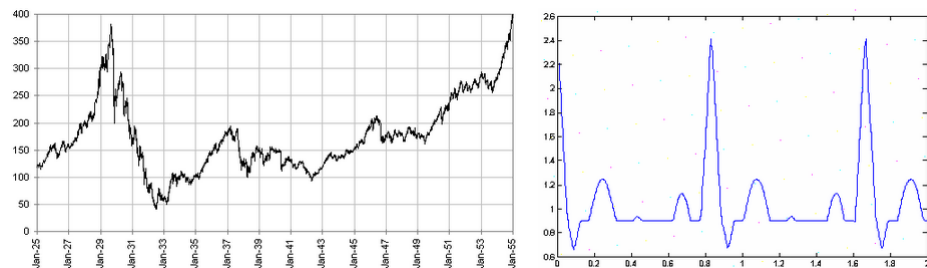


(a) Dow Jones market price - reproduced from [23]



(b) Electrocardiogram - reproduced from [24]
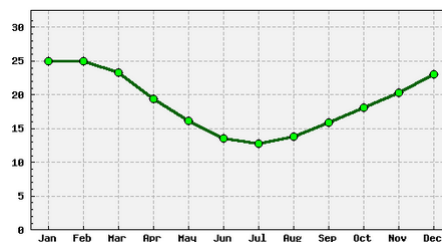


(c) Temperature variation in Mornington (Australia) - reproduced from [25]

Figure 1.1: Time series examples

1

Time series are often produced as continuous streams and therefore their storage in databases presents some challenges, given the need to store large volume of data and to handle frequent updates. Furthermore, due to their nature, queries over such databases are not based on an exact match, but on the similarity among the series in question [11].

In the last decade, several studies have addressed time series data analysis. Examples include work involving series mining e.g., search for similar series, search for patterns within the series, search for subsequences. Such work often requires dimensionality reduction and segmentation [11]. However, as efficient as these solutions may be, they do not completely solve the problems of analysis and interpretation of the resulting series, which are extremely complex and specific to the domain in question. One solution to alleviate this problem is to associate annotations with series (e.g., [20]).

Annotations are a type of metadata, or data about data. They are used to provide further information on data that may be relevant for its analysis. Although annotations are usually represented in textual form, they may also be represented with other media such as audio or images. Figure 1.2 provides an example of an annotated electrocardiogram. If annotations can be efficiently stored and associated with series, then series management and retrieval can be improved. In the example, doctors looking for a particular heart behavior in electrocardiograms can combine series mining with queries on annotations.
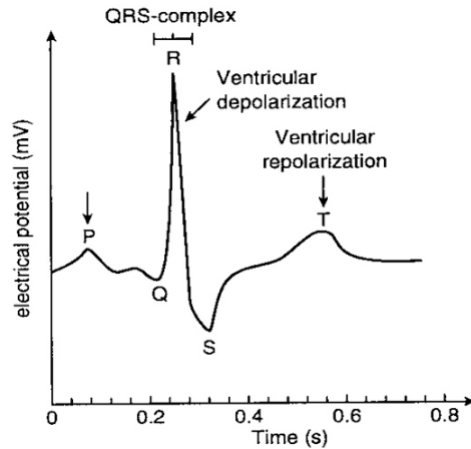


Figure 1.2: Annotated electrocardiogram example - reproduced from [24]

The same analysis difficulties can be observed in the agriculture domain, as shown in figure 1.3. This figure shows an example of a situation, which we will subsequently use in our case study. The y axis corresponds to the variation of NDVI (Normalized Difference Vegetation Index) values. Roughly speaking, NDVI is a numerical value that indicates the

"greenness" of a region [1]. The time series in the figure is annotated with many kinds of information. Note that without the annotations it would be very difficult for a non-expert to acquire useful information from this time series.
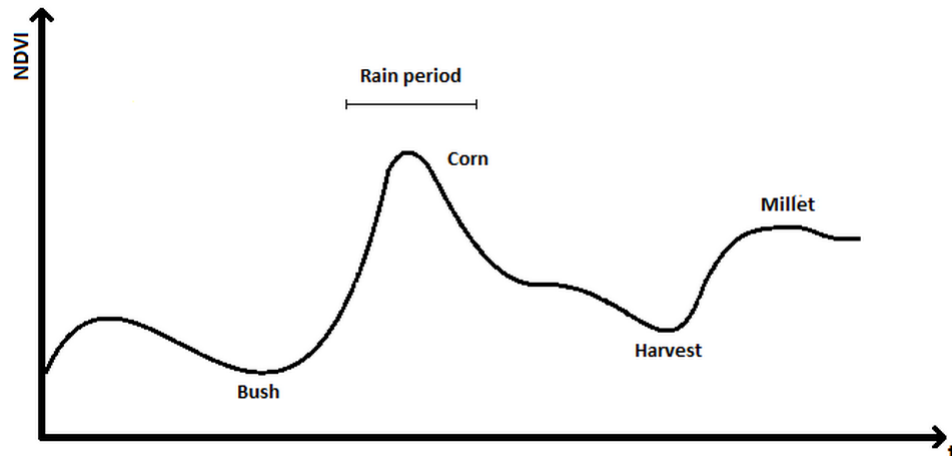


Figure 1.3: Annotated time series - agriculture domain

Up to now, related work has not dealt directly with the management of annotations associated with time series, although several papers deal with the storage of annotations in a relational database (e.g., [9, 2]). Furthermore, there is also related work dealing with annotation processing, based on the concept of one annotation per time series [20]. This dissertation investigates issues related with annotations and their management. As a result, a database-centered framework that supports creation and management of annotations for use on multiple series (for both static series and streams) has been designed and implemented. This dissertation deals with textual annotations only, which still represent the great majority of annotations of time series.

The series analyzed and used on the validation of this work originate from the agricultural domain and were provided by experts from EMBRAPA[2]. Agriculture is of extreme importance for any country, and in Brazil it was responsible for 22% of the GNP in 2011 [13]. Time series are continuously generated by agricultural data sensors and satellites. The proper analysis of such series and their annotations will help experts in their research. Even though the main focus of this work is the agricultural domain, the proposal is generic enough to be applied to other domains, as long as textual annotations are used.

The main contributions of this work are the following:

---

[1] For more details on NDVI, see chapter 4

[2] Brazilian Agricultural Research Corporation

1. The design and implementation of a framework to support the management of multiple annotations associated with one or more series, that allows insertion, deletion, update and versioning of such annotations.

2. Validation of this framework via a smart phone application, tested with real data and annotations. This application deals seamlessly with both historical and stream data, which can be directly input via, e.g., wireless communication network.

The rest of this work is organized as follows: Chapter 2 presents the basic concepts and reviews related work describing the state of the art with respect to annotation management in relational databases. Chapter 3 presents the proposed framework, highlighting the database model and algorithms. Chapter 4 describes implementation aspects, presents the smart phone application and discusses the framework validation using real data. Chapter 5 presents conclusions and future work possibilities.

# Chapter 2

# Basic Concepts and Related work

This work focuses on the management of annotations associated with time series stored in a database. Section 2.1 gives an overview of some research lines dealing with time series. Section 2.2 broadens the concept of an annotation and reviews related work. Section 2.3 gives an overview of work related to annotations stored in a relational database. At last, section 2.4 gives a brief overview of temporal databases.

## 2.1 Time Series

A time series can be formally defined as a sequence of tuples $< v_i, t_i >$, where $v_i$ is the value of some variable measured at timestamp $t_i$. This definition can be extended to arbitrary objects, forming tuples $< S_i, t_i >$, where $S_i$ is the state of the object at timestamp $t_i$.

The usage of temporal data, and in particular, time series, has increased over time, leading to several kinds of research in the field of time series data analysis. However, because of their numerical and continuous nature, time series analysis and query processing over these series are complex subjects. In [17, 18], Lin et al. describe areas in which time series research has concentrated:

- Indexing: Design index structures to speed up similarity search.

- Grouping: Find natural groups within the time series in a database, given some similarity measure.

- Classification: Given a time series Q, classify it according to predefined classes.

- Summarization: Given a time series Q, create a description (textual or graphical) that retains all its characteristics but is concise enough to fit in a single presentation screen or page.

- Anomaly detection: Given a time series Q, and a model of what should be a "normal" behavior, find sections of Q that contain anomalies (also called surprising, interesting, or unexpected patterns).

Note that these topics are concerned with time series data mining, in which the majority of research concentrates on pattern searching [17, 21, 22, 7]. A detailed survey on time series data mining can also be found in [11].

As will be seen, this dissertation is not concerned with research in time series themselves. Rather, its focus is on the management of annotations of series, to enhance their interpretation, comparison and retrieval. Thus, this section aims at giving only a brief overview on the concept of time series and on research in this area.

## 2.2  Annotations

To annotate means to attach data to some other piece of data [27] - similar to metadata. Annotations describe a resource (digital or not) considering its characteristics. An annotation has many purposes. It may be used to explain something, provide additional information, or improve information retrieval. Annotations are also often used to describe characteristics that are hard to be observed using the media format of the annotated object. An address or the name of a building, for instance, cannot be derived from a picture of the building. Another frequent use for annotations is to help information exchange among experts, or to attach semantics to objects. To accommodate all these different purposes, several formats of annotations have been proposed in the literature. Annotations may be, for instance, in form of text, voice comments, videos or images. Figure 2.1 shows a tool used to associate drawing annotations with the image of a vehicle. A survey and comparison on this and several other tools used to annotate different media formats can be found in [31]. This dissertation concentrates on textual annotations.

A more formal definition for an annotation has been proposed by Euzenat in [10], where an annotation can be viewed as a function that relates a document to its formal representation, enabling the interpretation of document content. Using such an annotation schema, it would be possible to reconstruct the annotated content, assuming that background knowledge is available and a formal terminology (e.g., ontology) is used on the annotations.

Besides specifying this formal annotation concept, Euzenat [10] also indicates that without clear guidelines, annotations risk producing incoherent information. In order to avoid this problem, the author recommends answering a set of questions before the annotation process begins, so that the expert can create annotations in close relation to their use.

(a) Drawing with a pen (b) Visualizing the annotation
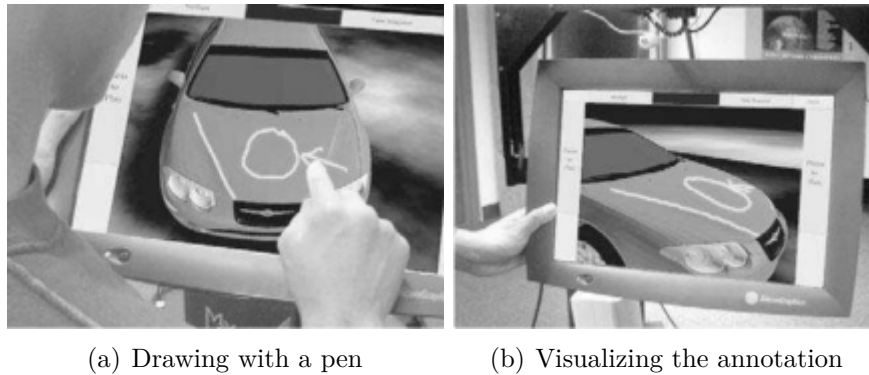
Figure 2.1: Annotating an image - reproduced from [31]

The usage of formal annotations is necessary for the concept of Semantic Web introduced by Berners-Lee in [3]. Here, annotations must be interpreted not by humans, but by machines, which must be capable to analyze the annotations and infer useful information. To achieve this, annotations must use controlled vocabularies or metadata fields from ontologies [36]. These so-called semantic annotations (as opposed to free text annotations) allow for greater interoperability, since they may be interpreted by any computational system that known the controlled vocabulary.

