A **P**olicy **L**anguage **D**efinition for **P**rovenance in **P**ervasive **C**omputing

Aeshah Alsiyami
a.a.d.alsiyami@sussex.ac.uk

A thesis submitted, on 3 January 2012, in partial fulfillment of the requirements for the degree of Doctor of Philosophy (DPhil) in the School of Informatics, University of Sussex, Brighton, UK

DEDICATED TO

MY WHOLE FAMILY,

ESPECIALLY MY MOTHER,

MY HUSBAND AND MY CHILDREN

Rayan, Mohaned, Razan, and Majd

ABSTRACT

Recent advances in computing technology have led to the paradigm of pervasive computing, which provides a means of simplifying daily life by integrating information processing into the everyday physical world. Pervasive computing draws its power from knowing the surroundings and creates an environment which combines computing and communication capabilities. Sensors that provide high-resolution spatial and instant measurement are most commonly used for forecasting, monitoring and real-time environmental modelling. Sensor data generated by a sensor network depends on several influences, such as the configuration and location of the sensors or the processing performed on the raw measurements. Storing sufficient metadata that gives meaning to the recorded observation is important in order to draw accurate conclusions or to enhance the reliability of the result dataset that uses this automatically collected data. This kind of metadata is called *provenance data*, as the origin of the data and the process by which it arrived from its origin are recorded. Provenance is still an exploratory field in pervasive computing and many open research questions are yet to emerge. The context information and the different characteristics of the pervasive environment call for different approaches to a provenance support system.

This work implements a policy language definition that specifies the collecting model for provenance management systems and addresses the challenges that arise with stream data and sensor environments. The structure graph of the proposed model is mapped to the Open Provenance Model in order to facilitating the sharing of provenance data and interoperability with other systems. As provenance security has been recognized as one of the most important components in any provenance system, an access control language has been developed that is tailored to support the special requirements of provenance: fine-grained polices, privacy policies and preferences. Experimental evaluation findings show a reasonable overhead for provenance collecting and a reasonable time for provenance query performance, while a numerical analysis was used to evaluate the storage overhead.

ACKNOWLEDGEMENTS

First of all, I give thanks to God for granting me the strength and courage to complete this study.

Special thanks go to my supervisors, Dr. Ian Wakeman and Dr. Dan Chalmers, for their guidance and inspiration during this work. Without their support this work would have never been completed.

My in depth love and grateful thanks to my husband who offered his help, and gave his endless support and encouragement. Also I would like to thank my brothers and sisters for their continuous prayer, encouragement and assistance, and particular thanks to my lovely mother (God bless her), who was the most encouraging and supportive person.

I am thankful to Dr. Des Watson for his valuable feedback during the annual review interviews. I am also thankful to all of the members of the Foundations of Software Systems Group. I am especially indebted to the following people for their insightful comments, helpful discussions, and proof reading: Simon Fleming, Yasir Malkani, Lachhman Das Dhomeja, Renan Krishna, and Roya Feizy.

Finally, I would like to express my deep thanks to all my friends who have been such an asset to me in my study.

DECLARATION

I hereby declare that this thesis has not been, and will not be, submitted in whole or in part to another University for the award of a PhD or any other degree.

Signature

CONTENTS

3. PROVENANCE POLICY LANGUAGE

4. ACCESS CONTROL POLICY LANGUAGE

5. MAPPING TO IMPLEMENTATION

LIST OF FIGURES

LIST OF TABLES

LIST OF CHARTS

LIST OF ALGORITHMS

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

This chapter presents an overview of the research area and the motivation behind the need for the proposed system. It also describes our contribution, followed by the challenges imposed by pervasive computing systems. At the end, it describes the organization of the remaining parts of the thesis.

1.1 Introduction and motivation

Recent advances in computing technology have led to the paradigm of pervasive computing, also called ubiquitous computing, which provides a means of simplifying daily life by integrating information processing into the everyday physical world (Li et al. 2008). It creates an environment combining computing and communication capability, and is built on two earlier steps: distributed systems and mobile computing (Satyanarayanan 2001). A pervasive environment requires traditional computer system inputs and outputs such as keyboard, microphone and screens, besides the context information, which is the key data for pervasive systems, such as location, temperature, time, and levels of light and noise (Chalmers 2007).

In pervasive applications, sensors are most commonly used for connecting the environment to the computing process and generating context information,

which needs to integrate the information from a diverse range of sources. Sensor data generated by sensor networks depends on several influences such as configuration and location of the sensor or the aggregation process performed on the raw measurements data (Lange 2010). In order to draw an accurate conclusion from this automatically collected data and provide some form of trust and credibility concerning the source or the data owner, metadata need to be stored that give meaning to the recorded observations. This kind of metadata is called *provenance* data, as the original data and the process by which the result is arrived at from its origin are recorded.

In general, provenance is defined, according to the Oxford English Dictionary (Oxford Dictionaries 2009), as:

> The fact of coming from some particular source; origin, derivation.
>
> The history or pedigree of a work of art, book, etc.; a record of the derivation and passage of an item through its various owners.

Provenance is one kind of metadata and is also referred to as 'lineage' and 'pedigree'; these words describe the creation, recording, processing, ownership, and version history of data. In computer science, the same definition can be applied to data and its provenance information. However, provenance is generally defined as the description of the data source and the process by which it is derived from its origin (Groth et al. 2006, Szomszor & Moreau 2003, Moreau et al. 2004).

Provenance has been recognized as an important consideration in many domains, such as scientific experiments and business transactions, where a new item of data is formed from a variety of diverse resources and complex analysis or simulation. Keeping a complete record of how it was formed and where it is from is essential for demonstrating its quality and trustworthiness, and also for finding errors and reproducing results. In many scientific experiments, recording a complete history of workflow provenance is important in order to enable these experiments to be repeated and verified; avoid duplication of effort; recover the source data of errors; and provide an attribution of the data source (Davidson et

al. 2007, Simmhan et al. 2005). In the business domain, it is essential to provide an audit trail; inquire about the source in the data warehouse; and track the creation of intellectual property (Simmhan et al. 2005). Concerning the database, it is important to determine the reliability and the quality of data; understand the transport of annotation between data sources; and view updates and maintenance (Buneman et al. 2007).

Provenance has been studied for many years and a number of techniques for supporting it have been proposed (e.g. myGrid (Zhao et al. 2004), Trio (Widom 2005) and Karma (Simmhan et al. 2008)). However, it is still an exploratory field, and many open research questions remain to be answered (Simmhan et al. 2005). Some of the emerging research directions are: combining dataflow provenance and fine grain provenance by breaking the black box in the workflow and extending ideas from the data provenance (Buneman et al. 2007); addressing the problem of outsize storage when using the annotation approach for fine grained provenance recording or when using inversion for an unretained data source; using provenance in the trust policy; and archiving the provenance data (Simmhan et al. 2005).

The use of provenance in pervasive systems seems to be different from that mentioned above, since context information may have multiple representations in different forms and at different levels of abstraction, and it is highly interrelated. Context information is not a common kind of data such as a tuple or an attribute of a table in a database, and is not a workflow process. Context information in pervasive computing systems can be static or dynamic (Henricksen et al. 2002). The majority of information is dynamic, such as a person's location and activity, which can change from time to time, while the static context information is invariant, such as a date of birth in personal information. Additionally, the different characteristics of the pervasive environment call for different approaches in order to build a provenance-aware application.

Pervasive computing environments are highly dynamic and acquire their power from a knowledge of the surroundings. “Much of the context information involved in pervasive computing is derived from sensors” (Henricksen et al. 2002) (p 170). Sensor networks are widely deployed and range from personal to scientific applications. Examples of these include: building a smart environment (Hiramatsu et al. 2005), meteorology forecasting (Liu et al. 2006); body sensors for health monitoring (Blount et al. 2007); location detection (Ray et al. 2004); environmental condition monitoring (Gehrke & Madden 2004); home energy monitoring (Harris et al. 2007); and traffic monitoring (Guitton et al. 2007).

Sensors produce continuous real-time data streams, which are time ordered. A data stream is a potentially infinite sequence of time-ordered data elements. All such applications are stream-processing systems, including real-time analysis of high volume sensor streams (Gaber 2007). In many cases, this data goes through a process pipeline in order to produce new useful data. Based on these data, different reactions are programmed, such as triggering the appropriate service or making decisions. Keeping a record of the entire process that input data go through in order to alert the output gives the ability to trace results back to the data set in the stream that caused them (Vijayakumar & Plale 2006). Storing sufficient metadata provides answers to critical questions and supplies important information such as:

- Justification for any decision made according to incoming data
- Justification for any trigger process on specific data
- Keeping a record of the activity being applied to data
- Having the ability to recreate the processing graph, which represents the processing provenance data, and to provide the data element of the stream used to generate the output
- Recording the historical stream data
- Defining the low level sensor data that alerts the event
- Defining which sensor was used to obtain this raw reading.

In such applications, to ensure low storage overhead and efficient collection of provenance information, a hybrid model needs to be adapted in order to capture the dependencies and lineage of individual data events. However, the capture mechanism requires access to the computational process's relevant details such as steps, execution information and its arguments. In other words, each process involved in the task needs to be documented. Newer workflow systems have been designed to support provenance collection, such as VisTrails, while earlier systems have been extended to capture provenance, such as Taverna and Kepler (Freire et al. 2008). Provenance management can be built-in during the design of new pervasive systems. However, existing systems can be upgraded to support provenance management, which requires a modification to the existing process, and awareness about how tasks are modelled. Each process has to be instrumented to automatically capture provenance and any relevant information in order to provide documentation of the complete task. So, when any process is executed, the instrumentation can capture and publish provenance information.

The idea of blending computing into the environment, with the feature of being easy to use, raises a number of new uses of provenance information, as follows:

1.1.1 Debugging compliance

Many applications require proper documentation and audit logs for electronic records. Such information can be used to trace the lineage of data, determine the resource usage and optimize the derivation process. Therefore, tracking provenance has become an important aspect of many pervasive applications in order to track back and detect the source data or a process that is the cause of any errors found, and to define the relevant correction (fault location). Faulty data that introduces errors and propagates them to all derived data based on it can be detected by reviewing the pedigree of data backwards, and by reviewing the provenance of the source forwards, all the derived data will be easily allocated (Simmhan et al. 2005a). For example, in an automated clinical

decision (Wang et al. 2007), the availability of a rich medical history can be used for proactive anomaly detection, drug side-effects monitoring and trend analysis of lifestyle activities.

1.1.2 Legal domain

In recent years, electronic records such as pictures and location traces have been widely used in legal proceedings as evidence (Hasan et al. 2007). Provenance information of these records is important for their reliability when used in litigation, digital forensics and intrusion investigations. When the provenance chain is recorded, investigators can follow the chain for ownership history, or detect changes performed on the data by malicious intruders.

1.1.3 Data quality

Collecting the provenance of data enables its user to evaluate its quality based on its source and transformation (Simmhan et al. 2005b). Provenance information is important in order to improve the ability to judge accuracy and evaluate trust. In addition, the sensor data stream often misses some reading or is affected by significant noise and calibration, which is the process of translating the sensor reading into a unit of measurement. When performing an analysis, these missing items of data or unacceptable noise levels need to be filled in by estimating the historical data, in order for it to be easily analysed, visualised and compared with other data. Therefore, this level of detail of lineage metadata can assist in estimating the quality of the data and can be a proof of any filters and interpretation applied.

1.1.4 Ownership

Properly maintained provenance records can help ascertain the ownership of the source data. The provenance chain can be recorded as a tree, and so users can look down to the roots of the derivation tree to see the creators of the data they have used and verify its copyright; or a creator can look up the chain to see who is using their data (Simmhan et al. 2005a).

As can be seen in figure 1.1, the roots of the derivation tree of result 2 are the sensors that generate the reading. The data from sensor 1 and sensor 2 are used by the two aggregation processes, while the data from sensor 3 is used by aggregation 2 only.

A provenance chain is also used to resolve the dilemma of ownership or liability in case of errors. In the case of a strange reading from a sensor, it is possible to identify the cause of the problem, which could be a user default, or a sensor failure due to the lack of a battery, or wireless transmission limitations. For example, in the monitoring of electrical energy usage, when a user connects an extension to a plug, the sensor that measures the electricity usage of that plug may or may not give a high reading, depending on what is in use; in this case it is the user's responsibility.

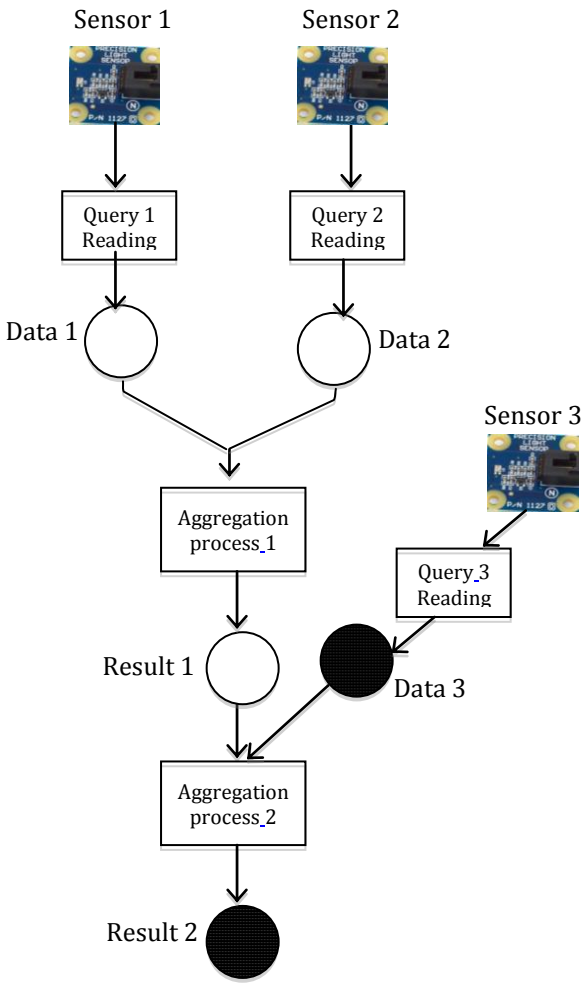


Figure 1.1: A derivation tree of result 2

1.1.5 Causality

One of the compelling benefits of provenance is the causality of production data. Causality captures the input data and parameters together with the sequence of steps that have caused the creation of the derived data (Davidson & Freire 2008). The information provided by causality explains the dependency relationships between data products and the processes used to generate them. An adequate description of a process with the input data and the dependencies between these processes and data provides useful documentation for the data generation process. This information can be used to determine the results that rely on specific data and to reproduce or validate a process.

1.1.6 Informational

Provenance is recorded in order to document data generation and all processing steps used. This information can be searched or queried in order to locate data of interest or for data discovery (Simmhan et al. 2005b). A user can add additional information such as comments or tagging along with provenance, in order to interpret the data in the context that was intended (Simmhan et al. 2005a).

1.1.7 Proof of context

Context is key data for pervasive applications, and is not a common kind of data such as a tuple or a document. Sensor data that is used as context information needs to be interpreted for the user or for an application, in order for it to be understood (Chalmers 2007); for instance, a temperature sensor report voltage that may be converted to a temperature in degrees. Sometimes, a sensing application depends on physical coupling. Tracking the history of the gathered contextual information and any calculation rules applied can be a statement of proof, and can be used to define where a fault occurs, such as from poor coupling or from an analogue/digital conversion.

1.2 Challenges

The ubiquitous computing vision is in providing smart computing services that are embedded into everyday life (Satyanarayanan 2001). These services rely on the context information captured by networked sensors, actuators, mobile devices and appliances. This information has both historical and real-time value and is useful immediately from the moment the services are provided, but it could have additional value when combined with other data collected in a larger distributed area or for historical analysis long after it has been collected. For example, in an emergency situation of a heart attack (Welsh et al. 2003), data on heartbeat and blood pressure, collected by biosensors, are important for monitoring the vital signs of the patient in real time and for doctors to make decisions. While data and results collected by different sensors are important in the immediate situation, they are also useful later when these data can be retrieved by doctors to show the whole treatment carried out on the patient or to show the patient's situation before and after treatment. Hence, keeping track of the provenance of the sensor data or the services provided in a particular situation is very important in order to reconstruct the subject's contextual status that triggered the emergency situation at that time. It can provide a valuable insight for the interpretation of the episodically collected data stream (Chowdhury et al. 2009) or to trace the causality reasons in the case of service failure.

Some of the problems of recording provenance in applications correspond to those already identified in the related work on provenance such as the granularity level of the collected provenance data, or storage overhead. However, data streams produced by sensor networks have different characteristics from other streaming data such as media streaming or finance streaming. The fundamental characteristics are as follows (Kim et al. 2005):

1. The sensor data stream is time-ordered and will generally be a sequence of data elements with a timestamp
2. The sensor data flow rate could be data bursts such as data from traffic monitoring or a steady rate such as hourly monitoring of water levels

3. Data streams from sensors are often long term and generate a high volume of data
4. Sensor data streams, in most applications, require real time analysis
5. Data elements may be simple or complex and formed from single/multiple sensors

The demands of these special characteristics of the sensor data stream and the different requirements of sensor applications impose novel challenges when capturing data provenance. The list of these challenges is as follows (Vijayakumar & Plale 2006; Vijayakumar 2007; Wang et al. 2007; Misra et al. 2008):

- Addressing the high data rate: sensors generate data streaming with a high data rate and tiny data elements. General provenance models record the metadata of each data element, but this is not efficient, for stream data from a storage perspective and will result in burdening the system by recording overheads. Therefore, the main process is to identify an effective dataset for collecting provenance, when an interesting change or event occurs, with a reasonable balance between storage and retrieval, efficiency and accuracy.

- Expressing lineage dependencies: in sensor applications, enormous quantities of data need to be processed in order to extract interesting information and a large number of continuous queries running on newly arrived data items. In other words, many data streams are derived from multiple past input streams which themselves could be derived from other streams. This poses a challenge when tracing back the source of dependent streams, as the entire set of data items may not be available. The provenance solution has to be in adequate granularity with meaningful descriptive capabilities in order to express the dependencies between data sets.

- Maintaining relevance: queries executed on sensor data streams, with support for joining data streams and filtering, are typically associated with a lifetime. For the provenance system to be reactive, it has to be able to

trace back the source of the derived data and describe the condition when the processes were applied, even after the query is completed.

- Avoiding redundancy: in many instances, multiple data elements in the stream have invariant primitives such as the time period and value. Consequently, metadata associated with these data elements are invariant. In order to avoid redundant provenance data and storage overhead, it is essential to choose the correct dataset and the critical condition of when to record the metadata. Systems need to find a way to compactly store them with respect to the derivation process.

- Addressing node limitations: such applications involve a large number of sensors that are coordinated to perform a specific measurement. Wireless sensor networks face a number of challenges including the following:
    - Sensor nodes are battery powered with a limited amount of local processing and storage capacity, and wireless communication. Energy efficiency is a most important consideration, since the node has a small and finite source of energy, and needs to last as long as possible in order to avoid frequent battery replacement. The sensor network needs to undertake distributed processing, in order to reduce the communication distance and consequently constrain the power consumption. Low batteries do not often cause fail-stop behaviour of the sensor nodes. Rather, nodes may show Byzantine behaviour at certain low battery states, such as strange sensor readings (Tolle et al. 2005; Ringwald 2006).
    - Communication bandwidth is limited over a wireless connection in a sensor network. Data aggregation and data reduction techniques are used to reduce the communication overhead and also the energy consumption (Gaber 2007).
    - A sensor node has limited processing capability and speed to perform advanced computation tasks.
    - Memory is limited in an on-board sensor node, which restricts some traditional processing.

A stream could experience gaps where no data is present due to loss or corruption of data caused by node limitations. These limitations impose challenges when tracking provenance in a wireless sensor network application.

- Addressing privacy issues: privacy, already a complicated problem in pervasive computing, is also a sensitive issue in provenance systems. Some provenance records may contain confidential information about the ownership chain, or the task performed may be a secret that should not be revealed (Hasan et al. 2007). The ownership history may be associated with privacy information about a person, or a user may perform private actions on his created data. Another example, when gaining access to sensor information in a home, is that details about the inhabitants' private activities can be extracted. Thus, the use of this information has to be strictly controlled in order to deter malicious insiders or outsiders from misusing it. The provenance system needs to be confident of the user's identity and authorization before revealing any information. Developing a mechanism to control access with different levels of granularity and introducing secure provenance information is an important challenge.

- Providing multiple administrative domains: pervasive computing applications may form part of a critical information infrastructure. The owner and the user expect all the services and data to be confidential and trustworthy. Provenance information of such applications may reveal critical information about the owner or the process and actions performed on data. Some users do not want to reveal their actions or information to everyone, only to a highly trusted agent (Hasan et al. 2007). In some cases, the sensitivity of a provenance record and the data it describes are different (Chong et al. 2005): the data is more sensitive than the provenance, or vice versa. Therefore, in order to protect this provenance information and prevent unauthorized access, a selective or differentiated access mechanism is required. In other words, control of the access to a provenance record can be achieved by associating an individual user with

one or more administration domains, depending on what level of access is required.

1.3 Contribution

Provenance is well studied in the field of database and workflow systems, while very little work has focused on pervasive systems. With the special vision of these systems, there are many new requirements for a provenance system. This work is not the first to record provenance information about context data in pervasive computing. Several techniques have been developed with different capabilities and for various purposes, depending on the domain in which they are applied. However, the questions of how, when and where to track provenance are still a significant challenge that needs to be addressed in context systems (Sheng et al. 2008). The main objectives of this work are to find an efficient way of recording the history of the gathered context information and all the calculation rules applied to this information, so that it can be useful, and to find an efficient way to represent it so that access is easy and secure.

The proposed solution, which addresses the above-mentioned key challenges, has to combine low storage and processing overheads with higher descriptive capabilities. Therefore, our first contribution is to define a policy language for recording provenance information based on an event alert or when events of interest occur. The policies specify the different kinds of information that need to be recorded as provenance data and they describe how events are filtered. The provenance is collected relative to the data stream and the processes applied to it, and events associated with data streams and process are specified. Four atomic units have been identified to represent the provenance structure in a relation database: process, stream, dependency, and events associated with streams and process.

The proposed model is mapped to the Open Provenance Model (OPM), which is used as a model for inter-operability systems by exchanging provenance

information, in order to generate an OPM-compliant provenance graph and facilitate sharing provenance data with other systems.

Because of the problem of multiple administrative and privacy information, the second contribution is in defining a policy that provides different levels of access control in order to provide secure information and address privacy issues. The proposed access control model provides finer grained control over exactly which participants can access which details of the provenance information by applying a Role Based Access Control (RBAC) model.

This work explores the proposed provenance model application within the context of monitoring electricity usage to support energy saving, in order to provide the driving impetus for the subsequent design of our solution. Describing the use case scenario and investigating the model definition in different contexts of the application, such as debugging on deployment and auditing for correct operation, are effective methods of clarifying the concept of provenance in the application.

The work involves analysing the challenging issues of identifying the key characteristics and requirements of provenance architecture, and serves to explore some of the design issues. The subject of provenance and its representation are involved in the cost of the collecting process, while the manner in which this information is stored is important to its scalability. Therefore, the main parameters in evaluating this proposed system are collection and storage overheads.

1.4 Thesis road map

This section presents a brief summary of the thesis:

Chapter 2 – Background: This chapter provides background information on provenance and discusses the different approaches in representing, recording, storing and querying. Provenance security is discussed as an important concern, followed by a discussion of the provenance challenge series. Then, related work in

the field of provenance and provenance security is demonstrated. At the end of the chapter, a detailed description of the use case application is presented.

Chapter 3 – Provenance policy language: The focus of chapter 3 is on identifying an event policy language for provenance collection based on Deterministic Finite Automata. It then presents an Open Provenance Model and discuses how our proposed structure model is mapped to it, followed by examples explaining how the proposed language is integrated with the mapped structure.

Chapter 4 – Access control policy language: This chapter presents the need for fine-grained access control over provenance information. It then defines an access control language for the proposed provenance model, which is based on the Role Based Access Control (RBAC) specification language. The final section provides examples explaining how this can be applied to our proposed model.

Chapter 5 – Mapping to implementation: Chapter 5 discusses the experimental requirements for evaluating our proposed model. It also explains how our collecting model presented in chapter 3 and the proposed access control model presented in chapter 4 are implemented.

Chapter 6 – Evaluation: Chapter 6 provides an experimental evaluation of provenance recording and querying performance. The storage overhead is evaluated by numerical analysis.

Chapter 7 – Conclusion and future work: The final chapter summarises this thesis by reiterating a summary of its contribution, followed by a description of future work and closing remarks.

CHAPTER 2 BACKGROUND

The main focus of this chapter is a survey of the literature on the management of provenance, and a discussion of related work on provenance and its access control. At the end of this chapter, we present the use case application that is used in this work.

2.1 Background information

Provenance can be defined in different terms depending on the domain where it is applied. In data base systems, it is the description of the source data and the process by which it arrived at the database (Buneman et al. 2001). In the scientific domain, it is defined as a description of the process in the experiment workflow and notes about the experiment (Greenwood et al. 2003), while in the business domain it is the information that is used to trace the data in the warehouse back from where it was generated (Cui et al. 2003). From a general view, Simmhan et al. (2005b) define provenance data as "information that helps determine the derivation history of a data product, starting from its original sources" (p. 1), and refer the data product or a data set to any form of data such as files, tables and virtual collection. A data set has two important features: the source of the data and the process of transformation of that data source. Manually capturing such information and writing detailed notes has become insufficient

because of its limitation with the increased data volume and the complexity of analysis (Freire et al. 2008). Recently, systematically capturing and managing provenance has received significant attention because of its importance and usage in a wide range of domains and applications. However, few sources are available in the literature of comparing across approaches (Simmhan et al. 2005b). The following sub-sections discuss the different domains for provenance, and the different kinds of data processing architecture in which provenance plays a role.

2.1.1 Motivating domains

Provenance has been used in the scientific field as well as in business. However, the way in which provenance is collected and used differs according to the particular environment (Simmhan et al. 2005b). Below is a discussion of provenance in these two domains:

2.1.1.1 Scientific domain

Recently, with the increasing existence of Grid computing, such as the Large Hadron Collider (LHC), the scientific field has become a more collaborative environment, where data is shared across multiple distributed systems. With the advantage of performing scientific tasks in low organizational boundaries, different issues arise, such as trust and quality when using third party data, and data copyright (Simmhan et al. 2005b). In addition, experiment and laboratory information systems attempt to record and retrieve the details of many related collections of experiments required for sensitive analyses (Bose et al. 2005). Provenance metadata can address some of these concerns.

Provenance has been used in the scientific domain in many forms and for different purposes (Simmhan et al. 2005b). In scientific publications, Digital Object Identifiers (DOIs) cite all data related to the experiment process and the description from which the actual data was produced (Brase 2004). Also, the lineage of a dataset can be used for determining the quality of the dataset and to help the user in deciding whether the data meets the standard requirement of the

application (Clarke et al. 1995). In the manufacturing field, the design of critical components is based on statistical analysis (Romeu 1999), which is needed for recording provenance in order to locate the bad sources of faulty components in cases where the system fails (Simmhan et al. 2005b). As sensornets have been increasingly deployed to share data across sensors on-line, understanding the data flow and its republishing is important in order to track the evolution of data, reproduce results, and detect and correct anomalies (Park et al. 2008).

All the above examples show the importance of having a detailed history in order to determine the veracity and quality of these datasets, and credit their creator.

2.1.1.2 Business domain

In a business environment, it is necessary to work with third-party data and from different parts of the enterprise within a data warehouse. Traditionally, business users work with organized data and usually with trusted parties, which is in contrast to the scientific domain (Simmhan et al. 2005b). However, bad sources could exist and need to be corrected, in order to avoid costly errors (Simmhan et al. 2005b). Therefore, data needs to go through a cleansing and transformation process in order for the relevant information to be identified before it is loaded in to the warehouse (Bernstein et al. 1999).

Lineage information in a warehouse domain means tracing the data back to the source, which could create a problem because data could build upon many layers of earlier data, where data in one layer is derived from data in the layer below (Vassiliadis et al. 1999). On the other hand, lineage information can help in analyzing the data source and exploring its characteristics, in addition to tracing faulty data and correcting it (Simmhan et al. 2005b).

2.1.2 Data processing architectures

Data processing architecture refers to the means by which these processes execute and bring about the transformation of the data. The way in which

provenance is collected differs according to the architecture used for the data processing (Simmhan et al. 2005b), while Bose et al. (2005) use data processing systems to categorize lineage retrieval systems.

2.1.2.1 Service-oriented architecture

Service oriented architecture, which allows services and transformations to be discovered and composed dynamically, has been adopted in both the scientific and business domain (Moreau et al. 2007). These services and transformations could be specified in the form of a workflow such as web or Grid services (Simmhan et al. 2005b). Collecting provenance in the workflow services involves tracing the execution of the workflow and the input and output of each service in the workflow, and this can be coupled with the workflow system and enable straightforward capture or it can be extended (Freire et al. 2008).

Provenance information is formed when the static information of the workflow is combined with the runtime detail. Some of the available lineage systems have been extended to support the case of a dynamic workflow such as Karma (Widom 2005). In some cases, provenance information is collected on behalf of each service provider and client and generates a log of their invocation, which is aggregated to form the provenance for the workflow, such as in Provenance Aware Service Oriented Architecture (PASOA) (Chen et al. 2005).

2.1.2.2 Database architecture

In this architecture, data can be a table, a tuple, an attribute or a pointer to external data, and the update queries and functions that form the data processing are the transformation processes of the data (Simmhan et al. 2005b). The provenance information of the data product is the series of functions and update query requests on the source data.

A particular case for a data processing system is a data warehouse. As discussed earlier, the processes of extracting and transforming data in the warehouse are modelled as queries and user defined functions, which construct

the lineage information. This information can be recorded using annotation or inversion techniques, which are discussed in section 2.2.1. Trio (Widom 2005) is an example of a lineage technique that use a data base architecture, which is discussed in detail in the related work.

Tracing the lineage of a data in a database architecture could face problems when the data source is externally linked and processed outside the database boundaries, or the source is a transparent one such as a federated database, which requires a special technique (Simmhan et al. 2005b).