In order to associate an annotation with Web content, there are several free-text annotation systems that may be used. Some of these systems use schemas based on the Annotea annotation schema [33, 15]. Annotea is a Web-based shared annotation system based on a general-purpose open Resource Description Framework (RDF) infrastructure. RDF is a W3C standard language for representing information about resources in the Web [29]. Annotea allows only simple free-text annotations.

An annotation can be created manually [16], semi-automatically [14], or automatically [8]. Another way to create annotations is through "crowdsourcing", that is, creating annotations based on contributions from a large group of people or from an online community. In [35] Wu et al. show that a global semantic model can be statistically inferred from informal annotations collected from web blogs and social bookmarks (called social annotations). These informal annotations can be any strings that the user deems appropriate for the web resource. The term folksonomy is coined to refer to these informal social tags and categories in social bookmarks. The authors also show that the semantics that emerge from folksonomies can then be used to search for semantically-related content, even if the content is not tagged by the query tags and does not contain any of the query keywords.

Another problem faced by annotation systems is the propagation of annotations during data processing. That is, if a system has an input data set $I$, where some items

have associated annotations, it is not straightforward to determine how these annotations should be propagated to the output set $O$. A solution to this problem was proposed by Amiguet-Vercher et al. [1], where the proper mapping from input to output annotations is described as a clustering problem.

As already mentioned, in this work, annotations are considered to be a textual representation of the annotated content. They may provide further information, clarify the object being annotated or provide some kind of communication among experts analyzing the same object.

This section covered the importance of the annotation process and the variety of possibilities for annotating content. The next section will cover work related to textual annotations associated with entries in relational databases.

## 2.3 Annotations in relational databases

In the context of relational databases, annotation is an information linked to data items inside the database. Data can be annotated at multiple granularities, e.g., annotating an entire table, an entire column, a subset of the tuples, a few cells, or a combination of these. Figure 2.2 shows annotations with such multiple granularities. In this figure, for instance, annotation $A_1$ is associated with all cells in the entire first row while annotation $A_4$ is associated with all cells in the last two columns.



Figure 2.2: Annotations with multiple granularities - reproduced from [9]

Storing annotations in relational databases presents some architectural questions. The main ones are: where should the annotation be stored and how should it be linked to the annotated data.

Despite their importance, annotations are not supported by most database systems. In [4], Bhagwat et al. present an early study addressing annotation management, where a very simple schema was developed. In this schema, annotations are stored together

with the annotated data irrespective of the annotation granularity. This raises storage problems, as a single annotation must be replicated through all annotated space. If, for instance, an annotation is associated with all cells in a column, as many annotations as the number of lines in the annotated column will be created. Although very simple, this schema presents a first evolution on annotation storage and has been used as a benchmark for further annotation management schemas. It is also important to note that this naive storage schema facilitates annotation propagation through database operations.

A more sophisticated annotation management schema can be found in [9], where the problem of multiple granularity levels is better addressed. The authors propose that each annotation should be linked to the annotated cells through a mapped space. Annotations are also stored on separate tables called annotation tables that have a predefined structure. Figure 2.3 provides an example of an annotation table.

| Annotation Id | Curator | Timestamp | Annotation Value | Annotation CoveredCells |
|---|---|---|---|---|
| 10 | Admin | | A1 | ((1,1), (6,1)) |
| 20 | Admin | | A2 | ((1,2), (2,4)) |
| 30 | User | | A3 | ((2,3), (3,3)) |
| 30 | User | | A3 | ((5,3), (5,3)) |
| 40 | User | | A4 | ((5,1), (6,4)) |

Figure 2.3: Annotation table - reproduced from [9]

Note that annotation $A_1$ is associated with all cells in the first row, that is, the annotation extends from the first column in the first row (1,1) to the sixth column in the first row (6,1). In the same way, annotation $A_4$ is associated with all cells in the last two columns, that is, the annotation extends from the fifth column in the first row (5,1) to the sixth column in the fourth row (6,4).

Another work dealing with annotations in relational databases is that of Aoto et al. [2], in which the main focus is the propagation of the annotation when a database operation is performed. In order to correctly propagate annotations, the authors propose that instead of associating an annotation directly with the annotated data, annotations should be associated dynamically, by means of data correlations. Correlations are then recomputed after database operations are performed, applying the annotation on the new data. An example of such an annotation schema can be observed in figure 2.4, where the annotation "Discomfort" is associated with relations where the attribute Temperature has values larger then 30 or smaller then 15.

Figure 2.4: Annotation correlation

Table 2.1 presents a comparison among the previously presented papers, highlighting their characteristics. As will be seen, the framework proposed here to manage annotations associated with time series in a database is derived from combining features from these three papers.

| Work | Proposal | Storage | Versions | Multiple Annota-tions | Granu-larity | Provenance Information |
|---|---|---|---|---|---|---|
| Bhagwat et al. [4] | Stores annotations together with annotated data | With Data | No | No | No | No |
| Eltabakh et al. [9] | Manages multiple granularities in annotations by mapping annotated space | Separate | No | Yes | Yes | No |
| Aoto et al. [2] | Specifies correlations in order to propagate annotations | Separate | No | Depends on the correlation | Yes | No |

Table 2.1: Comparison among annotation storage schemas

Annotation storage has also a close relationship to data provenance storage. Provenance information describes the origins and the history of data in its life cycle [6]. Since there is no standard to store data on provenance, it can be stored in the form of annotations. However, work dealing with data provenance is focused in the "why", "where" and

"how" of every update of data [6, 5], which creates complex structures for provenance and makes it unsuitable for storage as textual annotations. Provenance data can also be used to enhance security in a system [28]. It would be possible, for instance, for a system to have a policy where only the author of an annotation can update it, or where the author cannot review his/her own annotations. In this work, data provenance can be related either to time series or to annotations. For time series, provenance is considered to be the information related to the originator of the series, that is, which sensor, satellite or entity created the series. For annotations, only the author name and annotation modification storage time (transaction time) are considered as provenance of the annotation themselves.

Annotations can be stored in many kinds of formats and systems besides relational databases. XML files, for instance, can store set-valued attributes and could be more suitable for annotation storage. XML has also the advantage of being more computer-processable, allowing for interoperability. RDF triples offer also another means to store annotations (e.g., [31]). However, there is an associated complexity to convert from/to XML/RDF file structures while performing database operations. For simplicity reasons, this work deals only with annotations stored in relational databases.

## 2.4 Temporal databases and database versioning

In a collaborative system, it is common for experts to modify annotations created by other experts. This means that the annotation content changes over time. Such modifications are usually performed due to errors on previous annotations or to changes in the understanding of the observed phenomenon. Usually, however, it is important for experts to analyze previous contents of an annotation as they may provide insights on the observed phenomenon along time. Thus, it is important to preserve old annotation contents in the database. As will be seen, this work preserves annotation history taking advantage of research in temporal databases.

The work of Snodgrass [30] presents an overview of the main concepts of temporal databases. Snodgrass defines four database types depending on the given temporal entries, which are: valid time (the time range when an entry is valid) and transaction time (the time when the information was stored):

1. **Snapshot database:** No temporal information stored.

2. **Rollback database:** All past states of the database are stored and indexed by the transaction time. Queries can be performed on any previous state of the database.

3. **Historical database:** Uses valid time, recording a single historical state per relation. While rollback databases can roll back to a previous snapshot relation, historical databases can represent current knowledge about the past [30].

4. **Temporal database**[1]**:** Combination of the previous two approaches, using both valid time and transaction time.

In the proposed framework, when an annotation is stored, the transaction time is stored with it. This leads to a rollback database approach, since it is possible to perform queries over previous annotations' states. However, annotations are seen as static labels, created by experts usually (in our case study) while they are working at external locations. While these annotations may be changed by other experts later on, this is not the best approach if experts want to collaborate over annotations. Managing this collaboration may be a complex task when multiple annotators and multiple sites are involved. To alleviate this problem, collaborative annotation frameworks have been proposed (e.g., [34, 19]). Such frameworks allow discussions among experts before the annotation is stored. Future extensions of this work shall consider using ideas from these frameworks to enhance collaboration among experts.

Last but not least, literature on temporal databases contemplates a third kind of time, user-defined time, in which users define their own time units based on application semantics - e.g., seasons of the year, specific holidays and so on. In agriculture, users can mark events according to activities - e.g., harvest, seeding and so on. This work does not consider these kinds of issues, although user-defined time values can also be treated as textual annotations.

## 2.5 Conclusions

This chapter presented the main concepts necessary to understand the framework proposed to manage annotations associated to time series. The next chapter presents the proposed framework.

---

[1]**The term Temporal database is used by Snodgrass in [30]. However, the combination of Historical and Rollback databases is also known as Bi-temporal database.**

# Chapter 3

# Framework for managing annotations of time series

This chapter presents the framework proposed to manage, in a database, annotations associated with time series. Section 3.1 describes the proposed method to associate annotations with time series and section 3.2 describes the data structure and database schema. The framework architecture is presented in section 3.3. Section 3.4 shows some query possibilities offered by the proposed framework.

## 3.1   Annotation storage

The proposed framework aims to solve two problems related to annotation storage:

- How to store multiple annotations associated with a time series.

- How to store annotations associated with multiple time series.

This section will go through the fundamentals behind the proposed methodology for annotation storage.

Whenever an expert annotates a section of a time series, (s)he is in fact annotating both an interval and a set of values the time series assumes in that interval. If the expert associates multiple annotations with a time series as illustrated in figure 3.1, then there is a set of annotated intervals and values. In this figure, annotation $A_1$ is associated with interval $[t_1, t_2]$ and values $[v_1, v_2]$, while annotation $A_2$ is associated with interval $[t_3, t_4]$ and values $[v_3, v_4]$.
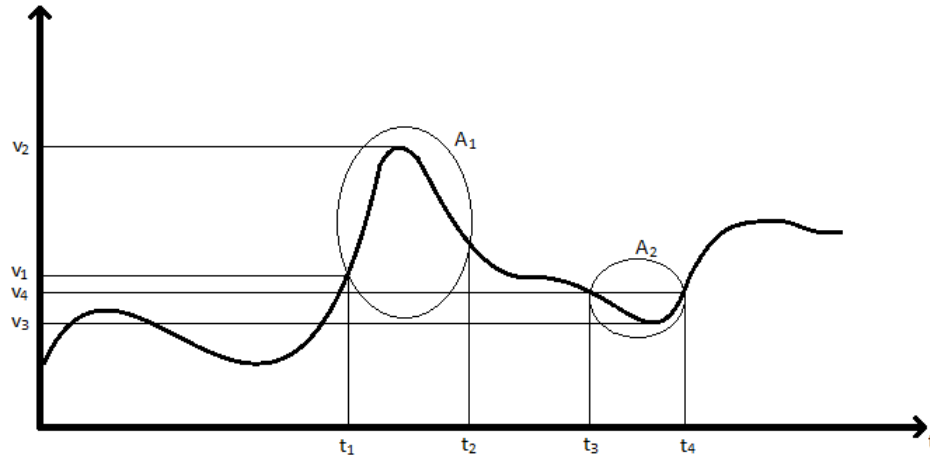
Figure 3.1: Time series annotations