2.1.2.3 Operating system architecture

An operating system (OS) is a type of command processing architecture where the user interacts with the system through commands entered in a shell interface or batch executed file (Simmhan et al. 2005b). The execution of these commands is the transformation of data products, which is usually logged, by the shell interface, with their associated input and output for debugging (Lanter 1990). Lineage information can be collected from these log files with additional annotation and stored by the data management subsystem (Simmhan et al. 2005b). Detailed information about all system calls and file interactions during a command execution can be captured at the OS level (Freire et al. 2008). This is an advantage of OS architecture, that it does not need any modification and has the ability to transparently capture data and data processes at kernel level (Muniswamy-Reddy et al. 2006), or at user level via the system call tracer (Frew et al. 2008).

2.2 Provenance approaches

Different approaches have been used in the current provenance systems in order to support its requirements. These approaches and their trade-offs can be used to help system developers in making decisions when detecting or developing provenance solutions (Freire et al. 2008). Three major components of provenance management have been identified, and below is a discussion of the different approaches used in each of them.

2.2.1 Representation approach

Different techniques can be used to represent provenance information, and these have implications for the recording cost and the richness of its usage. There are two major approaches to represent this information: annotation and inversion (Simmhan et al. 2005a, Simmhan et al. 2005b).

Annotation is a formal metadata that describes the source data and the process where provenance is pre-computed, and it is called an eager form because the provenance is pre-computed and can be used directly (Bhagwat et al. 2004). It requires a copy of the source data or a link to it and information related to the transformation process. Using a link to the source is a good choice when the source data is large, since it is independent from the size of the source data. It has the advantage of being readily usable and flexible in its richness because it provides the derivation history of the data and the derivation process; on the other hand, the size of the provenance data could be huge (Simmhan et al. 2005a). MyGRID is an example of an annotation system, where the annotated provenance information logs are collected during the execution of a workflow.

The *inversion* method consists of inverting some derivation such as queries and user defined functions to find the original input (Cui et al. 2000). Provenance data in the inversion method needs to be computed before it can be used; therefore it is called a lazy form. The inversion query or the inversion function is operated on the output data to identify the source data. It has the advantage of being attractive and compact because it keeps only the inversion function or query as provenance, and the derived data can be identified using the inverse query or function. However, it is restricted to certain relational queries and not all user-defined functions have this inverse function. In addition, the data provided by the inversion method is limited to identifying the source data that created the derived data (Simmhan et al. 2005b). An example is presented by Cui et al. (2000b). Figure 2.1 shows the content of three tables: store, item and sales in warehouse data with retail store data.

s_id	s_name	city	state
001	Target	PA	CA
002	Target	AL	NY
003	Macy's	SF	CA
004	Macy's	NY	NY

Store Table

i_id	i_name	category
001	Binder	Stationery
002	Pencil	Stationery
003	Skirt	Clothing

Item Table

s_id	i_id	price	num
001	001	4	1000
001	002	1	3000
001	003	30	600
002	001	5	800

Sales Table

Figure 2.1: Tables content

In order to follow the selling of California stores, a materialized view Clif is defined as shown in figure 2.2, and the view table is shown in figure 2.3.

```
CREATE VIEW Calif AS
SELET s_name, i_name, num
FROM store, item, sales
WHERE sales.s_id = store.s_id AND
      sales.i_id = item.i_id AND
      store.state = "CA"
```

Figure 2.2: View Definition

s_name	i_name	num
Target	Binder	1000
Target	Pencil	3000
Target	Skirt	600

Figure 2.3: View Table

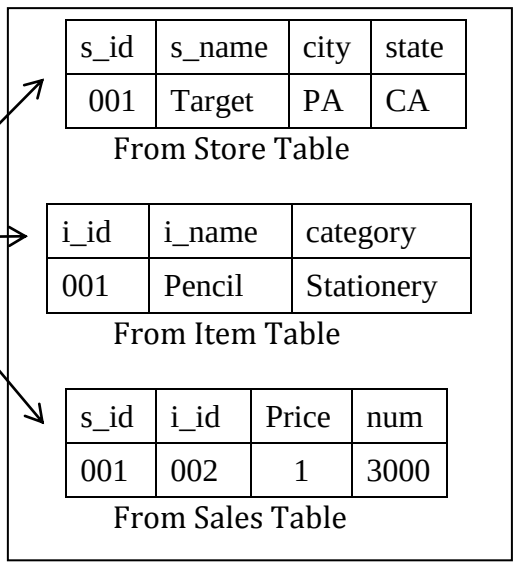


Figure 2.4: Lineage of the tuple

Figure 2.4 shows the lineage of the second tuple in the view table, which indicates that the target store in PA sold 300 pencils at a price of 1 dollar. The main idea of the provenance derivation process using an inversion approach is to re-execute the view definition with the information from the tuple; then the source data item contribution is identified. So, the three base tables (Store, Item and Sales) are joined and form the intermediate table. Then, the query conditions are obtained from the view table definition, the tuple is retrieved from the intermediate table; "state = CA $\wedge$ s_name = Target $\wedge$ i_name = Pencil $\wedge$ num = 3000" is then split into different source tables.

Inversion could be preferred by organizations that have a large number of datasets and the derivation of data is what is most required. However, the problem is that the source data has to be available, otherwise the inverse query or the inverse function cannot be executed (Simmhan et al. 2005b). Trio is one of the systems that uses an inversion approach to determine the source data, and this will be discussed in detail in section 2.5.2.

2.2.2 Recording approach

Provenance data can be recorded about different resources and in various levels of detail, depending on the domain where it is applied. Provenance systems can be classified based on what the provenance is collected about, and the granularity of this information (Simmhan et al. 2005b).

Provenance can be collected about the data product, which called the *data-oriented* model. For example, the transformation applied to the data is considered as a lineage of the data product. When the process is the primary entity for collecting provenance, it called *process-oriented.* In this model, data is recorded as an input or output of the process, and the two models can be used in the same application, depending on the application context (Howe 2002).

The granularity level of provenance collecting depends on the domain requirements. However, the cost of recording provenance can be in reverse to its

granularity and this can play a role in choosing which approach to develop (Simmhan et al. 2005b). Two granularity approaches have been presented:

i- Workflow provenance (coarse grained provenance) is the recording of a complete history of the data derivation. Workflow in the scientific domain is used to perform complex data processing tasks. Tan (2007) defines a workflow as "a program which is an interconnection of computation steps and human-machine interaction steps" (p. 1), and refers a workflow provenance to "the record of the entire history of the derivation of the final output of the workflow" (p. 1). The amount of recorded information may vary, and the complete record can provides a description of how a particular result has been arrived at, by tracking the interaction of programs and the involvement of any external devices such as cameras, sensors and other collecting equipment. It is of considerable value to scientists since it records the research documentation and the complete process of how the experiment was performed, which can be useful for avoiding duplication of effort (Buneman & Tan 2007).

Figure 2.5(a) is a simple example of a workflow specification that describes the process of inference of phylogenetics where a node represents a step with a unique ID and edges donate the flow of data between these steps (Davidson et al. 2007). An example of an execution of the phylogenetics workflow is shown in Figure 2.5(b) where the loop in the workflow is unrolled. Provenance systems, at a coarse-grained level, capture provenance information such as the start and the end of a particular step in the run and corresponding data read and write events while the steps are treated as a black box. This approach does not provide a detailed analysis of the transformation taking place in these steps.

Research on workflow provenance and data provenance have so far been independent. However, in some cases, such as web applications and warehouse systems, where data go though a sequence of transformation and become like a workflow, there is a need for fine provenance recording (Tan 2007). Therefore, Tan (2007) suggests a solution by combining the research effort of workflow

provenance and data provenance toward a uniform approach. Buneman et al. (2007) mention that some research have break the "black-box" assumption in order to give a fine-grained level on the workflow provenance such as the one presented by Bowers et al. (2006).

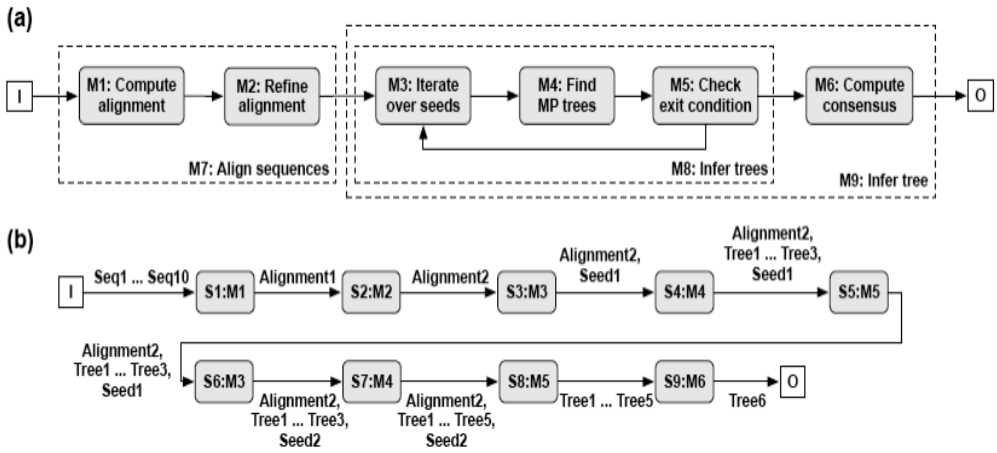


Figure 2.5:

Phylogenetic workflow specification and run (Davidson et al. 2007)

ii- Data provenance (fine grained) records the derivation of part of the resulting dataset instead of the whole process. It has been defines by Tan (2007) as a "detailed account of the derivation of a piece of data that is in the result of a transformation step" (p. 2). It has an advantage when the entire workflow is large and complicated, and the interesting derived data is simple, or when the whole workflow is not available. The particular case is a transformation of an item of data in a database. The following example is discussed in order to illustrate the differences between workflow provenance and data provenance (Buneman & Tan 2007).

The Structured Query Language (SQL) query is considered in two relation tables: Employee (Name, deptid) and Department (id, DName), as shown in Figures 2.6 and 2.7.

Employee

Name	deptid
Kim	D01
John	D02
Susan	D04

Department

Id	DName
D01	CS
D02	math
D03	chim

Figure 2.6: Employee table and Department table

```
Q = SELECT  Employee.Name, Department.DName
    FROM   Employee, Department
    WHERE  Employee.deptid = Department.Id
```

Figure 2.7: SQL query

Q result

Name	DName
Kim	CS
John	math

Figure 2.8: Query Result

The provenance data of the tuple (Kim, CS) in the output consists of the source facts Employee (Kim, D01) and Department (D01, CS) according to the query Q condition. Within fine grained provenance it is essential to distinguish between 'where' and 'why' provenance (Buneman et al. 2001). 'Where' provenance means 'Where does a given piece of data come from?' In other words, it is the identification of the source element where the data is derived from, while 'why' provenance gives the justification or explanation for why it is chosen to be part of the data element in the output. In the example, the 'where' provenance of (Kim) is the Name attribute of the Employee tuple, and (CS) is the DName attribute of the Department tuple. The 'why' provenance is that the tuple result satisfies the WHERE of the query and the condition of the query is agreed, Employee.diptid = D01 = Department.id.

2.2.3 Storage approach

The scalability of any provenance system can be affected by the number of data sets, the level of granularity, the manner in which the data is represented and the geographical distribution of its store. Provenance information can be larger than the data it describes, and annotation may not scale well because of the size of the provenance data to be stored for both granularities. Simmhan et al. (2005b) suggest solving this problem by recording the immediately preceding transformation steps, and then inspecting the complete history from this information. The inversion method scales well except in one case when the source data is geographically distributed, since it has to be fetched before the inverse query or the inverse function can be executed (Simmhan et al. 2005b).

For system scalability, multiple stores may be required (Groth et al 2006). A distributed store can support a large quantity of data, thereby providing the system with flexibility and scalability. Although it has the advantages of eliminating the central point of failure and spreading the demand across multiple stores, it also presents several deployment problems, such as how many provenance stores the system needs, and the need for a mechanism to facilitate the retrieval of this information in order to optimize the query performance.

A wide variety of storage systems have been used in storing provenance information ranging from Web language to tuples in a relational database (Freire et al. 2008). However, most existing systems use relational databases, XML or RDF, for provenance storage (Moreau et al. 2008). Each of these techniques has its own advantages and drawbacks as follows:

i- The relational data model can represent a graph as a list of nodes and edges, while the path is created by joining the list to itself, so the queries based on paths must be translated into relational algebra (Holland et al. 2008). However, it has the advantage of the availability of robust relational database systems and is ready to deploy (Holland et al. 2008). Another advantage is that

it provides centralized, efficient storage that a group of users can share (Freire et al. 2008).

ii- XML supports paths in the provenance graph, but has the shortcoming of being hierarchal yet not naturally representing objects with multiple parents (Holland et al. 2008). Many systems that use the annotation approach adopt XML, where XML is the primary format for their message exchange (Simmhan et al. 2005b). In addition, XML can be stored as files, which do not need additional infrastructure for storing provenance information (Freire et al. 2008).

iii- RDF (Resource Description Framework) supports graphs and paths to model provenance, but it lacks fundamental features and some query supports such as sub-queries and some aggregation functions (Holland et al. 2008). Some of the annotation systems use domain ontologies in language like RDF and OWL (Simmhan et al. 2005b). However, there is an open issue as to whether it scales when handling large provenance stores (Freire et al. 2008).

A provenance system could have the problem of storage size, and may exceed the data it describes. Some solutions have been suggested for reducing the provenance storage overhead, such as archiving the less frequently used information (Cameron 2003) or using a demand supply model, which is based on usefulness for those frequently used (Simmhan et al. 2005b).

2.2.4 Provenance querying approach

Provenance queries are user-tailored queries aimed at obtaining the stored provenance information (Moreau et al. 2007). The full details of captured provenance could be very large. The full provenance of an experiment would include, for example, the provenance information of the process performed on the data input, the provenance of the data input and the result, and the information of any hardware or software used. Therefore, in a provenance management system,

the infrastructure for effectively and efficiently querying provenance data is an important component (Freire et al. 2008).

The high volume of information can make it difficult to query and could return with a large sized result. A solution is proposed by Biton et al. (2008) ,which allows the user to characterize the data item that is of interest to him or her, and the system presents the provenance information according to these preferences.

Commonly, the querying approach used for querying provenance is closely tied to the storage model used in storing the information (Freire et al. 2008). Table 2.1 shows some of the storage approaches used in storing provenance information and the query language supporting each approach. In a formal way, these approaches require the user to write a queries specification, which could be complex for those not familiar with that type of syntax. Some systems such as those described by Scheidegger et al. (2007), address this problem by developing an interface to construct an expressive provenance query that is familiar to the user, and which the latter uses in building workflows.

Storage approach	**Provenance query support**
Relational DataBase Management System (RDMS)	SQL
RDBMS and files	Specialized language
RDBMS	SPARQL for metadata and workflow, SQL for execution log
XML database	XQuery, XPath
Filesystem, Berkeley DB	XQuery, Java query API
RDF	SPATQL
Semistructured data	PQL
Berkeley DB	nq (proprietary query tool)

Table 2.1: Provenance storage approach and query language support

2.3 Provenance security

Provenance information has gained increasing importance in a wide range of critical applications for ascertaining their trust level. Therefore, these kinds of information should be secured and have appropriate access control. Most provenance research efforts have focused on collection, semantic analysis and dissemination of provenance information, while little has been done in the field of its security (Hasan et al. 2008, Braun et al. 2008). Most teams who contributed to the first provenance challenge have not yet considered security (Braun et al. 2008). Therefore, a workshop on provenance, held in 2007, suggested that security is one of the potential applications to investigate (Tan 2007).