Usually, the reason for annotating a given section of a series is related to the values the series assumed on the annotated interval, e.g., unusual (value) patterns are likely to be annotated. Note, however, that a time series is a function in time, so storing annotations associated with both value and time would be a redundancy. Therefore, in order to facilitate annotation storage, instead of linking the annotation to both an interval and a set of values, the proposed suggestion is to associate an annotation directly with its interval. In this case, all values that fall within the annotated interval are considered associated with the annotation. Figure 3.2 is a rework of figure 3.1 illustrating this concept.
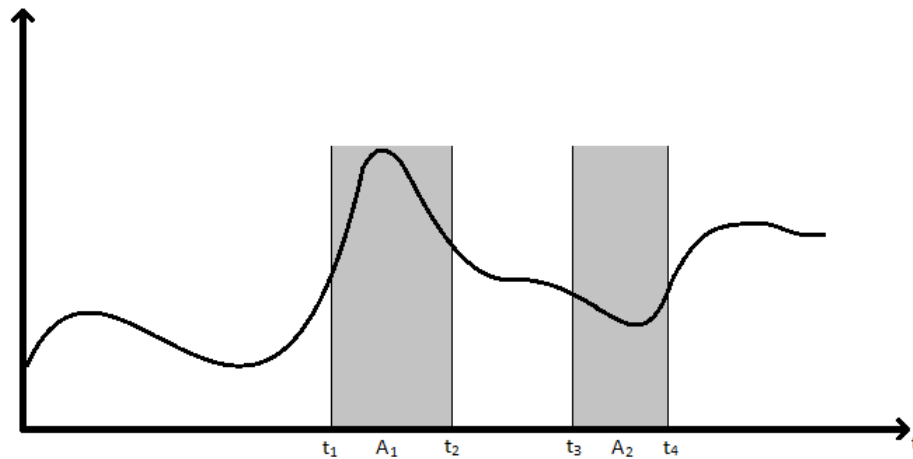


Figure 3.2: Annotations associated with intervals

This same concept also makes it straightforward to associate the same annotation with multiple time series at once. However a constraint must be specified on the involved time series: all series must have values on the annotated time interval. Figure 3.3 illustrates this. Note that in order to associate the annotations $A_1$ and $A_2$ with the time series $s_1$ and $s_2$, both series need to exist through the annotation intervals.
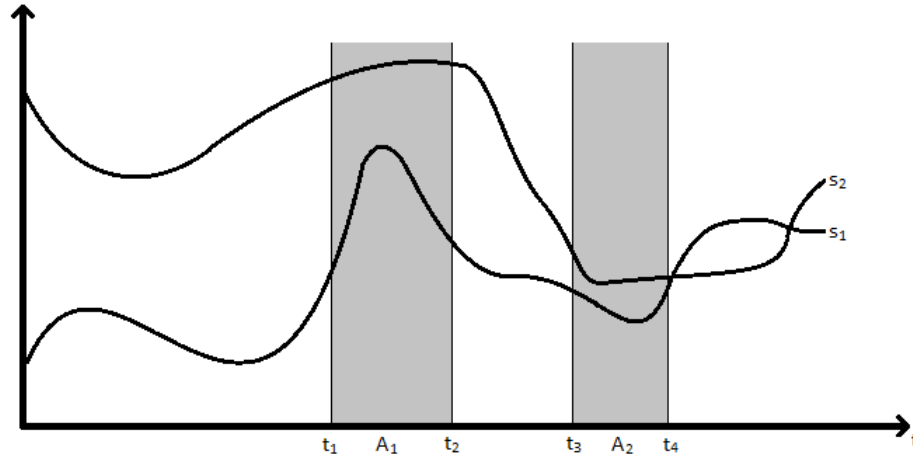


Figure 3.3: Annotating multiple time series with the same annotation

In order to better understand the storage structure behind this idea, tables 3.1 and 3.2 illustrate the storage of an annotation associated with two time series. Table 3.1 shows two stored time series with ids 1 and 2. Both series have values in the time interval 1 to 999. Table 3.2 shows a stored annotation with id 1 and content "corn" associated with these two time series in the time interval 10 to 111.

| TimeSeries | | |
|---|---|---|
| series__id | timestamp | value |
| 1 | 1 | 81 |
| ... | ... | ... |
| 1 | 999 | 233 |
| 2 | 1 | 68 |
| ... | ... | ... |
| 2 | 999 | 55 |

Table 3.1: Time Series storage example

| Annotations | | | | |
|---|---|---|---|---|
| id | annotated_series | range_start | range_end | annotation |
| 1 | 1,2 | 10 | 111 | corn |

Table 3.2: Annotation associated with multiple time series storage example

It is important to note that the proposed storage method does not allow gaps in the interval with which an annotation is associated (i.e. the interval must be continuous). To illustrate this, consider the time series in figure 3.4. Note that the same pattern appears twice in the series and assume that an expert finds it appropriate to associate the same annotation with both occurrences of the pattern. However, even if the content is the same, two different annotations must be created, since they correspond to different intervals.
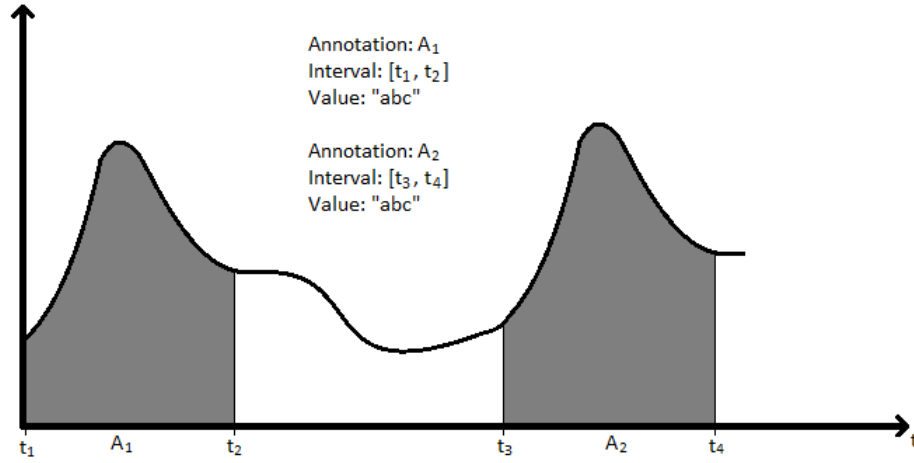


Figure 3.4: Two annotations with the same content

It must also be pointed out that different series can have multiple distinct annotations for the same interval. Figure 3.5 illustrates this concept. Annotation $A_1$ is associated only with series $s_1$ in the interval $[t_1, t_4]$. Annotation $A_2$ is associated only with series $s_2$ in the interval $[t_2, t_3]$ and annotation $A_3$ is associated with both $s_1$ and $s_2$ in the interval $[t_5, t_6]$.
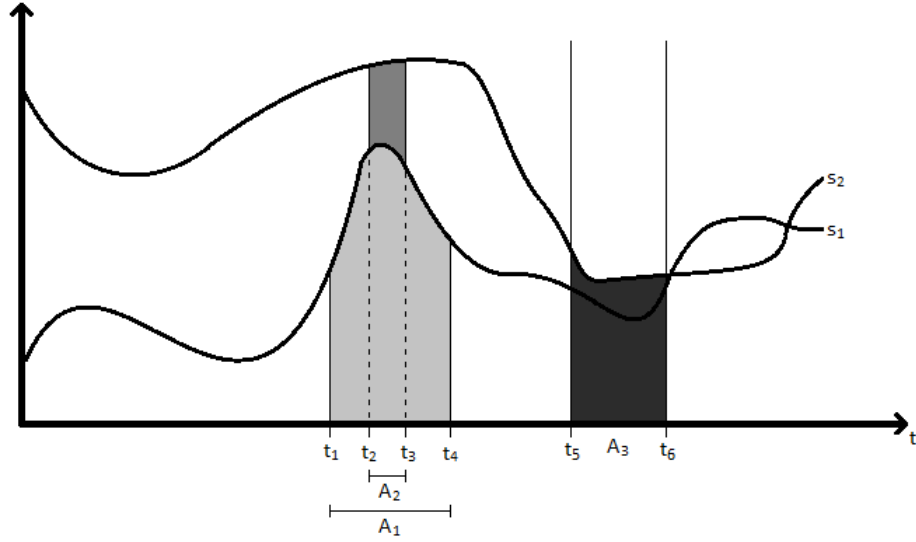
Figure 3.5: Different annotations in the same interval

## 3.2 Database model

This section presents the database model and schema proposed for the framework. The model is composed of three tables as depicted in figure 3.6. The Annotation table stores all annotations. The notion of a single table to store annotations is borrowed from [9]. This table is also used to separate annotations from annotated data. Annotations are associated with time series through the *TimeSeriesMap* table. This table stores the provenance information of the stored series and also the unique id of each series in the database. At last, the *TimeSeries* table stores all time series. This table is also associated with the *TimeSeriesMap* table through the unique time series ids. The arrows in the figure represent foreign key relationships.

We point out that this is a non-normalized schema. The main reason for this is to optimize performance in the target implementation platform (see chapter 4). For other platforms (e.g., desktop), this might not be the best database model.
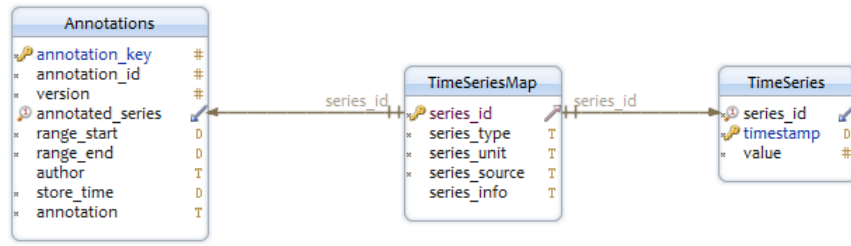
Figure 3.6: Database conceptual model

The database schema is derived from the model as follows:

- ***Annotations****(annotation_key:number, annotation_id:number, version:number, annotated_series:text, range_start:number, range_end:number, author:text, store_time:number, annotation:text)*

- ***TimeSeriesMap****(series_id:number, series_type:text, series_unit:text, series_source: text, series_info:text)*

- ***TimeSeries****(series_id:number, timestamp:number, value:number)*

There follows a detailed description of the tables and their attributes:

- **TimeSeriesMap table:** This table stores metadata information for every time series in the database.

    - *series_id:* This is the id of a time series. It is the primary key of this table and is exported as foreign key to all other tables. Whenever a time series is about to be stored, an entry is created in this table, assigning an id to the time series. The series' values can then be stored in the *TimeSeries* table.

    - *series_type:* The time series type, e.g., satellite-ndvi (for series generated by satellites measuring ndvi), sensor-temperature (for series generated by sensors measuring temperature), and so on.

    - *series_unit:* The unit in which the values of this time series are stored, e.g., for a series related to temperature it can be $^o$C or $^o$F.

    - *series_source:* The source of the values for the time series. This may be a satellite or sensor id, or even a person name. In other words, this attribute provides provenance information. Provenance data is essential for time series, since data is constantly being created with no centralized control over its integrity. Because time series sources may vary in terms of quality, it is important

to provide provenance together with other context information which can help experts judge whether results are trustworthy.

– *series_info:* Further information that can be associated with a time series.

- **TimeSeries table:** This table stores all time series in the database.

  – *series_id:* Foreign key from *TimeSeriesMap* table. It is the id of the time series.

  – *timestamp:* This is the timestamp of a time series value. This attribute, together with the series_id, form the primary key of this table.

  – *value:* This is the value of the time series associated with the timestamp.

- **Annotations table:** This table stores all annotations associated with any time series stored in the database.

  – *annotation_key:* Primary key within the *Annotations* table.

  – *annotation_id:* The annotation id. This uniquely identifies an annotation in the database. This could not be the primary key, since annotations may have multiple versions and all versions share the same annotation id.

  – *version:* The annotation version. Each annotation starts at version 1. The version is increased whenever a modification is done on that annotation. Deleting the annotation changes the annotation contents to some predefined value and also increases its version. This means that a deleted annotation can be restored if an expert chooses to do so. The only way to completely remove an annotation is to delete the series associated with it.

  – *annotated_series:* Set of foreign keys from the *TimeSeriesMap* table. These are the ids of all time series with which this annotation is associated.

  – *range_start:* The start of the time range for which this annotation is valid. This is a timestamp value.