Several security issues in a SOA (Service Oriented Architecture) based provenance system have been discussed by Tan et al. (2006), such as enforcing access control over process documentation and the sensitivity of information in p-assertion. However, they argue that the unique security requirements for provenance depend on the architectural and environmental context in which a provenance system operates.

Provenance has particular characteristics, such as the relationship between items, which reveals information about both parties in the relationship, and can be more or less sensitive than the data it describes. Therefore, provenance information may need a different security setting from that for data. Provenance systems can provide trustworthy information by handling confidentiality, integrity and availability (Hasan et al. 2008), which are discussed below:

2.3.1 Confidentiality

Provenance information may contain confidential data about the ownership or the processes that the data go through. Information about the owner may reveal confidential information about a person. For example, if the name appears in the provenance information of a health record held by a third party, which could be a research or analysis organization, it could reveal that a

person is suffering from a specific disease. In another case, the user might perform an action on a specific data item and would want to reveal this information only to the agency. For example, in an assessment of work produced for a competition, each judge rates all the work and wants the assessment to be viewed by the organiser only in order not to influence other assessments and to avoid problems with the contestants. In order to provide privacy, therefore, a provenance system needs to prevent any unauthorized parties from accessing the provenance records and to offer selective or differentiated access mechanisms. In other words, by creating different levels of authorization, this information will be available only to a selected subset or highly trusted parties.

2.3.2 Integrity

In order to achieve full integrity, provenance records have to be resistant to any modification by malicious parties. Many techniques can be used for securing the integrity of the provenance record, such as signatures, checksums or signed hash. However, protecting the provenance chain is more difficult, as a provenance record may pass through multiple domain boundaries.

2.3.3 Availability

In order to ensure information availability, provenance records have to be stored in a secure form of storage and the possibility of deleting these records has to be reduced.

2.4 Provenance challenge series

The growing number and size of collaborating resources in such an open environment is increasingly motivating researchers to focus on provenance. Many systems have been developed with different capabilities and for various purposes. Different systems use different techniques depending on the domain where they are applied. At the International Provenance and Annotation Workshop (IPAW 2006), a discussion on the need for provenance standardization has led the community at the International Provenance and Annotation Workshop (IPAW'06)

to decide on the need to understand the capabilities of these systems and explore their similarities and differences (Provenance Challenge 2011). As a result, they agreed on setting up a "Provenance Challenge" to understand and compare the existing systems.

2.4.1 The first provenance challenge

The first provenance challenge was set up using a simple example workflow inspired from a real experiment in the area of Functional Magnetic Resonance Imaging (Moreau et al. 2008). In addition, a set of core queries was defined in order to show how they could be addressed (Provenance Challenge 2011). Sixteen teams of researchers with different approaches responded and tried to address the same problems, which made the provenance challenge highly successful and created a greater understanding of the available systems. The aim of the challenge was to be more informative than competitive, by making a comparison between these systems in workflow representation, provenance representation and queries result representation.

2.4.2 The second provenance challenge

On the first challenge, queries and their expected results were interpreted differently by different groups because of the absence of a systematic way of comparing the capabilities of the participant systems. Based on that, the second challenge focused on understanding the interoperability of the approaches. That could be by composing the workflow execution system, each system executing a part of the workflow, and then exchanging and sharing the provenance information produced by their different systems (Moreau et al. 2010). Thirteen teams responded to the challenge, and this resulted in discussions about a common data model, which led to the proposal of the Open Provenance Model (OPM) (Moreau et al. 2010), which assumes that the provenance of an object is represented by an annotated causality graph capturing further information pertaining to execution.

2.4.3 The third provenance challenge

The second challenge was followed by the third provenance challenge to evaluate the OPM v1.01 and aimed at exchanging provenance information encoded and answering precise provenance queries (Moreau et al. 2010). The main goals were to identify the weaknesses and strengths of the OPM specification and determine how it represents provenance for different technologies (Provenance Challenge 2011). Fifteen teams participated, and this resulted in several proposals for changes to the OPM specification and the decision to adopt an open source model for the governance of OPM. This resulted in version 1.1 of the Open Provenance Model (which will be discussed in detail in the next chapter).

2.4.4 The fourth provenance challenge

The fourth and last provenance challenge (PC4) was to exploit the Open Provenance Model in a broad end-to-end scenario, and to demonstrate a functionality that can only be achieved by the presence of an interoperable solution for provenance (Provenance Challenge 2011). The fourth challenge started early and terminated early, because of events at the World Wide Consortium Incubator on Provenance. This was followed by the creation of the W3C Provenance Working Group, which continues to pursue the motivation of PC4.

2.5 Related work

The problem of systematically capturing and managing provenance for computations has received significant attention because of its relevance to a wide range of domains and applications. Previous research on provenance had focused on transactional systems, which involved a request-respond style and low data rate. The field of database and workflow systems has been well studied, but little work has been presented in the field of stream processing systems. A survey by Simmhan et al. (2005b) describes a taxonomy they developed to compare and classify five science systems. The main aspect of their taxonomy categorizes provenance systems based on why they recorded provenance, what they described, how they represented and stored the provenance, and ways to

disseminate it. Freire and colleagues (2008) identify three major components of provenance management (capture mechanisms, representation models, and an infrastructure of storage, access and queries) and discuss different approaches used in each of them. Their survey covers the recent literature and the current state of ten provenance systems. A summary of the characteristics of some related systems can be found in Table 2.2.

	Karma	**PASOA**	**Trio**	**Pass**
Application Domain	Weather forecasting	Biology	Generic	File system
Processing Framework	Service Oriented	Service Oriented	Database	Operating system
Representation	Annotation	Annotation	Inversed Query	Annotation
Data/Process Oriented	Process/data	Process	Data	Data
Granularity	Coarse / Fine grained	Coarse-grained	Fine-grained	Fine-grained
Storage	XML	RDBM and File	RDBM	Berkeley DB
Querying	XQuery	XQuery, Java query API	SQL	nq (proprietary query tool)

Table 2.2: Summary of characteristics of related work in provenance techniques

2.5.1 Workflow provenance

Provenance support for workflow-based systems has been conducted for scientific experiments and web oriented workflows. Several provenance-tracking solutions exist to support data provenance in workflow systems (Simmhan et al. 2005a). The details of these techniques vary, depending on the system domain.

Workflow provenance does not consider stream environment characteristics. Data provenance of sensor applications differs from workflow systems in several ways. First, data is often static in a workflow system, or

treated as a static snapshot. Sensor applications deal with live data with high rate feeds and streaming processing. Second, workflow systems use a coarse grain model which focuses on the history of inter-component interaction in the workflow and the input and the output, while sensor applications are needed for fine grain information in order to provide the necessary provenance information for reasoning and explaining the dependency relation between streams such as in health care applications. Lastly, in a scientific workflow, the computations are often heavyweight, while sensor applications are very lightweight and therefore the provenance collecting must be scaled accordingly. Examples of workflow provenance systems are Karma and PASOA.

KARMA

Karma (Simmhan et al. 2008) has been developed to support a dynamic workflow, where the execution path can change rapidly according to external events in a weather forecasting simulation. It is designed for collecting two kinds of provenance: provenance of workflow, and explicit data provenance. Workflow provenance – also known as a workflow trace or process provenance – describes the interaction of services and the process execution. The data provenance provides the derivation history of the output data including the service used and the input data source that generates it. Each service that composes a workflow has its own provenance. The provenance activities are represented as XML notifications between services and server, and are then stored in a relational database.

The Karma service has a provenance-querying interface that provides the essential query primitives to retrieve the provenance graph. Although Karma has a collect process and data provenance, it adopts a coarse-grained model and does not target stream data characteristics; it is dynamic and needs fine-grained provenance. In such applications the provenance needs to be collected for each stream and the process applied to the streams, while with Karma the emphasis is on collecting provenance of the interaction between services and process execution in the workflow.

PROVENANCE AWARE SERVICE ORIENTED ARCHITECTURE (PASOA)

PASOA (Chen et al. 2005) is a provenance infrastructure for recording documentation about the invocation of various web services. It is a service-oriented architecture that identifies different requirements such as verifiability of actors involved in the process, reproducibility of the process and scalability of the provenance system. Actors could be a client who invokes the service, or the service that is invoked. During the workflow execution, interaction provenance and actor provenance are generated. Interaction provenance describes the input and output parameters of the invoked service, and the actor provenance is the metadata about the actor.

In the provenance recording protocol, there are four phases: the negotiation phase, when the actors agree upon a provenance service to record the provenance; the invocation phase, when the service invocation is performed; the provenance recording phase, when the interaction provenance is recorded; and the termination phase, when the protocol is terminated. All interaction assertions in a workflow have the same ActivityID in order to be identified later. The granularity of the provenance collected is at the level of the input and output parameters to the web service. The PASOA provenance server saves the provenance records in a relational database and provides methods to access and update via a web service. Basic queries to retrieve the provenance information are available, such as locating all data that were derived by the service, or validity checking the service input and output.

PASOA uses a process-based provenance approach, which stores the description of the web services that consume and produce data in a given workflow rather than the actual transformation of the dataset. Pervasive applications are not simple transactional applications and mostly depend on changes in the surrounding environment, which needs different approaches to record the provenance of these changes and its dependency information. It needs more than an overview of the interaction process, which cannot provide

information about the derived data, the origin data, and the transformation performed.

2.5.2 Data provenance

Data provenance is a data-oriented model associated with file systems and databases. Work in data provenance has been classified in the overview paper (Tan et al. 2007). The data model collects provenance information of individual data items such as a file or database record. Although data provenance provides rich provenance information, sensor applications place several additional requirements on provenance. First, database systems focus on capturing SQL transformations. Sensor applications need to support arbitrary, external programs not strictly described by SQL. Second, a data provenance system works with a single data provider. A sensor application works in a distributed environment with many data providers. Thirdly, data provenance uses an annotation approach in recording provenance information, which cannot be used directly with a sensor application due to the high volume in stream data. Lastly, database or file systems support the addition, deletion, updating and amending of already existing data information, while sensor applications constantly add new sensor data from live sensors and corresponding transformations. The next part illustrates this with two examples of a data provenance system.

TRIO

Trio (Widom 2005) is a database system that traces lineage information and has data accuracy as an inherent component. Trio supports an inversion model to automatically determine the source data for tuples created by view queries. A view query is a query tree that evaluates from the bottom up: it starts with a leaf operator having tables as input and each successive parent operator taking as input the relation resulting from its child operators (Cui & Widom 2000a). The inverse queries are at the granularity of a tuple, and the lineage information for each tuple includes: the creation timestamp, the derivation type – such as an insert or update query or a user-defined query – and any additional related data. Therefore, Trio is a data oriented provenance scheme

because the lineage is simply the source tuples and the view query that creates the tuple. The dataset used in Trio is an individual data item in the database system, which is persistent. Data stream lineage is different, since it is dynamic and could be in a specific time period.

The lineage table, where the inverse query and the lineage data are stored, can be queried using a construct in the Trio provenance query language (TriQL), which is an SQL-like query language. It supports the querying of both lineage and accuracy information, since Trio also manages the data accuracy and lineage. Any scientific data management can apply such techniques to model data transformation. The inversion method is difficult to adopt in our solution, although it provides many advantages. The arbitrary aspect of the process generated on the data, and the instantiation of the data generated by sensors invalidate the inversion method.

PROVENANCE AWARE STORAGE SYSTEM (PASS)

PASS (Pass n. d., Muniswamy-Reddy et al. 2006) is a storage system that automatically collects and maintains the complete history or ancestry of an item (e.g. a file). It is an operating system base that generates system level provenance. It records information about which program is executed, its inputs, and any new files created as outputs. The provenance is collected about the derived data, the original data and the transformation process if it exists. Provenance collection and management are transparent, as the capture mechanism consists of a set of LINUX Kernel modules (Freire et al. 2008).

The provenance graph is stored as a set of tables in a database, which can be queried using proprietary tools that support a recursive search over a provenance graph. PASS is typically annotation-based provenance and collects the modification history of files such as information on calling application and the file description state. It collects provenance at fine granularity, which leads to a large storage size. However it stores the provenance and the data together in the storage system to ensure that it is not lost. It also has the limitation of being

restricted to a local file system, which cannot be used to track files in a grid environment.

Facilitating the automatic collection of provenance is a common goal with this proposed solution in order to avoid the disadvantage of manually recording provenance information with the high volume of events generated by the sensor data stream. However, the annotation approach that is used by PASS can burden stream systems and cause overheads on the system performance and storage, which leads to defining a different approach.

Source code control and build system

The main purpose of these systems is to provide versioning and building capabilities by tracking changes within the source data and providing version history. However they are actually provenance systems. In these systems, source files are original data and object files are derived data. Their primary goal is versioning and reconstruction of derived objects. There is a significant overlap in functionality between these systems and file systems such as PASS, even though they have different emphasis goals and different design decisions. For example, PASS as a file system maintains the complete provenance about an object's origin, while a build system maintains some description of how a derived object is created (e.g. Make File), and does not explicitly track the dependencies between objects (Seltzer et al. 2005). These systems usually use a mechanism to comment on the changes, in order to find who made certain changes and the reason.

Such systems are widely used in software engineering to manage the different versions of source code, and for process documentation for identifying the provenance of a software application (Gudeet al. 2007). A number of source code control systems exist and manage provenance, such as Concurrent Versioning System (CVS), and subversion (SVN) (Collins-Sussman et al. 2004).

CVS consists of software version control repositories that manage the changes made on documents over time. It records the author and a description and

version of a document in metadata, and holds the information for all file versions and their metadata in the same file. It is an open source client-server architecture, where the server stores the current version of the document and the client can connect to the server and check out the document, work on it, then check in their changes on the document to the server. The client and the server may run on the same machine or connected by LAN or over the internet.

Several clients can work on the same document at the same time, but they each work on their working copy and send their modifications to the server. The CVS server records the user description, the date and the author's name to its log files. The client can review the history of changes, compare versions, or check the historical changes using a date or a version number.

2.5.3 Data stream provenance

Only limited works that focus on provenance in data stream systems and satisfy the unique requirements and constraints of ubiquitous environment applications have been presented. Time value centric (TVC) (Misra et al. 2008, Wang et al. 2007) is a biomedical system for online health care analyses. It creates a model that collects both the provenance of data and its process, and specifies the dependent relationship between an event and the input stream using stream segment level semantics. The main goal of the system is to support scalable automated near-real-time analysis of high volumes of medical sensors. The key driver for this model is to define the time interval window of the sensor data stream and to record the causative relationship between the data event of the output of the stream generated at the output and the dependent input stream within a finite time window for every processing. The time scale used can vary widely depending on the different analysis component; for example, monitoring abnormal weight gain can be determined using a week of weight readings, while arrhythmia patterns may be monitored on an hourly basis.

TVC provides a low overhead approach for capturing dependencies compared to the conventional annotation approach; however, its scalability is

limited due to the need for all elements of all data streams to be in persistent storage in order to derive the set of input causative data in a straightforward process. The common idea between TVC and our solution is the use of a hybrid provenance model in recording the dependencies and lineage of the data event. However, our solution records the provenance information of an event with the time instance that occurs in the stream, while in TVC, the dependencies of the event are recorded using the stream segment semantic. Furthermore, the windowed time interval is not appropriate for our cases when the stream data is not changed for many windows. Consequently, the provenance metadata is invariant, which causes redundancy of information and a waste of storage space. Moreover, our solution allows for capturing the event when it occurs at any time, which cannot be done with a stream segment approach.

Vijayakumar and Plale (2006, 2007) have proposed a system architecture for near-real-time provenance collection in data streams. It focuses on identifying and storing the dependencies' relation among streams rather than the data dependencies for various elements of the stream. Users invoke the provenance services and register the input streams and filters queries, and the system registers the derived stream when the query is submitted. Additional annotation and metadata can be added to the provenance data set by users. Their system captures the provenance history of the streams by encoding the IDs of ancestor streams of a derived stream as a tree. The collecting provenance of each stream or filter is stored in a stack with a time stamp, where the base information is initially collected and then a list is made of any changes in the information. The system has been proposed for a specific domain, which is meteorology forecasting (Lim et al. 2009).

Our model has borrowed the idea of recording the input stream and the output stream of a query, but this information and the provenance information of any event are automatically recorded. We may share the same concept, but the emphasis, goals and the design decisions are different. For example, our model records the event provenance dynamically and does not depend on any previous

process, while in their system, recording the base provenance, which registers the query and streams, is the key for recording the other provenance information during the processing.

2.5.4 Related work on provenance access control

Provenance information has been widely used in critical application areas. Therefore, provenance access control is an important consideration in provenance security (Groth et al. 2006). Although a large number of research activities focus on the management of provenance as mentioned earlier, only a few of them have investigated the area of securing provenance and access control in particular.

Hasan et al. (2007) present research challenges to secure a provenance chain and discuss these challenges to secure each phase in the lifecycle model of provenance recording. They propose and evaluate a cryptographic mechanism for securing document provenance and maintaining the confidentiality and integrity of provenance (Hasan et al. 2009). The provenance information is captured for each change to the document and that information is appended to the provenance chain. The provenance chain is secured and a particular entry can be accessed only by an authorized auditor and cannot be removed from or added to the middle of the chain without detection.

In the scientific workflow provenance, Chebotko et al. (2008) propose a security specification mechanism for provenance and provide a different access granularity level. The authors discuss a framework that outputs a partial view of a workflow which conforms to a set of access permissions. They define the specification for three security levels: port level security for data value produced or consumed by the ports of modules, channel level security for edges between modules, and task level security for all input and output of modules.

A recent piece of research presents a first step towards the formalization of secure provenance by developing a unifying model that identifies security properties for the static provenance, and describes a single run or behaviour of a system (Cheney 2011). The author has developed a high level and generic framework for provenance by identifying some commonalities and general properties of systems in domains, based on automata, database queries and workflow provenance graphs, and regardless of the details of a particular system. The proposed formalization of security properties of provenance include disclosure, which means ensuring that a provenance query is always answerable by using provenance views, and obfuscation, which means ensuring a provenance query can never be answered by provenance views. Provenance view is a function in provenance information that hides some of the trace information.

2.6 Provenance use case

This section will discuss the scenario of implementing a system for the monitoring and feedback of electrical energy usage. The application will consider shared spaces such as an office environment, and address specific issues around sustainable energy usage in these spaces through better practice in the use of electrical devices. Shared spaces are inhabited by a number of people who contribute to energy consumption by using shared resources such as air-conditioning or ceiling lights, as well as their individual use. Therefore, the focus is on collecting individual contributions towards responsibility for energy consumption in a shared space. It allows them to track their usage, and provides feedback to the individuals in order to support energy saving. Consumers require real-time feedback in order to influence their behaviour and consumption habits and reduce their energy demand. It extends previous work in treating the household and all its members as a unit and reports the energy usage for the unit of a household.

Weiss & Guinard (2010) classify the work in the field of residential energy monitoring and consumption feedback by examining the type of sensor that is used to acquire the consumption data. The first area classifies systems that use a

single sensor, which is attached to the home fuse box, to obtain information on the entire energy consumption of a household. Several commercial products are available, such as Wattson (DIY Kyoto 2010) and Power Cost Monitor (Blue Line 2010), which consist of a central device with a display providing the feedback. However, these products require a complex installation and are not able to provide feedback on the consumption by a single device. Other systems, such as the approach by Lam et al. (2007), use a single sensor situated at the electricity meter to acquire the consumption information and try to further apportion the total consumption, using a statistical signature analysis and detection algorithm, in order to detect which appliances are currently running. The approach of Weiss et al. (2009) focuses more on the design of the user interface. They have developed an interactive system that provides instant feedback on the energy usage by using a portable user interface on a mobile phone and a smart electricity meter.

The second area includes approaches that use an electrical current sensor, which is installed in-line with each appliance, or deploy multiple sensors throughout the household. Commercial examples of products are Kill-a-Watt (P2 International 2010) and the SmartLinc INSTEON Central Controller (SmartLabs 2010). These products are easy to deploy, but do not concentrate on the electricity feedback. Other approaches focus on developing a system that allows for monitoring energy consumption at device level and the total load. Jiang et al. (2009) have developed a wireless sensor network that measures the power consumption at the outlet and transmits the reading to an application tier in order to store it in a database. A similar system is that of Weiss et al. (2010), which consists of a three-layer architecture and uses a wireless sensor for each outlet, but this system uses off-the shelf products (Plogg sensors) and is easily extendable.

Our use case is similar to the last two systems, which use a sensor for each outlet and transmit the reading to the application to be stored in a database and to be displayed to the user. An example scenario of energy saving is in an office

shared by several users. Sensors are assigned to individual electrical plugs and switches for each user; they collect details of the voltage and current used. The collected context data from sensors are converted to the AC current used then distributed to the user in order to display his/her energy usage, and are also sent to the system database. Displaying the usage by individuals within a shared space over the same time of usage is a way to enable them to understand the contribution of their behaviour to collective energy consumption. Hence, the interactive element and immediacy can lead to a higher level of savings. Additional usage data can be retrieved from the database, such as the previous day's usage or the total usage for the current day. In many cases, a set of tasks is executed on the incoming data, such as a query computational process. Storage and access infrastructure must enable detailed analysis and display personal or group consumption or relative consumption between spaces.

2.6.1 Application architecture

The architecture of this implemented application is based on three independent components (Figure 2.5): a WSN query service component, a WSN query execution engine, and sensors connected to the monitoring plug and switches in the shared spaces.

The WSN query service provides a user interface for building a real-time WSN queries request by selecting the attributes and conditions required, and sends them to the execution engine where they are executed. After the query is executed, the result is stored in the database and is sent back to the WSN query service in order to be displayed to the user.

In a normal operation, the system runs a long-lasting WSN query for collecting the readings from all sensors. The routing table is populated in order to obtain access to sensors. The number of sensors is adapted according to the number of sensors registered. The reading is collected each second; it has a time-stamp and is given a sequence number, and is then sent to the database and to the user. A user can query data from sensors or from the database. A query can be

built by selecting the attributes and conditions required, and then submitted. The WSN query planner accepts the query, translates it and sends it to the execution engine.

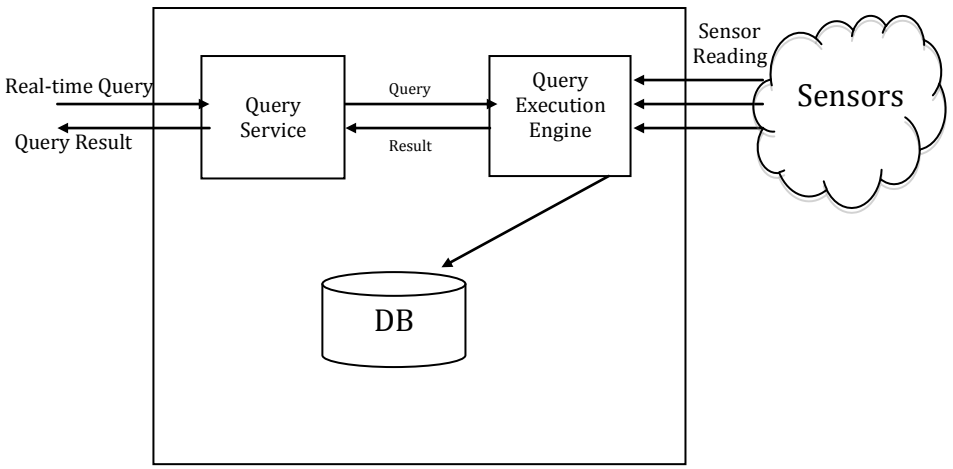


Figure 2.5: System Architecture

The query is executed in the execution engine and the result is collected. The result could be from the sensor or from the database, according to the exact query specification. The execution engine stores the result in the database and sends it back to the WSN query planner in order to be displayed to the user.

2.6.2 Provenance component

In order to enhance the application with the provenance data, the necessary lineage need to be kept track of. This information might contain data about the authority of the sensor row data, or the authority collecting the data, the time, location, the sensor reading and the process applied to it, such as aggregation. Additional information about the sensor characteristics, which describe the situation, is important, such as lifetime and sensor calibration information.

Storing sufficient metadata can provide critical information, which can be used for a variety of purposes. In such an energy monitoring system, there are

many places in the life cycle when provenance may be useful. In our case study we are going to focus on two examples:

1. Debugging on deployment, when detailed provenance information may be required.
2. Auditing for correct operation, when the administrator may wish to set up random probes that are checked against other measurements.

In later chapters, we will show how the queries required for provenance monitoring can be specified in our language (at the end of Chapter 3), and show how they can be implemented in the provenance management system (in Chapter 5).

2.6.3 Addressing the challenges of provenance

This section will describe the proposed model, which will be explained in detail in Chapter 3, from the perspective of how it addresses the challenge, which rise with stream data and a sensor environment. The discussion will describe the provenance approaches used by this proposed solution:

Representation of provenance

Data provenance can be represented by using two methods: annotation and inversion. Although the inversion method – which uses the process property to invert the derivation process – has the advantage of minimal storage, it is poorly suited to our solution, due to the arbitrary aspect of the process generated on the data, and the source data could be data generated by the sensor, which is instant data.

The annotation method, which keeps the metadata of each individual data element, is being used with some restrictions in order to address the problem of high storage and process overheads. In our proposed model, the provenance data is collected for each event alert, which mean that event of interest is the dataset

for collecting provenance and not the data element in the stream, in order to address the challenge of high data rate and avoid the recording overhead.

Recording approach

Provenance can be collected about data in a specific way, which is called a data oriented model, while the process-oriented model is one that collects the provenance about a deriving process. In the most pervasive application, data goes through many processes in order to produce new useful data; therefore, both the data and the process applied require provenance tracking. Our proposed model needs to combine both types in order to satisfy the application requirements and provide an adequate level of detail. Provenance has to be collected about the process performed on the data, and about the data source used in the process and the data out put from the process.

Granularity

Coarse-grained records provide an overview of the processing, but not enough for tracking all the information. Fine grain is a good choice for providing much detail of each processing data. The cost of collecting and storing provenance is inversely proportional to its granularity. The proposed system manages the fine granularity recording and storage overheads by collecting provenance only when appropriate attributes of interest change or when a specific condition is satisfied.

Storage

A relational database is used as a technique for storing provenance information, for the advantages mentioned previously. Provenance information is shredded and stored as tuples in a relational table which can be queried by SQL. A provenance query should have the ability to reconstruct the whole chain from these tables.

The base information of any derived stream is all input streams and their transformation. With this information stored in a dependency table it will be

possible to trace back to the source stream and the transformation that it goes through and identify the dependency relationship between these streams and the condition of the processing at different times. In order to avoid storage problems with a high data rate, the computation of provenance is invoked only when an event of interest occurs.

2.7 Summary

This chapter was divided into three parts. The first part provided background information on provenance by discussing its different approaches, its challenge series and its security. Provenance information can be represented by annotation or inversion approaches. Two levels of granularity approaches are used for provenance recording: workflow provenance (coarse grained) and data provenance (fine grained). Most existing provenance systems use relational databases, XML or RDF, for provenance storage. Four provenance challenges were set up in order to understand the existing systems and establish the interoperability of these systems, which result in an Open Provenance Model. Provenance security presents confidentiality, integrity and availability as important issues for provenance to be trustworthy and emphasises the need for fine-grained access control.

The second part provided a discussion that related the work of provenance in the fields of database, workflow and data stream systems. Finally, it discussed the related work on provenance security and access control.

The third part presented the use case used in this work by demonstrating the system architecture. Then it presented the system provenance component and discussed how it addressed the challenges in different provenance approaches.

CHAPTER 3 PROVENANCE POLICY LANGUAGE

In this chapter, we implement an event definition language for provenance collection based on Deterministic Finite Automata. Then, the proposed structured graph model is mapped to the Open Provenance Model. Finally, we present examples to connect the collection language to the structure model.

3.1 Collection policy

In most pervasive computing applications, it is necessary to combine and query the readings produced by a collection of sensors. The middleware needs to support access and query streaming sensor data, as sensors deliver data in streams. A combination of the two recording approaches is needed, therefore any proposed model should be designed to record the provenance of the process, which could be declarative queries or an application code that is executed on streams, and the provenance of a data stream. Streams could be base streams or derived streams. The former are generated by sensors, and in our use case they typify the electricity usage that the sensor measures. The latter are streams that are produced by executing real-time queries on a base stream or other derived stream; an example is the total usage of electricity measured by sensors in a room.

A provenance architecture must be deployable in many different contexts that support application preferences (Groth et al. 2006). Provenance services in

any application may be designated to record all or some of the provenance information. The different levels of recording detail should be stated in a policy (Groth et al. 2006). Policies are statements of goals for the behaviour of a system (Heather et al. 1994), and provide a means to control the system processing. The conditions of policies depend on environmental or contextual information.

Definition: Recording policy: *A recording policy should specify the various different kinds of information that a recording service has to record as provenance information.*

In order to combine addressing the challenges of provenance collection and storage overheads, which were mentioned in the first chapter, with the application needs, the collection is based on an event alert, or when an event of interest occurs. An event is considered as a dataset for provenance collecting instead of a data element and it can be used to specify when to start collecting.

Definition: Event: *An event is a happening that has an effect on an artifact or a process specified by a condition to enable event filtering, and has a description of when and how it happens.*

Events of interest include any of the following, which have been extracted and modified from the solution proposed in Vijayakumar (2006), Wang et al. (2007), Park (2008), and Chen (2005):

- A WSN query is started or finished: when a user requests a WSN query, the provenance service starts to record all the processes and inputs of the WSN query, until the end of the WSN query with the output.
- The stream is started or finished: when a WSN query on sensor data starts or finishes, or the output stream data starts or finishes, which are associated with the WSN query starting and finishing time.
- An execution plan of a WSN query is changed: in the case where input is missing, or a WSN query is interrupted.
- The reading value is changed: in our use case, the usage of electricity has no prior knowledge of a signature value, since different machines

consume different watt voltages. The detection depends on a trigger condition, when the current reading is above or below the previous one (by considering the noise level of the sensor).