  – *range_end:* The end of the time range for which this annotation is valid. This is a timestamp value.

  – *author:* The author who created or modified the annotation.

  – *store_time:* The timestamp when this version of the annotation was stored in the database (transaction time).

  – *annotation:* The annotation contents.

Note that this database adds versioning to the annotations. After an annotation is stored, its states are never lost. Instead, new versions are added to the database. The same does not happen with time series. Changes to time series values are not supported once they are stored. New values may be appended to a time series, but it is still considered the same series and not a new version of it. This structure is related to a rollback database model as introduced by Snodgrass in [30]. Since the transaction time of annotations is stored, it is possible for an expert to restore (rollback) an annotation to any of its previous states. Table 3.3 illustrates the storage of an annotation containing three versions associated with the time series with id = 1. Note that the versioning schema allows for collaboration between experts. In more detail, the annotation contents show that John discusses with Bob about the actual meaning of the series for the range [2,20].

| Annotations | | | | | | | |
|---|---|---|---|---|---|---|---|
| id | version | annotated_series | range_start | range_end | author | store_time | annotation |
| 1 | 1 | 1 | 2 | 20 | John | 12450702 | corn |
| 1 | 2 | 1 | 2 | 20 | Bob | 12450864 | Update, this is rice |
| 1 | 3 | 1 | 2 | 20 | John | 12451022 | New analysis shows this is indeed corn |

Table 3.3: Annotation versioning example

## 3.3   Architecture

The framework architecture is depicted in figure 3.7. Full arrows correspond to data flow and dashed arrows correspond to service invocations and responses.

raw time series    annotated time series    raw annotations    query

Interface

Time series processing    Annotation processing    Query processing

Persistence layer

Time series

Annotations
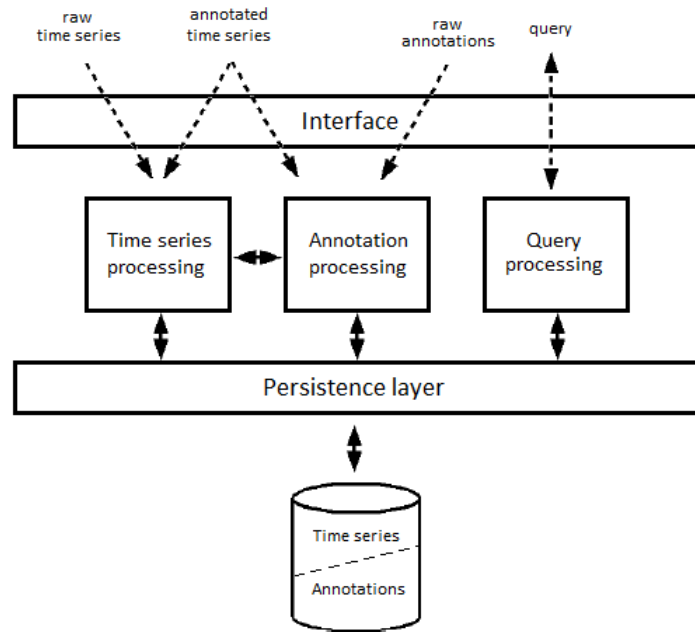
Figure 3.7: Framework architecture

The framework is composed of 3 layers:

- **Interface:** Provides access to the framework services to either end users or external services.

- **Modules:** Provides all services needed to process time series and annotations.

    - **Time series processing module:** Processes and stores time series. Also responsible for appending new data to existing time series.

    - **Annotation processing module:** Processes and stores annotations, associating them to the given time series in the database.

    - **Query processing module:** Handles all query requests that come through the interface layer. Handles queries for time series, annotations, or both.

- **Persistence:** Receives requests from the modules layer to store or retrieve data on the database where annotations and time series are stored.

The following use cases will provide a better understanding of the framework:

1. **Storing a time series:** In order to store a time series, the framework may receive as input a file containing the entire time series or, alternatively, continuous updates of

time series values to be appended to a given series. In the first case, a file containing a time series is input through the interface layer. This file goes through the *Time Series processing* module, that requests the persistence layer to create an id for the new time series in the database. This id will then be used to store all *<value, timestamp>* tuples from the time series file. In the second case, an event arrives through the interface, containing the time series id (as stored in the database) and a set of *<value, timestamp>* tuples. The *Time Series processing* module will then obtain the time series id from the database and request the persistence layer to append all *<value, timestamp>* tuples to it.

2. **Annotating stored time series:** The interface provides facilities for an expert to annotate stored series. The expert may create, delete, and update annotations. To create a new one, (s)he must select one or more series, indicate the time interval to be annotated and provide the annotation content and author name. This information is passed to the *Annotation processing* module that requests the persistence layer to store the annotation associated to that time series. Annotation modification or deletion is processed as follows: The expert requests to see one or more series and their associated annotations. The *Query processing* module returns this information and displays it to the expert. The expert selects the desired annotation and performs the modification or deletion. The *Annotation processing* module receives the information and requests the persistence layer to create a new version for that annotation id.

3. **Storing an annotated time series:** In this case, the time series must be stored in a file together with its annotations. This file goes through the *Time series processing* module in the same way as in the "Storing a time series" use case. This time however, the module will also retrieve the annotations from the file (see algorithm 1) and will forward them to the *Annotation processing* module. Both modules request the persistence layer to store the time series and associated annotations.

4. **Querying for an annotation:** In order to query for annotations, the interface will provide means to search for annotation content, authors, and annotations on a specific time series set. Section 3.4 presents further details on query possibilities.

Algorithm 1 shows the pseudo code to retrieve annotations from a file containing an annotated time series:

---

**Algorithm 1** Annotated time series processing

---

**Input:** Let F be a file containing a time series and its annotations. Each line shall contain a tuple in the form: <timestamp, value, annotation>, where the annotation item may be empty.

**Output:** Time series are stored in the database and annotations are forwarded to the processAnnotations algorithm (algorithm 2).

1: $i \leftarrow 0$
2: **for all** $line \in F$ **do**
3:     $timeSeries[i].timestamp \leftarrow line.timestamp$
4:     $timeSeries[i].value \leftarrow line.value$
5:     $annotation[i].timestamp \leftarrow line.timestamp$
6:     $annotation[i].content \leftarrow line.annotation$
7:     $i \leftarrow i + 1$
8: **end for**
9: $id \leftarrow$ storeTimeSeries($timeSeries$)
10: {Call algorithm 2 passing $annotation$(containing a set of annotation tuples) as $A$ and $id$ as $S$.}

---

After the time series <timestamp, value> tuples are retrieved from the file, a call is made to the persistence layer to store the series (line 9 of algorithm 1). When this happens, the persistence layer assigns an id to the time series. Once the file processing finishes and the time series is stored, the annotations can be further processed as illustrated in algorithm 2. Algorithm 2 does not receive the start and end of the time interval of each annotation, therefore the role of algorithm 2 is to find the start and the end of the time interval of the input annotations. After the annotation interval is found, a call is made to the persistence layer to store each annotation, together with the corresponding time interval.

---

**Algorithm 2** Process annotations

---

**Input:** Let A be a set of annotation tuples in the form <timestamp, content> and S the id of the time series the annotation is associated with.

**Output:** The annotations are stored in the database together with the corresponding time intervals.

1: $i \leftarrow 0$
2: $\{x$ will store the content and interval of the annotation currently being processed inside the loop below.$\}$
3: $x.content \leftarrow NULL$
4: $x.intervalStart \leftarrow 0$
5: $x.intervalEnd \leftarrow 0$
6: **for all** $annotation \in A$ **do**
7:     **if** $x.content \neq annotation[i].content$ **then**
8:         $\{$New annotation found. Mark the end of the interval of the previous one, if any.$\}$
9:         **if** $x.content \neq NULL$ **then**
10:             $x.intervalEnd \leftarrow annotation[i-1].timestamp$
11:             storeAnnotation($x$, $S$)
12:         **end if**
13:         $x.intervalStart \leftarrow annotation[i].timestamp$
14:         $x.content \leftarrow annotation[i].content$
15:     **end if**
16:     $i \leftarrow i + 1$
17: **end for**

---

## 3.4   Query possibilities

This section presents examples of queries that are possible with the proposed framework. Some of these queries have been discussed with EMBRAPA experts. The queries are organized by input. Each input has a set of possible outputs.

1. Input: annotation content

    1.1. Output: all annotations which have the given content.

    1.2. Output: all series that have that annotation content among its annotations, considering all annotation versions (i.e., past contents of annotations are also returned).

    1.3. Output: all series that have that annotation content among its current annotations (i.e., considers only the current content of an annotation, ignoring its

past contents). The series should be displayed in a way that allows experts to compare them. Figure 3.8 shows an example of the result of this query, for input="corn", and output=all series that have "corn" among their annotations.
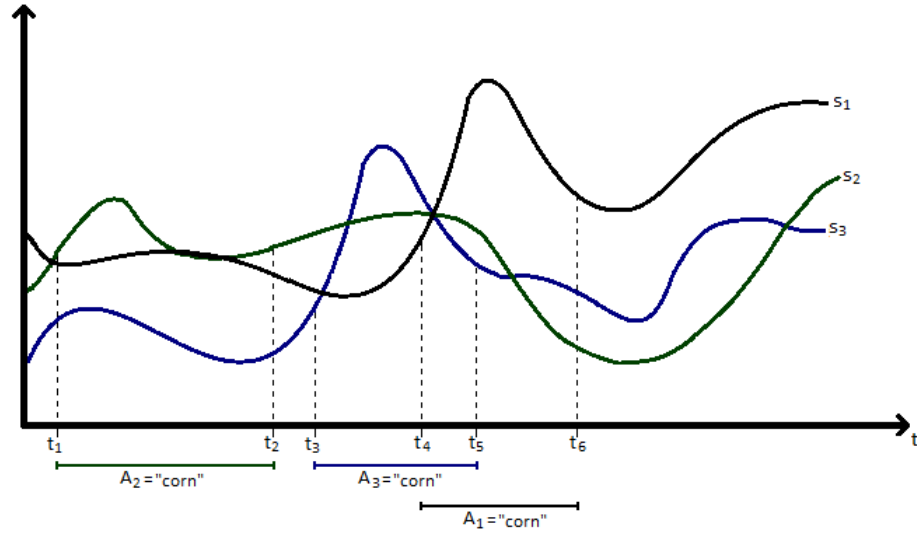


Figure 3.8: Query result - annotation content

1.4. Output: Same as 1.3, but returning only the interval of each series that is associated with the given annotation. Figure 3.9 shows an example of such result. Note that each interval is set to an unique start point and the length of the time interval in each time series may be different. In other words, the x-axis is normalized. The goal is to show the patterns that are annotated, regardless of the annotation period. This is an important output, and has been requested by EMBRAPA experts.
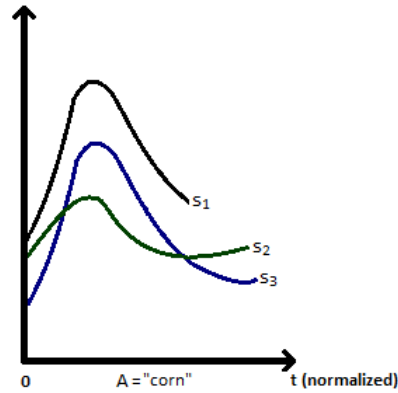
Figure 3.9: Query result - annotation content (annotation intervals only)