- A data transmission is changed: in the case of lost or strange packets. A strange packet could be an abnormally small or large sensor reading (Ringwald et al. 2006). An example of a strange sensor reading as reported in temperature value > 100 °c (Tolle et al. 2005). The current sensor, which is used in the evaluation experiment, provides a 0 to 5 reading value. Any reading above or below this range is a strange packet.
- Problems with sensor: the case of adding or removing causes fewer changes than failure, since a failed sensor can occur suddenly, which can effect changes in streams and queries.

These events are a happening that affects a process (WSN query) or are associated with a data stream, each with its own condition that may happen. In each case, different kinds of provenance information need to be recorded. The resulting provenance graph describes the historical events related to the process such as the process start and process end, and events that happened in the data stream such as the start of the stream, a change in the reading value, or the end of the stream. There have been a number of general-purpose event query languages, such as CAYUGA (Demers et al. 2007) and SASE (Gyllstrom et al. 2007), which filter and correlate the stream data for pattern detection and transform it into events. However, we have chosen to implement a small event definition language of our own for provenance collection. Our principal reason is one of scale; in all our examples, we have yet to define a composite event (e.g. query start, finish, reading increase and decrease) in our scenarios, which cannot be represented by simple Deterministic Finite Automata (DFA). We have therefore crafted a small domain specific language to define composite events and their attributes, and simple DFAs, which generate the events (Appendix 1, Appendix 2). This is then used in the provenance filter to maintain state machines and record these events as provenance information in such systems.

A DFA consists of 5-tuple $\mathbf{M} = (Q, \Sigma, \delta, q, F)$ where:

$\mathbf{Q}$ is a finite set of states

Σ is the input alphabet

$\delta : \mathbf{Q} \times \Sigma \rightarrow \mathbf{Q}$ is a transitive function

$\mathbf{q}$ is the start state

$\mathbf{F} \subset \mathbf{Q}$ is the set of acceptance states.

In our case, the input alphabet (Σ) is a finite set of arbitrary happenings with state transition controlled using the event conditions as a predicate. Each automaton state is assigned a fixed related event, and the acceptance states are states for events of interest that are required for provenance recording. Our model automata operate as follows. Suppose an automaton instance is in state **q**, when a happening occurred and that satisfies an event condition in state **q1**, then the machine deterministically transitions to that state. Edges that are derived from a predicate that filters the acceptance state are called *filter edges, and the associated predicates filter predicates.*

The recording policy is based on the notion of *event occurred* as specified by a condition. Therefore, when the filter predicate has occurred and the automaton is in the acceptance state, which means the condition of event of interest is fulfilled, this requires a certain obligation. In other words, for all events that affect the stream or the process, if the event of interest condition is fulfilled then the provenance information of that event should be recorded.

$$\forall \textbf{ (streamID or processID, event): event.condition = true } \Rightarrow \textbf{[record provenance]} \quad (3.1)$$

The following is a detailed explanation of events and their conditions, which indicate when and what should be recorded.

3.1.1 Process events

In many cases, a set of tasks is executed on the incoming data such as a WSN query's computational process. The provenance system, which is a computer-based representation of provenance, has to record the process performed on these real time data and their historical information by capturing events associated with queries and the deriving process. The base information of any process is all the input streams, the output streams and the execution status.

3.1.1.1 WSN Query start and finish

One of the aspects of recording provenance in any pervasive application is for debugging compliance. Many applications require proper documentation and data logs to trace the lineage of data and optimize the derivation process. In a pervasive application, in order to obtain data from a sensor, a request is sent to sensors and then sensor readings are logged according to any conditions specified in the request. Therefore, a query start and finish are events that need to be recorded. In order to document this process, the provenance service has to record all the information about the request (query) and any process required by the query or data imposed as sequences of a query. To model this policy, we assume the following finite sets (Bauer et al. 2009): Q for queries, P for processes, D for data, and E for events.

A query is a request sent by a user to a sensor or set of sensors. The query can be to obtain the reading, the total reading or the average. The condition for a query start event is for the user to send a query. For example, when a query is sent by the user to obtain the average, the event concerned is the query start. The information needed for recording is the query information, base stream log corresponding to the query, and the result. The base stream is bounded by the query time. In other words, the new stream starts when the query starts and it ends when the query execution is completed (duration time has expired), or when the query has been interrupted (the user stops the query). The level of granularity of the provenance information is linked to the usefulness of the provenance to the application. The simplest form (coarse grained approach) of

provenance in this example is recording the information that is externally observable for calculating the average from the input data by recording the input, the main process and the output. However, the 'zoom in' on the process will record extra provenance information of what is actually happening inside each process, which can provide detailed information and allow for tracking each data and process. The coarse grained approach can be represented as follows:

$$\forall \textbf{(query, event): event = start} \wedge \textbf{start.condition = true} \Rightarrow$$
$$\textbf{[record (q)} \wedge \textbf{\textit{record}} \textbf{ (data}_1\textbf{, …, data}_n\textbf{)]} \qquad (3.2)$$

The policy specifies that for all events associated with the query, if the event is a query start and its condition is satisfied, then that implies recording the provenance information of the query and recording the provenance information of the data log for the base stream (input) and the derived stream (output result). The input data log depends on the number of sensors involved in the query.

A query execution includes sequences of process such as summation and division, to obtain the output result data. The zoom-in form (fine-grained approach) of collecting provenance information requires recording the detail of these processes and their input and output data. Therefore, the policy will be as follows:

$$\forall \textbf{(query, event): event = start} \wedge \textbf{start.condition = true} \Rightarrow$$
$$\textbf{[}\textbf{\textit{record}}\textbf{ (query)} \wedge \textbf{\textit{record}}\textbf{(process}_1\textbf{, …, process}_n\textbf{)} \wedge$$
$$\textbf{\textit{record}}\textbf{(data}_1\textbf{, …, data}_n\textbf{)]} \qquad (3.3)$$

As a query is a kind of process, these two can be integrated into one:

$$\mathbf{Q \in P}$$

$$\therefore \quad \forall(process, event): event = start \wedge start.condition - true \Rightarrow [record(process_1, \ldots, process_n) \wedge record\ (data_1, \ldots, data_n] \qquad (3.4)$$

As mentioned above, the query finish is an event that requires provenance information recording, which in turn requires recording the end time of the query and the data stream associated with the query. The condition for a query-finished event is when the query time duration has expired or the query is stopped by any interruption condition. This can be specified by:

$$\forall(process, event): event = finish \wedge finish.condition - true \Rightarrow [record(query.endtime) \wedge record(data_1.endtime, .., data_n.endtime)] \qquad (3.5)$$

For the fine-grained approach, however, it can be expressed by:

$$\forall(process, event): event = finish \wedge finish.condition - true \Rightarrow [record(process_1.endtime, .., process_n.endtime) \wedge record(data_1.endtime, .., data_n.endtime)] \qquad (3.6)$$

3.1.1.2 Changes in the WSN query execution

During the execution, the query can be interrupted by changes such as the user stopping the query before the intended end time, or one of the input data associated in the query is missing (the sensor is missing), which leads to ending the query. These changes are considered as events which need to be recorded as reasons for the forced end of the query.

$$\forall(process, event): event = interrupted \Rightarrow [record\ event] \qquad (3.7)$$

3.1.1.3 Automaton for process events

To illustrate how these events are automated in our model, let the finite set of states be those states assigned to events associated with the WSN query, while the acceptance states are: query is started, query is finished, and query is interrupted. Respectively, the predicate conditions are: query is submitted, query time has expired and query is stopped or input is missing. At the acceptance state, for each event the provenance information needs to be recorded as indicated in the above expression.

Q = {Query is created, Query is started, Query is executed, Query is interrupted, Query is finished}

Σ = {Submitted, Sent, Time expired, Input missing, Query stopped}

Query is created is the start state

F = {Query is started, Query is interrupted, query is finished}

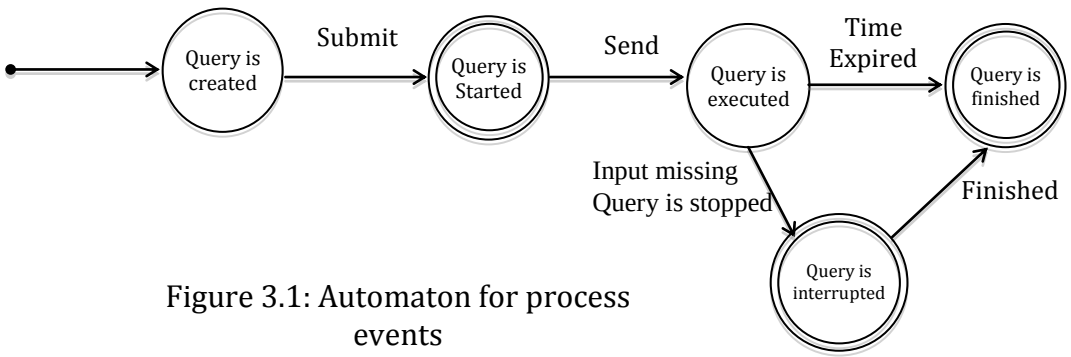


Figure 3.1: Automaton for process events

As shown in figure 3.1, when a user builds a WSN query:

IF query is submitted **THEN** state = Query is started

ELSE state = Query is created

At the state Query is started, the query is sent to the execution engine where it is executed. During the execution:

IF (user send a stop command **OR** input is missing) **THEN** state = Query is interrupted

ELSE state = Query is executed

IF (state == Query is interrupted **OR** Query time expired) **THEN** state = Query is finished

3.1.2 Stream events

A sensor data stream is an indefinite sequence of time ordered readings:

$R = < r_1, r_2, , \ldots, r_{n-1}, r_n >$

where

$r_1.timestamp < r_2.timestamp <\ldots< r_{n-1}.timestamp < r_n.timestamp$

Collecting the provenance of the sensor readings stream is useful for data quality and for auditing. However, in order to address the high data rate and process overhead challenge, the provenance is collected corresponding to events associated with the data stream, which specify changes happening to the data stream. This stream can be affected by different changes such as changes in the reading value, a missing reading, or where the reading is out of range. A significant change in the reading value could be an important event which needs to be recorded, since it means a change in the sensed environment, and could be useful information for finding causality and for informational purposes. While recording missing or changing values as an event can provide the necessary information for evaluating the quality level of each stream, an event is not just a digital representation; it could be a physical embodiment in a physical object such as sensor failure, which can give a null reading. The collection policy is expressed by:

$$\forall \textbf{ (streamID, event): event.condition = true } \Rightarrow \textbf{[record event]} \quad (3.8)$$

3.1.2.1 Reading increased and decreased

When the current reading is above the previous reading, then the event is the reading is increased, and when the current reading is below the previous reading, then the event is the reading is decreased.

$IF\ r_n.value > r_{n-1}.value$

$THEN\ event = Increase$

$IF\ r_n.value < r_{n-1}.value$

THEN event = Decrease

So, when these conditions are satisfied, the action is to record the information of these events as provenance information of the stream, where these two events occur. To discuss this, let M be the DFA given by:

M = (Q, ∑, δ, Reading, F)

Q = {Reading, Increase, Decrease}

Reading is the start point

Increase is the state when r_n.value > r_{n-1}.value

Decrease is the state when r_n.value < r_{n-1}.value

∑ is the input alphabet = { r_n.value > r_{n-1}.value (>), r_n.value < r_{n-1}.value (<), r_n.value = r_{n-1}.value (=) }

F = {Increased, Decreased}

The transaction function $\delta : Q \times \Sigma \rightarrow Q$ is given by this table:

q	δ (q, >)	δ (q, <)	δ (q, =)
Reading	Increase	Decrease	Reading
Increase	Increase	Decrease	Reading
Decreased	Increase	Decrease	Reading

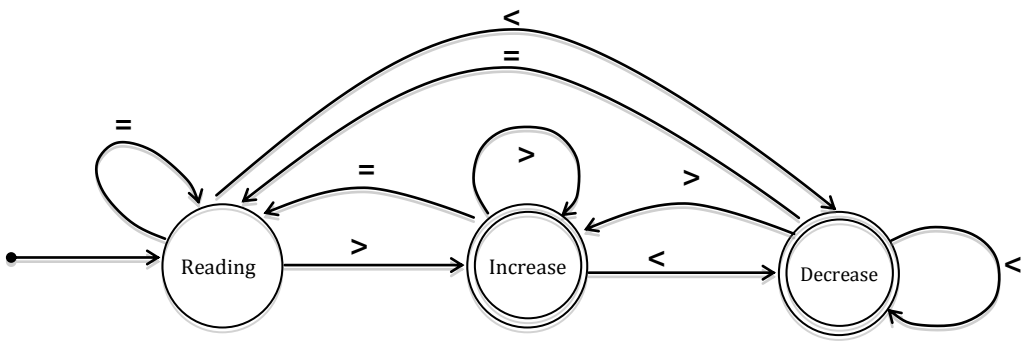


Figure 3.2: Automaton for increased and decreased events

The detection starts when the second reading arrives.

r_n.value = the second reading

r_{n-1}.value = The first reading

Start detection

IF r_n.value = r_{n-1}.value **THEN** move to Reading state

IF r_n.value > r_{n-1}.value **THEN** move to Increase state

IF r_n.value < r_{n-1}.value **THEN** move to decrease state

r_{n-1}.value = r_n.value

r_n.value = next reading

End Detection

The detection process will go in a loop until the end of the stream.

3.1.2.2 Null reading

The reading could be null when either the sensor or the connection has failed. Sensor failed is an event that affects the streaming of the sensor reading and could be followed by a normal reading when the problem is resolved.

IF (r_n.value == Null)

THEN event = Failed

IF (r_n.value = Reading && r_{n-1}.value == Null)

THEN event = Back

These changes in the stream flow should be recorded as provenance information of the data stream. To automate these events, let M be the DFA given by:

M = (Q2, $\sum$, δ2, Reading, F2)

Q = {Reading, Null, Back, Stationary}

Reading is the start point

Failed is the state when r.value = Null

Back is the state when the readings back to normal after a Null reading r.value = Normal

Stationary is the state when the reading is not changed and is still null.