1.5. Output: the name of annotation authors that created annotations with that content.

1.6. Output: all provenance information associated with series that have that annotation content among its annotations.

1.7. Output: all history (current and past contents) of annotations that have that content as the current one.

2. Input: set of annotation contents

2.1. Output: all series that have all annotation contents in the given set among its current annotations (i.e., considers only the current content of an annotation, ignoring its past contents). A simple example of such a query is when a time series is associated with one annotation in its entirety, and also with other partial annotations. This annotation associated with the entire series could be, for instance, the geographic position where the time series values have been measured. Therefore this makes it possible to query for time series generated at a given geographic position that are also associated with further annotations. Figure 3.10 shows a result of a query for time series associated with annotations "corn" and "-22.83125,-47.121735" (annotation containing the geographical position associated with the time series).
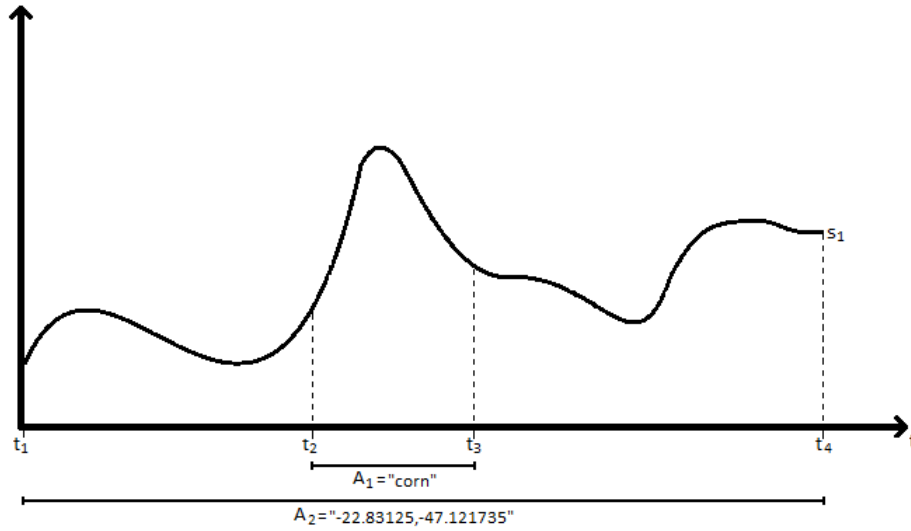
Figure 3.10: Query result - annotation content set

3. Input: annotation author

   3.1. Output: all annotations created by that author, considering all annotation versions (i.e., past contents of annotations are also returned).

   3.2. Output: all current annotations created by that author (i.e., considers only the current content of an annotation, ignoring its past contents).

   3.3. Output: all series that have annotations created by that author among its annotations.

   3.4. Output: all provenance information associated with series that have annotations created by that author.

4. Input: time series id

   4.1. Output: all annotations associated with that time series, considering all annotation versions (i.e., past contents of annotations are also returned). Deleted annotations shall also be returned (i.e., the predefined content for deleted annotations will be returned in this case).

   4.2. Output: all current annotations associated with that time series (i.e., considers only the current content of an annotation, ignoring its past contents). Deleted annotations shall also be returned if the current annotation content represents a deleted annotation.

4.3. Output: the name of all authors that have created annotations associated with the time series.

4.4. Output: all provenance information associated with the time series.

5. Input: time series id and a time interval

Same outputs as in 4.1, 4.2 and 4.3, but returning values within the given time interval only.

6. Input: sets of time series ids

Same outputs as in 4.1, 4.2 and 4.3, but returning only annotations associated with each and every series in the set.

7. Input: sets of series ids and a time interval

Same outputs as in 4.1, 4.2 and 4.3, but returning only annotations associated with each and every series in the set and within the given time interval.

8. Input: time series source name

8.1. Output: the names of all time series generated by that source (same provenance).

8.2. Output: all annotation contents associated with time series generated by that source.

It must be pointed out that, as will be seen in chapter 4, only queries 1.1, 1.3 (without the graphical comparison), 1.7, 4.2, 4.4 and 6 (only the outputs related to 4.2) have been implemented in this work. The remaining queries may be implemented by extensions of this work.

## 3.5 Conclusions

This chapter presented the specification of the framework proposed to manage annotations associated with time series in a database. It also presented the proposed database model and core algorithms within the architecture. The next chapter presents implementation aspects of this framework and discusses the framework validation using real data.

# Chapter 4

# Serial Annotator: Implementation aspects

This chapter presents the implementation aspects of this dissertation. Section 4.1 presents the technologies used and a few implementation details. Section 4.2 presents the smart phone application. Section 4.3 presents the tests performed to validate the framework, using data from the agricultural domain. Finally, 4.4 presents some conclusions.

## 4.1 Technologies used and database implementation details

### 4.1.1 Android framework

The framework was implemented on a mobile device running the Android platform. Since the main focus of this application is the agricultural domain, and in this domain there is extensive work in external locations, having an application running on a mobile device or a tablet facilitates the adoption of the application, allowing for better information exchange and analysis.

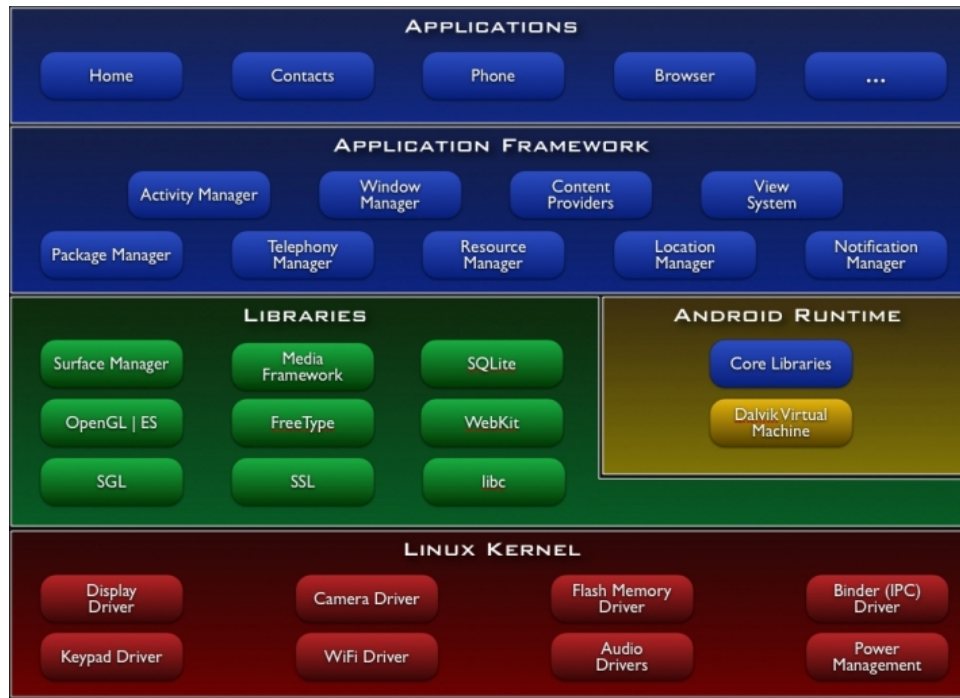The architecture of the Android platform can be visualized in figure 4.1.

Figure 4.1: Android architecture, reproduced from [12]

The Android OS can be referred to as a software stack of different layers, where each layer is a group of several program components. Together it includes operating system, middleware, and important applications. Each layer in the architecture provides different services to the layer just above it. A complete explanation of all Android framework components can be found at [12]. This section will provide some details on a few of these components in order to provide better understanding of the implementation in section 4.2:

- **Applications:** This is the layer where all applications live. They communicate with the Android framework using components of the Application Framework layer. Considering the components of our framework depicted at figure 3.7, the interface layer and part of the modules layer lie in the Android application layer.

- **Content Providers:** Content providers lie within the Application Framework. They manage access to a structured set of data. Content providers are the standard interface that connects data in one process with code running in another process. Considering the components depicted at figure 3.7, the persistence layer is implemented through a content provider. This content provider will connect the application with the underlying SQL database.

- **SQLite:** The Android framework has an embedded SQLite library. SQLite is a software library that implements a self-contained SQL database engine [32].

### 4.1.2 Database implementation details

The database model was slightly modified on the implementation to increase flexibility and for performance reasons. Instead of one table holding all time series entries as described in section 3.2, each time series has been stored in its own table. Whenever a new time series is inserted, a new table is created in the database and the table name is inserted in the *TimeSeriesMap* table. Figure 4.2 shows this modification.
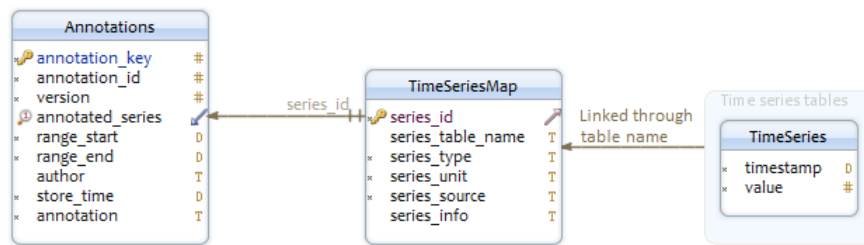


Figure 4.2: Database model (implementation)

Another performance enhancement was done in the *annotated_series* attribute of the *Annotations* table, where the time series ids are kept sorted. That is, if an annotation is associated with series 5, 15, and 9, the *annotated_series* attribute will have the content "5,9,15". The ids are sorted in order to optimize queries, otherwise in order to find annotations associated to series 5, 9, and 15, six queries would need to be performed, considering all possible id combinations: "5,9,15", "5,15,9", "9,5,15", "9,15,5", "15,5,9" and "15,9,5".

## 4.2 Presenting Serial Annotator

This section presents Serial Annotator, the application developed to validate the framework. In order to provide a better understanding, it will be presented through its main use cases.

### 4.2.1 Inserting a time series

This use case shows how a time series can be inserted from a file. The file must be a comma separated value (csv) file. Each line in the file must have the format depicted in

figure 4.3 (lines containing other formats are ignored):

```
"20000218",0.352400,""
"20000305",0.365700,""
"20000321",0.875900,""
"20000406",0.951900,""
"20000422",0.870000,""
"20000508",0.889800,""
"20000524",0.890200,""
"20000609",0.900600,""
"20000625",0.888100,""
```

Figure 4.3: CSV file example

The first item in each line is a timestamp that must follow a predefined format (selected by the user as can be seen in figure 4.4). The second item is the time series value at that timestamp. The third value is an optional annotation for that timestamp.

Figure 4.4 shows the flow to be followed in order to insert a time series. Figure 4.4(a) shows the screen that is displayed when Serial Annotator is used for the first time. To insert a new time series the user should click on the *add time series button* ✚ . This will lead to the screen in figure 4.4(b), where a csv file must be selected. The application will then present the screen in figure 4.4(c), where the user must enter a few metadata, such as the name, the timestamp format (that can be found in the first column of the csv file), the series type (selected from a predefined set), the series unit and series source. After all information is selected, if the time series is successfully stored, the time series metadata is displayed in the list shown in figure 4.4(d).