$\sum$ = {Null, Reading}

F = {Increased, Decreased, Failed, Back}

The transaction function $\delta: Q \times \sum \rightarrow Q$ is given by this table:

q	δ (q, Null)	δ (q, Reading)
Reading	Failed	Reading
Failed	Stationary	Back
Back	Failed	Reading
Stationary	Stationary	Back

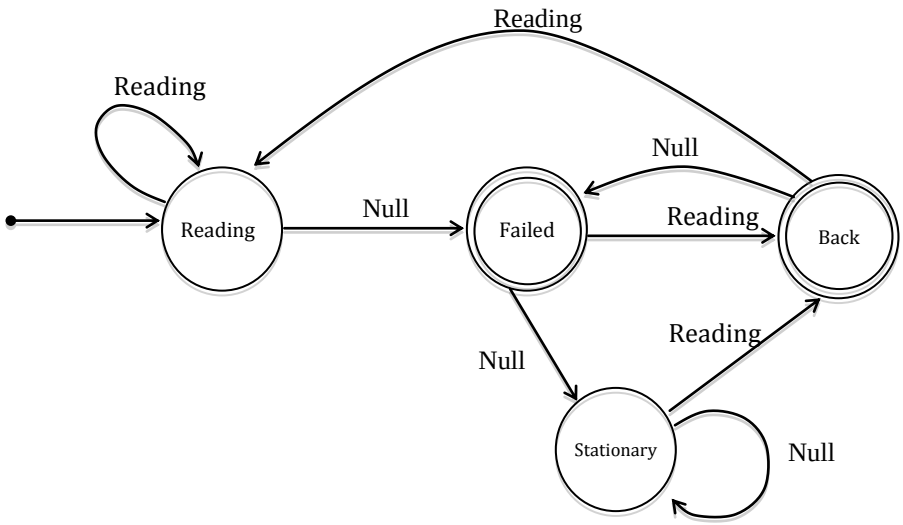


Figure 3.3: Automaton for Null events

When a null reading is detected, the transition is to Failed state, where the event needs to be recorded. However, if the next reading is still null, the transition moves to Stationary, which is not an acceptance state and does not require recording. Only when a normal reading arrives will the state be Back and need to be recorded. The reason for the Stationary state is to record only when the start failed and when it ended.

3.1.3 Extended policy

As discussed earlier, the proposed collection model is based on an event alert, and the policy definition for each event has been explained. Now, we want to extend our policy capture language by defining policies, which match simple predicates on streams, process or user to indicate and specify which events

should be written to the event log. This means that provenance is only recorded when selected events of interest occur in selected processes or streams.

Example: Create a policy to collect an increased and decreased event of streams belonging to user1 for 1 hour after 1pm. The increased and decreased events are of interest only when they occur in streams generated by sensors belonging to user1, and should be documented in the event log as provenance information of those streams.

The example indicates that provenance services start to apply the two policies for detecting and recording these two events if the condition is satisfied, which is that the owner of the streams is user1.

```
Policy  NewPolicy {
When stream.owner==user1, $time > 13:00, $time < 14:00

} Capture IncreaseEvent, DecreaseEvent

event DecreaseEvent {
  Long change;
  }  ReadingEvent[n].value  <  ReadingEvent[n-1]  =>
  DecreaseEvent.change  =  ReadingEvent[n].value  -
  ReadingEvent[n-1].value

event IncreaseEvent {
  Long change;
  }  ReadingEvent[n].value  <  ReadingEvent[n-1]  =>
  IncreaseEvent.change  =  ReadingEvent[n].value  -
  ReadingEvent[n-1].value
```

To achieve this end, we need a policy attribute. Attributes are used to specify policies that can be expressed by value (true or false). Thus, they can also be used to carry an attribute value (true) over the validation of the policy until the attribute is changed to false when the condition no longer exists.

Definition: **Attribute**: *An attribute is a characteristic that specifies the valid policies.*

An attribute is defined as 3-tuples with the following syntax:

Attribute (ID, PName, Value)

Where:

ID is the unique identifier of the policy

PName is the name of the policy

Value is its assigned value

The value of the attribute is assigned according to the policy rules defined by the creator of the policy. When the attribute is assigned to true, the policy condition is checked, and if it is met, then the obligated action is performed until the attribute value is changed to false when the stop condition is met. This can be expressed by:

$$\forall \text{ (policy, true): policy.condition =true} \wedge \text{ stop.condition = false } \Rightarrow \text{ [event.filter]} \quad (3.9)$$

In the example, the assignment rule is required to validate the policy when it is created. Then the stream is checked, and if the stream generated by the sensor belongs to user1, the end time is checked. If the Duration <1hour, then the stop condition is still false, and this point is an integration point between the current policy and the recording policy of the two events. The server will switch from the current policy to check the condition of detecting an increased or decreased event. To use DFA to define this extended policy:

M = (Q, $\sum$, δ, start, F)

Q = {Create, Policy True, Policy Condition, Stop Condition, Policy False, Event Policy}

Create is the start point

Policy True is the state when the policy attribute is assigned to true

Policy Condition is the state to check the policy condition (stream owner)

Stop Condition is the state to check the stop condition (t < 1 hour)

Policy False is the state when the policy attribute is assigned to false

Event Policy is the state to switch to the event policy

$\sum$ = {T, F}

F = {Event Policy}

The transaction function $\delta : Q \times \sum \rightarrow Q_3$ is given by this table:

Q	δ (q, T)	δ (q, F)
Policy True	Policy Condition	Create
Policy Condition	Stop Condition	Policy Condition
Stop Condition	Policy False	Event policy
Policy False	Create	Create
Event policy	Policy Condition	Policy Condition

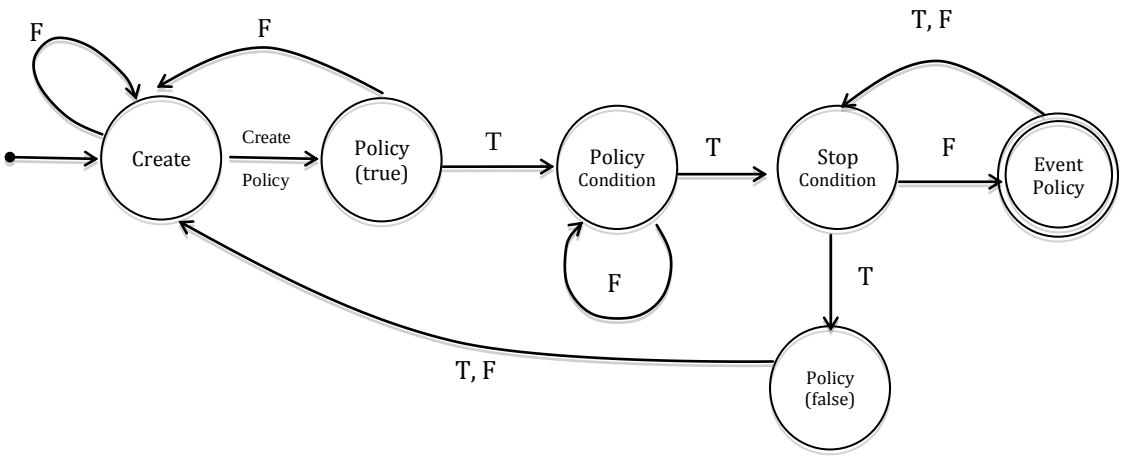


Figure 3.4: Automaton for extended policy

The extended policy can be used also to capture events associated with queries by mapping these queries to processID, streamID or userID. For instance, to activate a policy to capture the start and finished event of queries sent by user1.

3.2 Mapping to Open Provenance Model

The Open Provenance Model (OPM) (Moreau et al. 2010) has recently been proposed as a standardized representation of historical provenance for workflow in the e-science community. OPM v1.1 was the result of a series of proposals which were publicly reviewed and put to the vote, after using version 1.01 in the third provenance challenge to exchange provenance information and answer precise provenance queries. OPM was designed to meet the following different requirements (Moreau et al. 2010):

- Exchanging provenance information between systems based on a shared provenance model
- Building shared tools that operate on such a model
- Defining provenance in a precise and technology-agnostic manner
- Supporting a digital representation of provenance for any object, whether produced by a computer system or not
- Allowing for multiple levels of description to coexist
- Defining a core set of rules that identify the valid inferences on provenance representation

OPM defines the general standard for provenance information and consists of a directed graph expressing the causal dependencies. The graph consists of nodes, dependencies and roles.

3.2.1 Nodes

OPM is based on three primary nodes: process, artifact and agent, which are defined as (Moreau et al. 2010):

Definition: Artifact: *Immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system.*

Definition: Process: *Action or series of actions performed on or caused by an artifact, and resulting in a new artifact.*

Definition: Agent: *contextual entity acting as a catalyst of a process, enabling, facilitating, controlling or affecting its execution*.

Figure 3.5: Graphical representation of OPM entities

OPM represents a historical provenance graph, which means it describes processes that occurred in the past or are currently running or that explain the dependencies between artifacts in the past, and are not supporting the state of artifacts in the future or the activities of a future process. OPM introduces a graphical notation for a provenance graph in order to facilitate understanding and provide visual representation.

3.2.2 Dependencies

The graphical representation of the OPM graph also describes the causal dependencies between these entities, and is represented by an edge. Causal relationship is defined as follows (Moreau et al. 2010):

Definition: Causal Relationship. *A causal relationship is represented by an arc and denotes the presence of a causal dependency between the source of the arc (the effect) and the destination of the arc (the cause).*

OPM adopts the following five causal dependencies:

Definition: Artifact Used by a Process: *In a graph, connecting a process to an artifact by a "UsedBy" edge is intended to indicate that the process required the*

availability of the artifact to complete its execution. When several artifacts are connected to the same process by multiple "UsedBy" edges, all of them were required for the process to be completed.

Definition: Artifacts Generated by Processes: *In a graph, connecting an artifact to a process by an edge "wasGeneratedBy" is intended to mean that the process was required to initiate its execution for the artifact to be generated. When several artifacts are connected to the same process by multiple "wasGeneratedBy" edges, the process had to have begun, for all of them to be generated.*

Definition: Process Triggered by Process *The connection of a process 2 to a process 1 by a "was triggered by" edge indicates that the start of process 1 was required for process 2 to be able to complete.*

Definition: Artifact Derived from Artifact: *An edge "was derived from" between two artifacts indicates that artifact 1 needs to have been generated for artifact 2 to be generated.*

Definition: Process Controlled by Agent: *The assertion of an edge "was controlled by" between a process and an agent indicates that the start and end of a process was controlled by an agent.*

Roles are used to distinguish the nature of the dependency in the case of multiple edges connected to the same process according to this definition.

Definition: Role: *A role designates an artifact's or agent's function in a process.*

When a process uses more than one artifact, the artifact is used by more than one process, an agent controls more than one process, or a process is controlled by more than one agent, roles are used to differentiate these several relations. The meaning of roles is defined within the context where they are defined and not by OPM, therefore it is defined by the application domains.

The graphical representation of the OPM graph also describes the causal dependencies between these entities, and is represented by an edge. A variety of reasoning algorithms can exploit this data model, which explicitly represents all the dependency relationships between entities on the OPM graph. More detaile causal relationships are defined as follows (Moreau et al. 2010):

Overlapping and hierarchical description

The need to provide description at multiple levels of detail or from different viewpoints is common for provenance systems. In order to support these, OPM allows for overlapping accounts of the same execution, which offers an explanation at different levels of detail about the same derivation. However, these accounts may differ in their description semantics. For example, if two processes are executed to create an artefact, overlapping indicates that an alternative explanation exists for the process. The first account shows a description of the two process and their dependencies on the artifact. The second account indicates that a single process operates on input artifacts and produces output artifacts.

A hierarchy of accounts is created when the refinement explanation is repeatedly used. An example is when another account is created to explain how one of the processes in the last example was performed in more detail.

Completion and inferences

As mentioned above, causal dependencies are captured and represented by mean of edges in the provenance graph. Moreover, the OPM graph explains how processes and artifacts came to be. Edges can be a summarisation of a transitive relationship, which can define completion rules and multi-step inferences.

A completion rule explains how a sub-graph can be converted into another sub-graph. For example, a “WasTriggeredBy” edge that describes a relationship between two processes can be obtained from the existence of “Used” and “WasGeneratedBy” edges. That refers to a hiding artifact used by process 2 and generated by process 1. Figure 3.6 shows that a “WasGeneratedBy” edge is a

summary of the composition of "Used" and "WasGeneratedBy", which the completion rules allow to establish the existence of some artifact.

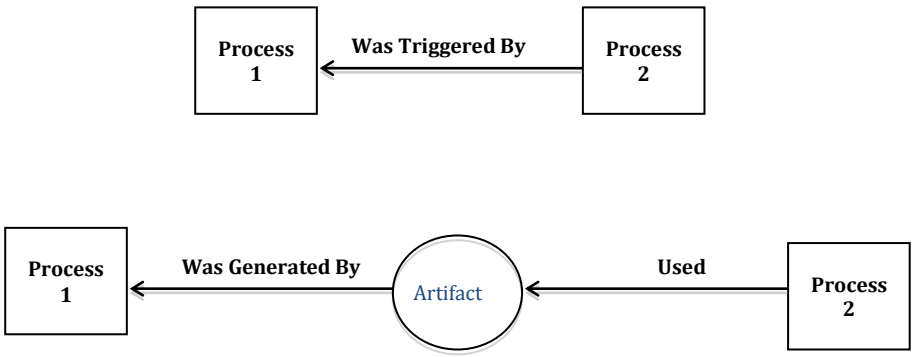


Figure 3.6: Completion of "WasTriggredBy" edge

Another relationship, which is "was derived from", refers to an intermediate process that is hidden. The intermediate process uses an artifact in order to generate another artifact as shown in figure 3.7.

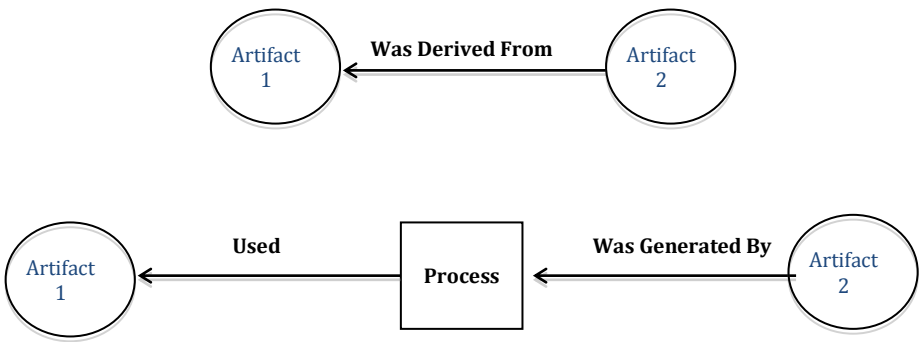


Figure3.7 : Completion of "WasDerivedFrom" edge

Multi-step inferences are when the indirect causes involved in multiple transitions are important as a direct relationship. Four new relationships express inferences in the provenance graph and represent a multi-step version of an existing dependency relationship.

Definition: multi-step WasDerivedFrom: *An artifact a1 that was derived from a2 in multiple steps is written as a1 →∗ a2, which expresses that a2 had an influence on a1.*

From this definition the three multi-step relations are as follows:

Definition: secondary multi-step edges:

- *A process p used artifact a using multiple steps is written as p →∗ a, if process p used an artifact a or an artifact that is derived from a.*
- *Artifact a was generated by process p in multiple steps is written as a →∗ p, if a was an artifact or was derived from an artifact that was generated by p.*
- *Process p1 was triggered by process p2 in multiple steps is written as p1→∗p2, if p1 used an artifact that was generated or derived from an artifact that was it self generated by p2.*

Artifacts that occur in the chain dependencies can be eliminated in order to represent the multi-steps edges by a single edge. However inferences do not allow for process elimination. The multiple steps dependencies are represented in the graph as a dashed edge, while a single step is represented by a plain edge.

3.2.3 OPM nodes primitive

As OPM represents the provenance information as a graph structure, it also assumes a few primitive sets of identifiers for the process, artifact and agent to define the structure of graphs. The five causality edges are specified by: Used, WasGeneratedBy, WasTriggeredBy, WasDerivedFrom and WasControlledBy. The following are the building blocks for creating the OPM graph (Moreau et al. 2010). The overlapping and hierarchical descriptions and inferences have been avoided, since they did not relate to applications such as this model.