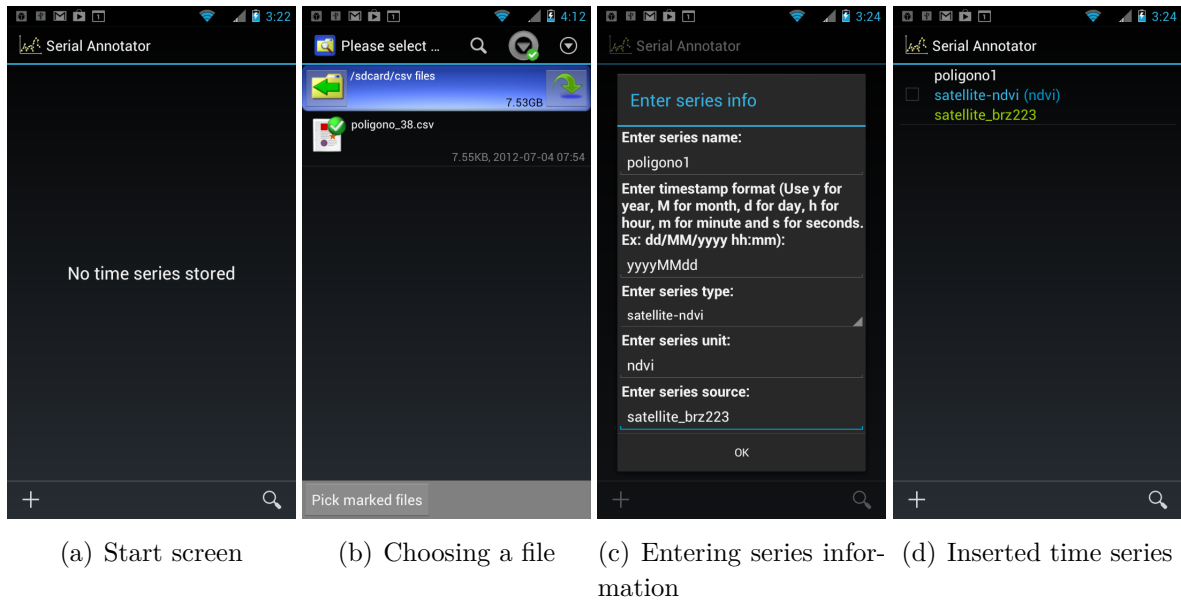(a) Start screen     (b) Choosing a file     (c) Entering series information     (d) Inserted time series

Figure 4.4: Inserting a time series

It is also possible to create time series from stream data. Serial Annotator accepts event-based data entries from available cell-phone sensors or from other applications. As an example, since most Android phones have a GPS, the phone position could be sent over events to the Serial Annotator application, creating a time series with the phone position over time. On the current implementation, the streaming time series data is received through an Android system event called Intent [12], but depending on the sampling interval, other event capture solutions may be adopted. Since the interface used to receive streaming data is public, independent applications may be created to send data to Serial Annotator and published to the Android ecosystem. This allows for flexibility and interoperability. The framework also supports associating annotations with stream data on the fly, even though this has not been implemented.

## 4.2.2 View time series annotations

Figure 4.5 shows the flow to be followed in order to view an annotation associated with a time series. At first, the time series must be selected. To select a series, the user must click on the box icon at the left side of the list item. Figure 4.5(a) shows a selected time series. The user must then click on the *view button* in order to open the time series. This will lead the user to the screen in figure 4.5(b), where the time series is displayed graphically. This graphical representation will contain all time series values, which may be hard to visualize on a smart phone screen. To alleviate this problem, the graphical representation has a zooming feature. The user can touch on the desired portion of the

series graphic to zoom on it. At the bottom of this screen, all annotations associated with the series are displayed in a list. When an annotation is selected, the annotation time interval is highlighted on the time series chart, as shown in figure 4.5(c).



(a) Select time series     (b) Time series displayed     (c) Annotation selected
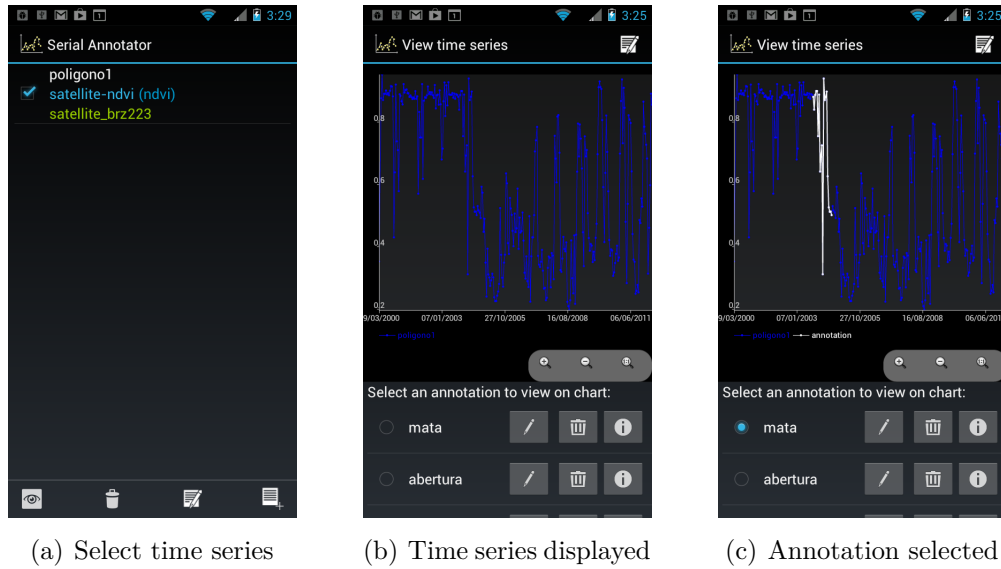
Figure 4.5: View time series annotations

It is also possible to view annotations associated with multiple time series. To do that, the user may simply select multiple series at once and click on the view button. Figure 4.6 shows this flow.

(a) Multiple series selected

(b) Time series displayed
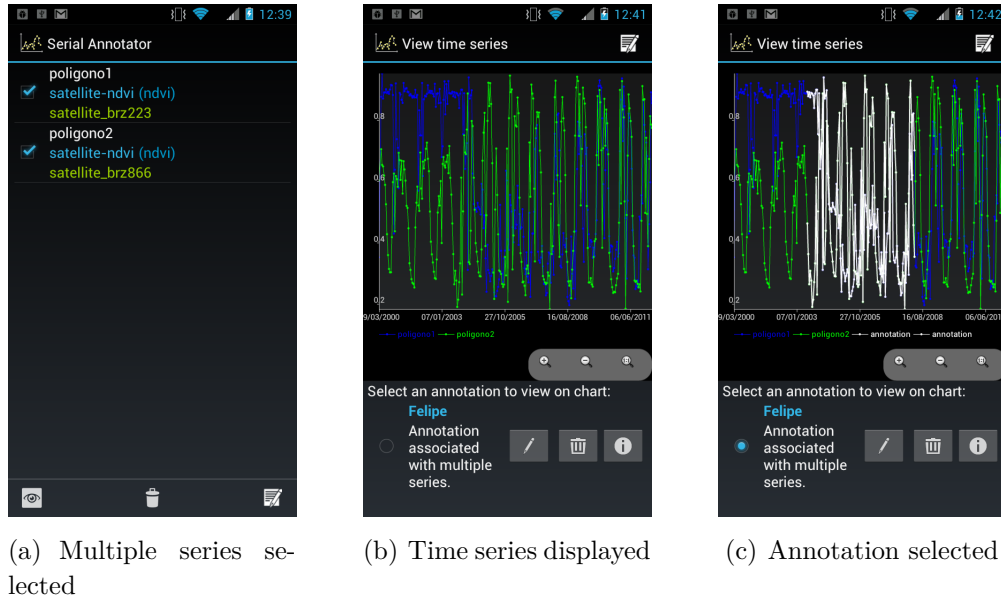
(c) Annotation selected

Figure 4.6: View annotations associated with multiple time series

### 4.2.3   View annotation history

As already discussed in previous sections, annotations are never deleted. Therefore, it is possible to query all previous states of an annotation, as described in item 1.7 of section 3.4. Figure 4.7 shows the flow to be followed in order to view previous states of an annotation. The screen in figure 4.7(a) shows a selected annotation. By clicking on the *annotation information button* ⓘ , the list of previous states is displayed as shown in figure 4.7(b). Note that this annotation has previously been deleted (version 3) and has been subsequently restored (version 4). Each line in the annotation history list shows the author of the modification and when the annotation modification was stored (transaction time).

(a) Annotation selected  (b) Annotation history
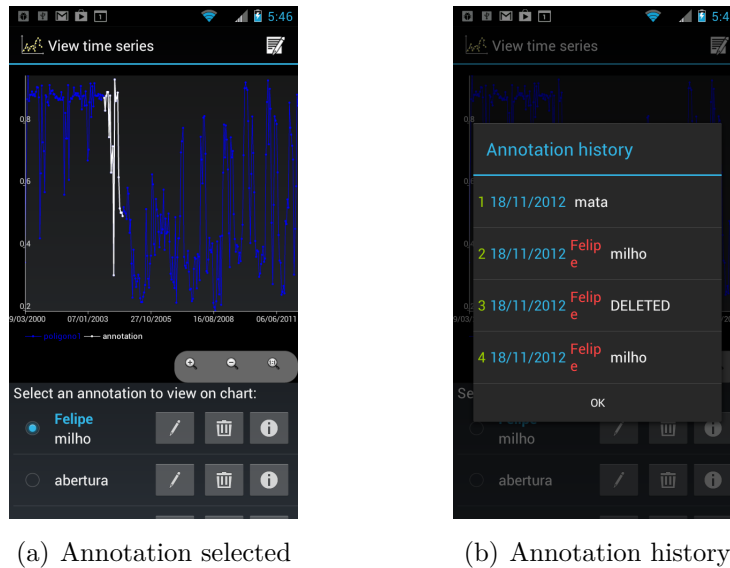
Figure 4.7: View annotation history

### 4.2.4 Editing an annotation

Figure 4.8 shows the flow to be followed in order to edit an annotation. Figure 4.8(a) shows a time series with a selected annotation. By clicking on the *edit annotation button* [ ], the edit annotation screen is displayed, as shown in figure 4.8(b). The user must then enter the author of the modification and the new annotation contents.

The flow to delete an annotation is very similar, and therefore not depicted here. If the user wants to delete an annotation instead of editing it, the *delete annotation button* [ ] must be clicked. However, remember that the annotation is not really deleted, instead, a new version is created with the content "DELETED".

(a) Annotation selected
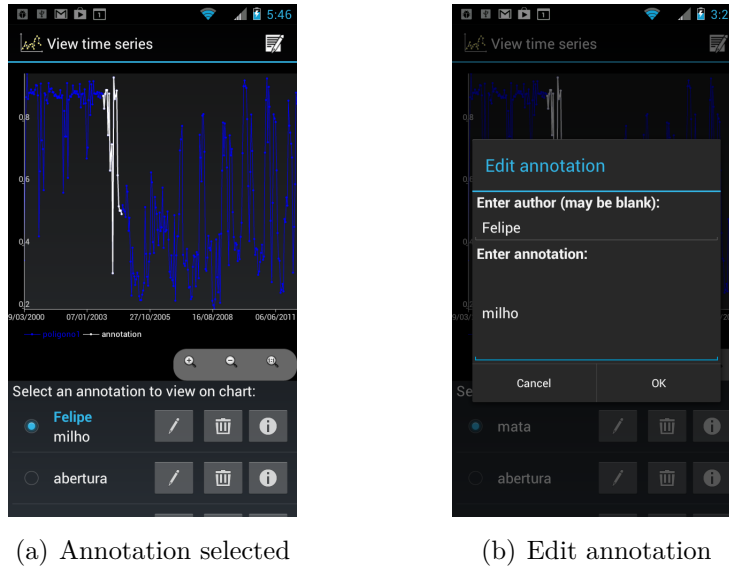
(b) Edit annotation

Figure 4.8: Edit annotation

## 4.2.5 Associating a new annotation with a time series

Figure 4.9 shows the flow to be followed in order to associate a new annotation with a time series. Figure 4.9(a) shows the selected time series. By clicking on the *annotate series button* ▦, the screen in figure 4.9(b) will be displayed. The user must then select the time interval to be annotated. This is done by clicking on the start and end of the interval in the time series chart. Figure 4.9(c) shows the interval selected by the user highlighted in the time series chart. Note that the interval start and end dates are displayed at the bottom of the screen. By clicking on the **OK** button, the screen in figure 4.9(d) will be displayed. In this screen, the user must then enter the author of the annotation and the annotation contents.
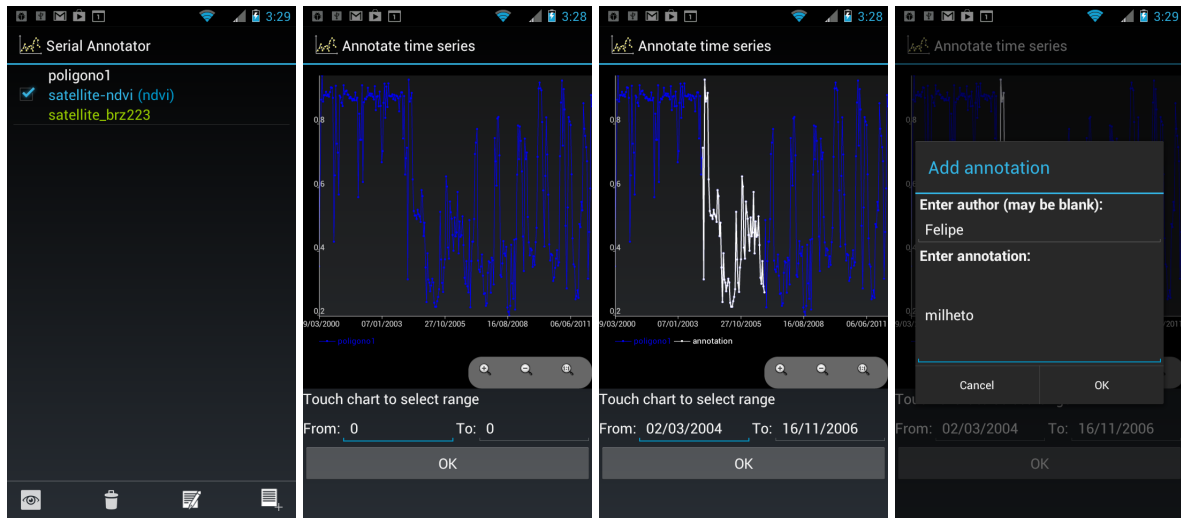
(a) Series selected  (b) Select annotation interval  (c) Interval selected  (d) Add annotation info

Figure 4.9: Associate annotation with a time series

It is also possible to associate an annotation with multiple time series. To do that, the user may simply select multiple series at once and click on the annotate series button. Figure 4.10 shows this flow.
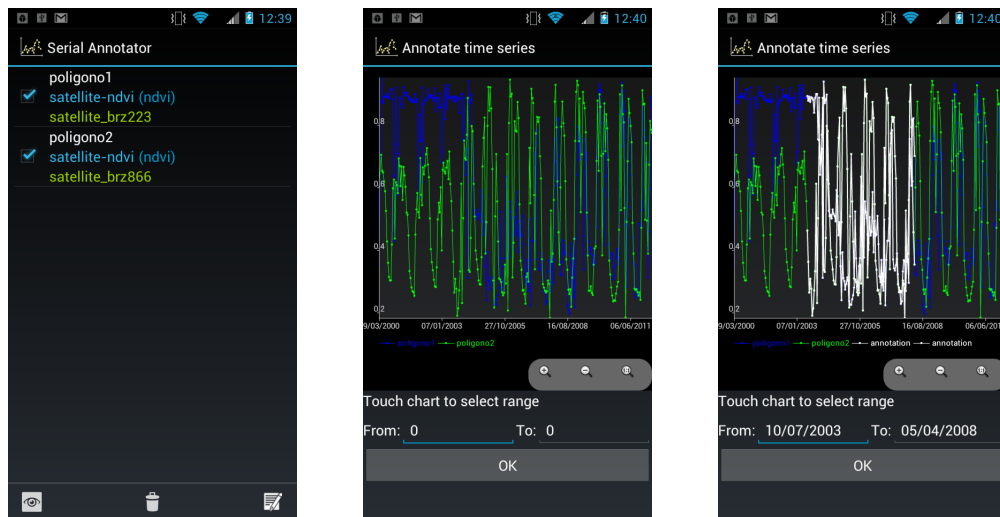


(a) Multiple series selected  (b) Select annotation interval  (c) Interval selected

Figure 4.10: Associate annotation with multiple series

### 4.2.6   Querying annotations

It is possible to query the annotations stored in the database given a query string, as described in item 1.1 of section 3.4. Figure 4.11 shows the flow to be followed in order to query for an annotation. Figure 4.11(a) shows the main application screen. The user must then click on the *query button* 🔍, which will lead to the screen shown in figure 4.11(b). After the user enters some string on the search field and clicks on the "Go" key, a list with all annotations that matched that string together with the annotation authors is displayed as shown in figure 4.11(c). The user may then select one annotation from this list and the time series with which this annotation is associated will be displayed, as shown in figure 4.11(d).
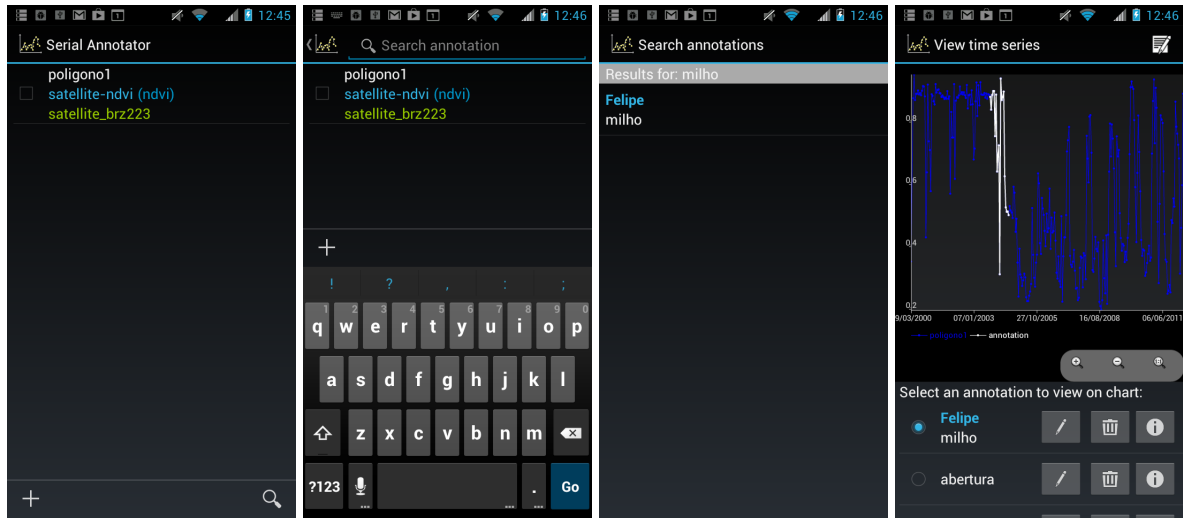


(a) Start screen     (b) Enter query string     (c) Query results list     (d) Annotation selected

Figure 4.11: Query annotations

## 4.3   Tests and validation

This section presents a validation of the proposed framework with real data, comparing it with related work proposals described in section 2.3.

The data used for tests and validation was provided by EMBRAPA. It consists on a set of files, each one containing a time series of Normalized Difference Vegetation Index (NDVI) values over time extracted from a polygon in a satellite range. The NDVI is a measurement of density of green on a patch of land. When sunlight strikes objects, certain wavelengths of this spectrum are absorbed and other wavelengths are reflected. The pigment in plant leaves, chlorophyll, strongly absorbs visible light and the leave cell

structure reflects near-infrared light [26]. By analyzing the reflected sunlight in a target area, it is possible to determine the vegetation density in that area. With the NDVI values, experts are also capable of identifying different kinds of crops. More than 400 time series with such information were provided, consisting of data collected between 2000 and 2011. Each series contains 281 tuples. In order to evaluate the framework with a larger series, the real series were concatenated over and over to build a test series with 120 thousand tuples.

Section 2.3 presented related work on annotations stored in relational databases. Our work makes a similar validation as performed by [9], that is, our schema for time series annotations was compared to the straightforward schema described in [4], where annotations are stored on each cell independently. In the straightforward schema, the number of annotation contents stored is the same as the number of time series tuples, even if the annotation contents are all equal (in the case of a single annotation associated with the entire time series).

In every experiment execution, all time series tuples were annotated, that is, the series may be annotated with a single annotation, i.e., the annotation interval is the entire series, or the series may be annotated with 120 thousand annotations, i.e., each annotation interval has a single tuple.

Two experiments have been performed:

- **Storage experiment:** This experiment compares the space needed (in Mb) to store a time series with all its associated annotations in our schema and in the straightforward schema. Figure 4.12 shows the experiment results. The space required in the straightforward schema is constant, since it stores annotations as an attribute in the time series table (annotations stored together with the annotated data). In our schema, the space required remains approximately constant until 10,000 annotations are inserted. After that, storage increases rapidly as too much space gets consumed in order to link the annotations to their associated time interval in the time series (annotations stored separated from the annotated data). This deterioration is however not a problem in the majority of cases, since it is unlikely that a single time series is associated with such a number of annotations. For the general annotation use case, our schema achieves around 30% storage savings when compared with the straightforward case.
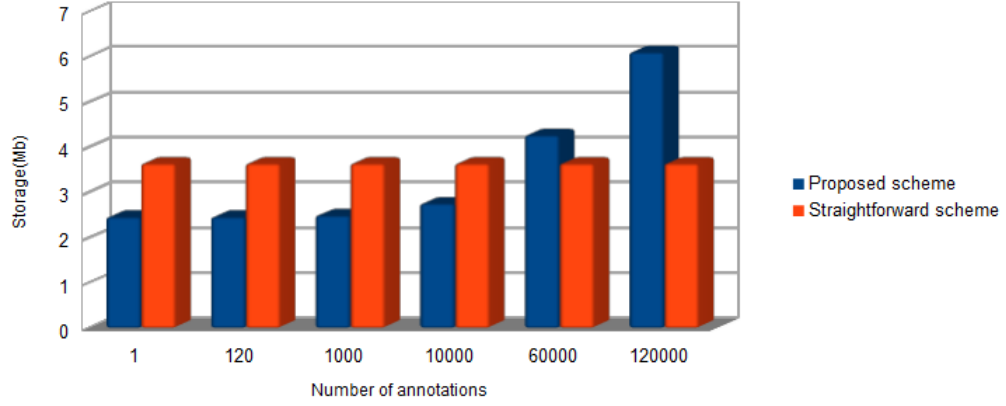
Figure 4.12: Storage overhead experiment

- **Query performance experiment:** This experiment compares the performance of a query in our schema and in the straightforward schema. The query used in this experiment represents a request to "return all annotations with a given content". This query was chosen since it is one of the most common queries for the domain in question (agriculture). Query processing time was measured using SQLite command line shell with CPU timer measurements enabled [32]. The results are measured in milliseconds and represent the mean value of 10 executions. Figure 4.13 shows the experiment results. The query performance in the straightforward schema is almost constant (larger than 80ms), since it needs to go over all 120 thousand annotation contents, regardless of the number of annotations. In our schema, query performance is much better (under 10ms) up to 60,000 annotations and increases up to values close to those of the straightforward schema when the number of annotations equals the number of time series values. Therefore, for the general annotation use case, our schema achieves an order of magnitude better performance when compared with the straightforward case.
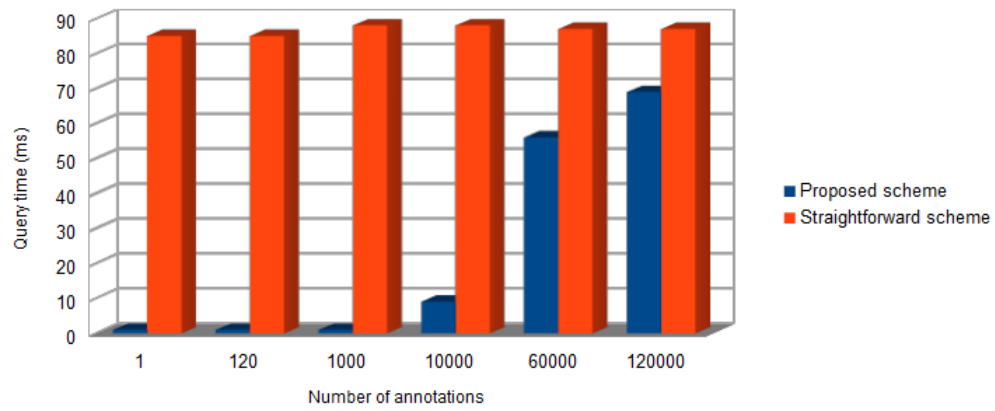
Figure 4.13: Query performance experiment

## 4.4   Conclusions

This chapter presented the implemented application, showing use cases and highlighting implementation aspects. It also showed how the proposed framework behaves when compared to the straightforward schema from [4]. Chapter 5 presents conclusions and future work possibilities.

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusions

This dissertation handled the problem of managing, within a database, annotations associated with time series. It presented the design of a framework to manage these annotations in a relational database and the database model used. Unlike other approaches in the literature, which only allow one annotation associated with an entire time series, the proposed framework makes it possible to associate an annotation with parts of time series. It also supports associating an annotation with multiple time series simultaneously.

The main idea behind the proposed framework is the association of an annotation only with its respective time interval, instead of associating it with the series values. Annotations are also stored apart from the annotated data, which requires less storage space. The framework also allows the versioning of annotations, much demanded by experts, since they can review and restore previous annotations states.

Serial Annotator is a smart phone application used to validate the framework. It has been implemented and tested with real data provided by EMBRAPA experts. The application has also been handled to EMBRAPA so that it could be tested on real situations and was deemed a suitable tool for agricultural field research. Since it runs on a smart phone, it allows experts to analyze data and create annotations while working in external locations.

The results of the validation tests executed with the application showed that it outperforms the benchmark schema, requiring less storage and presenting better performance for the execution of at least one of the most common query scenarios.

## 5.2 Future work

There are several theoretical and implementation-wise extensions to this work. Some of them appeared during reviews with peers and EMBRAPA experts. This section presents these ideas as possible extensions of this work:

- **Semantic annotations:** This work was based on free textual annotations, that is, there is no formal vocabulary for annotations. This creates inconsistencies and makes it almost impossible for the annotations to be interpreted by machines. The usage of semantic annotations, where the annotations use a formal vocabulary based on classes of specific ontologies, would allow the interpretation and interoperability of these annotations across systems.

- **Disassemble annotations:** Since the proposed framework deals with textual annotations, it is possible that a stored annotation represents multiple annotations. For instance, the annotation "corn,rice" is, in fact, a combination of annotations "corn" and "rice". This creates a problem for query processing, as a query for "corn" would also return results of type "rice". Future extensions of this work could therefore focus on means to disassemble multiple annotations.

- **Collaborative annotations:** In the proposed framework, annotations are seen as static labels. If an expert changes an annotation, a new version is created. This is not ideal when experts want to collaborate over annotations, since many pointless versions of the annotation would be created just to store the expert discussions. Future extensions of this work shall consider using ideas from related work mentioned in section 2.4 to enhance collaboration among experts.

- **Automatic annotations based on patterns:** As mentioned in section 2.1, most research on time series concentrates on data mining. Future extensions of this work could deal with data mining in order to allow for automatic annotation of time series based on pattern similarity. That is, given a time series S, which has an associated annotation A on a time interval $(t_1, t_2)$, and given that in this interval the time series has the pattern P, annotate all time series in the database which have a pattern similar to P with annotation A in the corresponding time interval.

- **Improve stream capabilities:** The smart phone application described in section 4.2 has the capability of receiving stream time series data from other sources (applications, sensors) in the smart phone. However, this interface has yet to be tested with a real streaming data source. The interface uses the Android framework Intents [12] to capture incoming data storage requests, and may therefore not be able to cope with a high data incoming rate.

- **Inter-annotation association:** The proposed annotation storage methodology does not allow the same annotation to be associated with different time intervals. This creates a limitation when two annotations in different time series have a causality relationship. That is, suppose that time series $S_1$ has pluviometric values for a specific geographic position and that time series $S_2$ has NDVI values for the same position. Since rain will directly affect the NDVI value, annotations in $S_1$ are likely to be related to annotations in $S_2$ that appear some time in the future. For instance, the annotation "heavy rain" in $S_1$ may be related to annotation "crop increase" in $S_2$. It would be reasonable for an expert to create only one annotation contemplating both observations in the two series, but this is not possible in the current framework. The Annotea project [33, 15] has a solution to relate annotations and might be used to solve this problem.

- **Additional queries:** Many of the queries listed in section 3.4 could not be implemented in the smart phone application due to time constraints. Most of those queries have been discussed with EMBRAPA experts and are considered to be essential in future versions of the application. It is also possible that further queries (not listed in section 3.4) will be specified when the application is put to use in the field.

- **Non-textual annotations:** There are several formats of annotations in the literature, as mentioned in section 2.2. Another extension possibility would be to use non-textual annotations. This would require, among others, changing the proposed model.

# Bibliography

[1] AMIGUET-VERCHER, J., APERS, P., AND WOMBACHER, A. The Identification problem: A description. In *Proceedings of the 8th World Congress on Services* (Hawaii, USA, 2012), pp. 33–40.

[2] AOTO, R., AND SHIMIZU, T. Propagation of Multi-granularity Annotations. In *Proceedings of the 22nd International Conference on Database and Expert Systems Applications* (Toulouse, France, 2011), vol. 6861, pp. 589–603.

[3] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The Semantic Web. *Scientific American Magazine* (2001), 34–43.

[4] BHAGWAT, D., CHITICARIU, L., TAN, W.-C., AND VIJAYVARGIYA, G. An annotation management system for relational databases. *The VLDB Journal 14*, 4 (Oct. 2005), 373–396.

[5] CHAPMAN, A. P., JAGADISH, H., AND RAMANAN, P. Efficient Provenance Storage. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada, 2008), pp. 993–1006.

[6] CHENEY, J., CHITICARIU, L., AND TAN, W.-C. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases 1*, 4 (2007), 379–474.

[7] DING, H., TRAJCEVSKI, G., SCHEUERMANN, P., WANG, X., AND KEOGH, E. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. In *Proceedings of the VLDB Endowment* (Auckland, New Zealand, 2008), pp. 1542–1552.

[8] DUCHENNE, O., LAPTEV, I., SIVIC, J., BACH, F., AND JEAN, P. Automatic Annotation of Human Actions in Video. In *Proceedings of the 12th International Conference on Computer Vision* (Kyoto, Japan, 2009), no. Section 3, pp. 1491–1498.

[9] ELTABAKH, M. Y., AREF, W. G., ELMAGARMID, A. K., OUZZANI, M., AND SILVA, Y. N. Supporting Annotations on Relations. In *Proceedings of the 12th International Conference on Extending Database Technology* (Saint-Petersburg, Russia, 2009), no. 1, pp. 379–390.

[10] EUZENAT, J. Eight Questions about Semantic Web Annotations. *IEEE Intelligent Systems 17*, 2 (2002), 55–62.

[11] FU, T.-C. A review on time series data mining. *Engineering Applications of Artificial Intelligence 24*, 1 (2011), 164–181.

[12] GOOGLE. Android developers web site. `http://http://developer.android.com/`, 2012. [Online; accessed 17-November-2012].

[13] IBGE. Agronegocio - Portal Brasil. `http://www.brasil.gov.br/sobre/economia/setores-da-economia/agronegocio/`, 2012. [Online; accessed 10-October-2012].

[14] IVANOV, I., VAJDA, P., GOLDMANN, L., LEE, J.-S., AND EBRAHIMI, T. Object-based Tag Propagation for Semi-Automatic Annotation of Images. In *Proceedings of the 11th International Conference on Multimedia Information Retrieval* (Philadelphia, USA, 2010), ACM Press, pp. 497–506.

[15] KAHAN, J., AND KOIVUNEN, M.-R. Annotea : An Open RDF Infrastructure for Shared Web. In *Proceedings of the 10th international conference on World Wide Web* (New York, USA, 2001), pp. 623–632.

[16] LESAFFRE, M., AND TANGHE, K. The MAMI Query-By-Voice Experiment: Collecting and annotating vocal queries for music information retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval* (Baltimore, Maryland, USA, 2003), pp. 65–71.

[17] LIN, J., KEOGH, E., WEI, L., AND LONARDI, S. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery 15*, 2 (Apr. 2007), 107–144.

[18] LIN, J., AND LI, Y. Finding Approximate Frequent Patterns in Streaming Medical Data. In *Proceedings of the 23rd International Symposium on Computer-Based Medical Systems (CBMS)* (Perth, Australia, 2010), IEEE, pp. 13–18.

[19] MA, X., LEE, H., BIRD, S., AND MAEDA, K. Models and Tools for Collaborative Annotation. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation* (Las Palmas, Spain, 2002).

[20] Macario, C. G. N. *Anotação Semântica de Dados Geoespaciais.* PhD thesis, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil, 2009.

[21] Mariote, L. E. Mineracao de series temporais de dados de sensores. *Master thesis, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil* (2008).

[22] Mariote, L. E., Medeiros, C. B., Torres, R. S., and Bueno, L. M. TIDES - a new descriptor for time series oscillation behavior. *Geoinformatica 15*, 1 (June 2011), 75–109.

[23] Marketoracle. The great depression. `http://www.marketoracle.co.uk`, 2012. [Online; accessed 5-April-2012].

[24] Medicalmingle. Normal ECG. `http://www.medicalmingle.com`, 2012. [Online; accessed 5-April-2012].

[25] Mornpen. About the Mornington Peninsula. `http://www.mornpen.vic.gov.au/`, 2012. [Online; accessed 5-April-2012].

[26] Nasa. Normalized Difference Vegetation Index (NDVI). `http://www.brashttp://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php`, 2012. [Online; accessed 21-December-2012].

[27] Oren, E., Möller, K. H., Scerri, S., Handschuh, S., and Sintek, M. What are Semantic Annotations? *Technical Report, DERI Galway* (2006).

[28] Park, J., Nguyen, D., and Sandhu, R. A Provenance-based Access Control Model. In *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust* (Paris, France, July 2012), pp. 137–144.

[29] RDF. RDF. `http://www.w3.org/RDF/`, 2013. [Online; accessed 22-June-2013].

[30] Snodgrass, R., and Ahn, I. Temporal Databases. *IEEE Computer 19*, 9 (1986), 35–42.

[31] Sousa, S. R. Gerenciamento de Anotações Semânticas de Dados na Web para Aplicações Agrícolas. *Master thesis, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil* (2010).

[32] SQLite. SQLite Home page. `http://www.sqlite.org/`, 2012. [Online; accessed 17-November-2012].

[33] W3C. The Annotea Project. `http://www.w3.org/2001/Annotea/`, 2013. [Online; accessed 22-June-2013].

[34] WENG, C., AND GENNARI, J. H. Asynchronous Collaborative Writing through Annotations. In *Proceedings of the Conference on Computer Supported Cooperative Work* (Chicago, USA, 2004), ACM Press, pp. 578–581.

[35] WU, X., ZHANG, L., AND YU, Y. Exploring Social Annotations for the Semantic Web. In *Proceedings of the 15th International Conference on World Wide Web* (Edinburgh, Scotland, 2006), pp. 417–426.

[36] ZONTA, G. P. J., DALTIO, J., AND MEDEIROS, C. B. Multimedia Semantic Annotation Propagation. In *Proceedings of the 10th International Symposium on Multimedia* (California, USA, Dec. 2008), pp. 509–514.