ProcessId: primitive set (Process Identifiers)
ArtifactId: primitive set (Artifact Identifiers)
AgentId: primitive set (Agent Identifiers)
Role: primitive set (Roles)
Value: application specific set (Values)
Time: primitive set (Time)

Process = ProcessId →Value
Artifact = ArtifactId →Value
Agent = AgentId →Value
OTime = Time ×Time
Used = ProcessId × Role × ArtifactId ×OTime
WasGeneratedBy = ArtifactId × Role × ProcessId ×OTime
WasTriggeredBy = ProcessId × ProcessId ×OTime
WasDerivedFrom = ArtifactId × ArtifactId ×OTime
WasControlledBy = ProcessId × Role × AgentId ×OTime
OPMGraph = { (A, P, AG, US, GN,T, D, C) | A ⊂ Artifact, P⊂ Process, AG ⊂ Agent, US ⊂ (Used) GN ⊂ (WasGeneratedBy, T⊂ (WasTriggeredBy), D⊂ (WasDerivedFrom), C⊂ (WasControlledBy)}

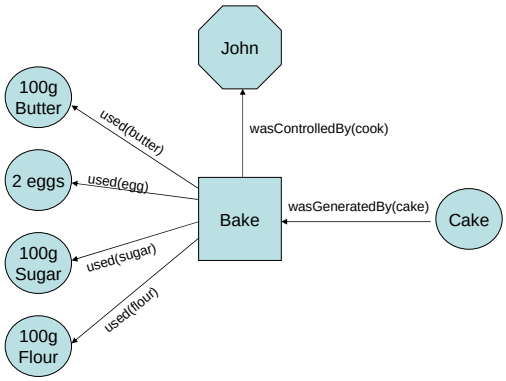


Figure 3.8: Victoria Sponge Cake Provenance

The graph in figure 3.8 illustrates an example of all the concepts and some of the causal dependencies, which express that John (Agent) baked (Process) a cake (artifact) with ingredients (artifacts) butter, eggs, sugar and flour (Moreau et al. 2010).

3.2.4 Time observation

In OPM, time information is optional and is not intended to be used for deriving causality. However, causal dependencies can be made more explicit with time. For example, if the same clock is used to measure time for both effect and cause, then the time of the cause should be before the time of its effect. Time may

be associated to instantaneous occurrences, which are related to the occurrence of creation and the use of an artefact or the starting and ending of processes.

Time information is expected to be acquired by an observer when an occurrence occurs, assuming time is measured according to a single clock or synchronised clocks in order to be compared. Time accuracy is limited to the granularity of the clock and the observer.

3.2.5 The Mapping

The main reason in mapping our model to the OPM is to facilitate the exchanging of provenance data and the interoperability with other systems, as OPM is a standardized model for representing provenance. Therefore, these definitions can be used to specify how our model can be mapped to OPM. According to these definitions, when recording the provenance information of a WSN query, a query is an action performed on an artifact (sensors) where the catalyst is a user, which means a WSN query is a kind of process. When recording the process for aggregation or the average, the process is an action on a sensor's reading, which is an artifact, enabled by a WSN query.

The OPM's three entities are represented in this model as follows:

- Artifact: sensor, data stream from sensor or derived data stream (data stream result from another process), and any event
- Process: WSN Query or any process operating on the data stream such as: aggregation, summation and division
- Agent: Users, WSN Query or other process

OPM defines five types of causal dependencies:

- Type 1: process used artifact
- Type 2: artifact was generated by a process
- Type 3: process was triggered by another process
- Type 4: artifact was generated by another artifact
- Type 5: process was controlled by an agent

This model has the following causal dependencies that map to OPM dependencies:

- Type 1: a query requesting a reading from sensors, or a process has input data such as summation and aggregation
- Type 2: a derived data stream is generated by a process
- Type 3: a process was triggered by a query
- Type 4: an event was generated in streams or from sensors
- Type 5: a query was controlled by a user, or any process was controlled by a query or another process

In order to support sensor network circumstances and preferences, a transformation was performed on the primitive sets that represent the graph structure of OPM, as in Ringelstein & Staab (2010a,b). The model has four types of primitive that represent the provenance graph structure: a process primitive, a data primitive, a dependency primitive and an event primitive.

Definition: process primitive: *a process primitive represents the provenance information of a single process.*

The process primitive represents the provenance information of each WSN query or any operation performed on the data, which is defined with the syntax:

`Process (ID, owner, category, Purpose, start time, end time)`

- **`ID`** *is the unique identifier of the process*
- **`Owner`** *is the agent controlling or enabling the process*
- **`Category`** *is the category of the process. The possible categories are defined in domain-specific ontology*.
- **`Purpose`** *is the purpose of the process. The possible purposes are defined in domain-specific ontology*.
- **`Start time`** *is the time when the process starts*

- **End time** *is the time when the process ends*

The ID together with the Owner creates a partial order of the process and specifies the ControlledBy relationship between the agent and the process. Category, purpose, start time and end time are properties of the process specified by the process primitive. The information provided by the process primitive, additional to the data and the dependency primitive, can express the provenance information and explain the dependency relationship between the input, the output and the process. The input and the output could be one element or a stream of elements, and their provenance information is represented by a data primitive.

Definition: Data primitive: *a data primitive represents the provenance information of the data, which contains all the properties that describe the data.*

Data (ID, category, source, start time, end time), which express the following:

- **ID** *is the unique identifier of the data*
- **Category** *is the category of the data (source – derived)*
- **Source** *is the source of the data*
- **Start time** *is the starting time for the data stream*
- **End time** *is the time when the stream stops*

Category specifies the type of the data stream, since it can be a source from the sensor or derived from another stream. **Source** specifies which sensor generates the data stream in the case of the source stream, or the process ID that the derived stream is an output of.

The dependencies relationship specifies the causal relationship between the processes or between the process and the data and is represented as an edge in the graph structure.

Definition: Dependency primitive: *a dependency primitive specifies the nature of the causal dependencies between the data and the process and between processes.*

`Dependency (ID, Object, Owner, Type)`

- **`ID`** *is the unique identifier of the relation*
- **`Object`** *is the related node*
- **`Owner`** *is the other related node*
- **`Type`** *is the type of dependencies relationship*

Since the data primitive represents the provenance information of the data stream, which has different characteristics as mentioned before, a different primitive is required to represent the provenance information that occurs during the data stream interval or during the activationof the process.

Definition: Event primitive: *an event primitive represents the provenance information of any change in the stream or the process by specifying all the properties that describe the event.*

Event `(ID, Owner, category, time)` where:

- **`ID`** *is the unique identifier of the event*
- **`Owner`** *is the provenance unique identifier of the artifact or the process where the event accrues*
- **`Category`** *is the category of the event*
- **`Time`** *is the time stamp of the event*

3.3 Connecting provenance policy language to the structured model

Based on the graph structure model of provenance, which is mapped to the OPM structure, and the provenance collecting policy, this section may join the two and specify the policies with regard to collecting the partial order, which constitutes a provenance graph structure model. Two case scenarios will be

presented, with an explanation of how the collection language definition built the provenance structure in a pervasive computing application. A graphical representation will be provided in order to facilitate an understanding of the process. The two scenarios focus on:

1- Debugging on deployment, when detailed provenance information may be required.

2. Auditing for the correct operation, when the administrator may wish to set up random probes that are checked against other measurements.

3.3.1 Scenario 1

Recording provenance can be used for debugging on initial deployment of a pervasive computing system: users may want to detect failure symptoms in the provenance records and diagnose problems in the process that generated an artifact. Detailed information about the data and the process have to be collected in order to allow for detecting any error in the data and tracing the process that the data go through. When an administrator tries to determine whether any error has been made due to a faulty process or faulty source information, without a record of the provenance of those data and the process it would be impossible to determine whether such a 'bug' in the result had indeed occurred.

In the scenario of this example, an administrator creates a policy to record all the events associated with queries submitted by user1 for 1 hour.

```
Policy  user1Policy {
When query.owner==user1
Capture query start event, query finish event
Duration 1 hour
}
```

The policy attribute is true. According to the extended policy automaton shown in figure 3.4, the machine transition is to check the policy condition state.

IF (query .owner == user1 && time < 1 hour) **THEN** `Capture query start event, query finish event`

When user1 submits a WSN query to calculate the average usage of his/her room measured by four sensors (1, 2, 3, 4) for 5 minutes, the condition for the query start event state is satisfied. Then according to the collection policy in expression 3.2, which is for the coarse grained approach:

$$\forall \textbf{(query, event): event = start} \wedge \textbf{start.condition = true} \Rightarrow \textbf{[record (q)} \wedge \textbf{\textit{record}} \textbf{ (data}_1\textbf{, …, data}_n\textbf{)]}$$

The information about the WSN query and the data log responding to the WSN query is the provenance information that needs to be recorded. As the process primitive represents the provenance information of a single process, the process primitive will represent the provenance information of this WSN query:

Query (ID, owner, category, purpose, start time, end time), which expresses the following constituents of the model:

- **P1** is the unique identifier of the process query
- **Owner** is the ID of user1, who sends the query
- **Category** is the category of the process, which is the query.
- **Purpose** is the purpose of the query which is to calculate the average
- **Annotation** for the query is the time period (5min)
- **Start time** is the time stamp when the query is sent.
- **End time** is the time stamp when the execution of the query has finished.

Provenance information of the input data has to contain all the properties that describe the data. A data primitive represents the properties of the data, which will be recorded for each data input log. In this case scenario, there will be four input streams collected from four sensors in the room:

Input data stream (ID, owner, category, source, start time, end time) which express the following:

- **ID** is the unique identifier for data stream (stream1, stream2, stream3, stream4)
- **Owner** is the ID of the WSN query (p1)
- **Category** is the category of the data input, which is a source stream from the sensor
- **Source** is the sensor ID that the stream is collected from (sensor1, sensor2, sensor3, sensor4)
- **Start time** is the timestamp when the source data stream starts
- **End time** is the timestamp when the source data stream stops

Additional documentation of the process, which explains the dependency relationship between the input, the output and the process, has to be recorded. A UsedBy edge connects the WSN query with the input data in the graph and is stored as a dependency relationship in the database. A dependency primitive represents the properties of the dependency relationship for each input stream:

Dependency (ID, Object, Owner, Type)

- **ID** is the unique identifier of the relation (D1, D2, D3, D4)
- **Object** is the input stream ID (stream1, stream2, stream3, stream4)
- **Owner** is the WSN query ID p1
- **Type** is the type of dependencies relationship, which is the input stream of the WSN query

A WasGeneratedBy edge connects the output data with the WSN query in the graph and is also stored as a dependency relationship in the database. A data primitive represents the properties of the output data:

Output data stream (ID, owner, category, source, start time, end time), which express the following:

- **S5** is the unique identifier for the data stream
- **Owner** is the ID of the WSN query (p1)

- **Category** is the category of the data output, which is a derived stream
- **Source** is the process number (WSN query ID p1) that generates the derived stream
- **Start time** is the timestamp when the derived data stream starts
- **End time** is the timestamp when the derived data stream stops

The dependency relationship for the result stream:

Dependency (ID, Object, Owner, Type)

- **ID** is the unique identifier of the relation (D5)
- **Object** is the output stream ID (S5)
- **Owner** is the WSN query ID p1
- **Type** is the type of dependencies relationship, which is the output stream of the WSN query output

When the five minutes has expired, the finished event is detected. The WSN query-finished event specifies that the end time of the WSN query and all the input and the output data stream have to be recorded as shown in the following expression:

$$\forall(process, event): event = finish \wedge finish.condition = true \Rightarrow [record(query.endtime) \wedge record(data_1.endtime, .., data_n.endtime)] \quad (3.5)$$

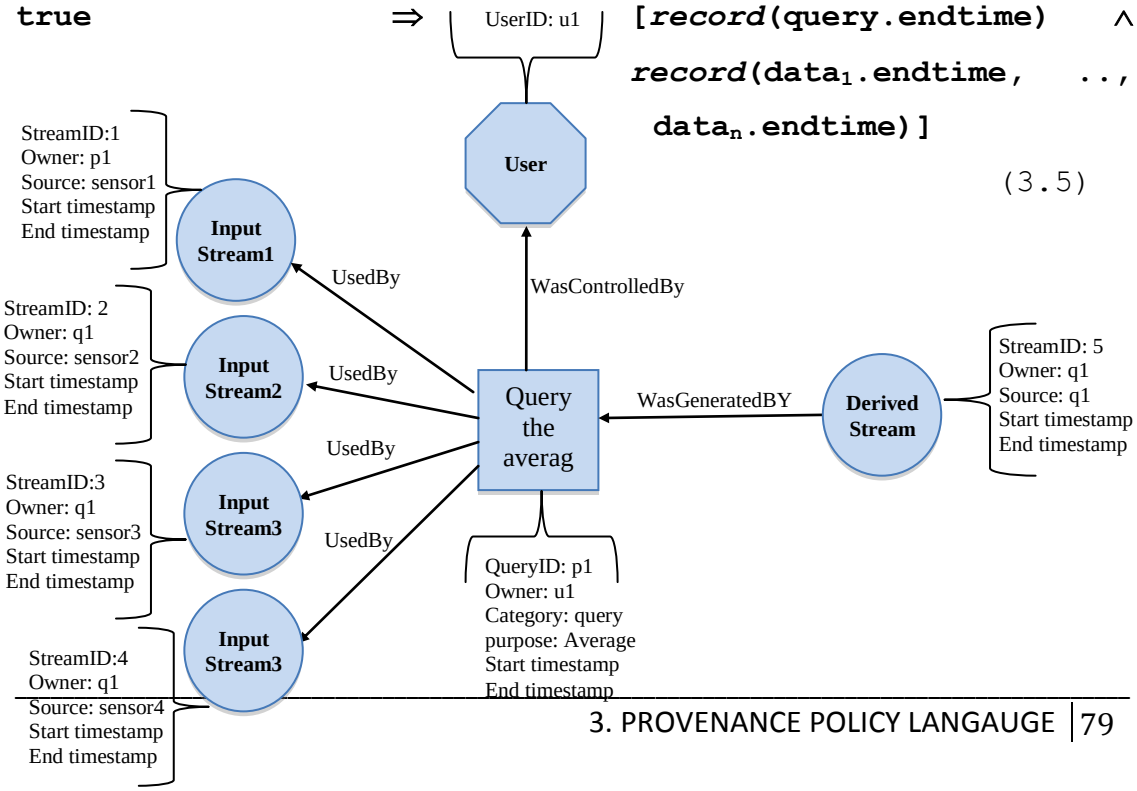


Figure 3.9: Provenance graph of coarse-grained form

Figure 3.9 shows the graphical representation of the WSN query provenance information structure, which expresses that user1 (Agent) queries the average (Process), and generates the derived stream result (output artifact) from stream1 of sensor1, stream2 of sensor2, stream3 of sensor3 and stream4 of sensor4 (input artifact).

As mentioned above, the fine granularity of recording provenance information could express the detailed information that describes the process. If detailed information is needed when recording the process of calculating the average, the WSN query execution includes sequences of the process to obtain data results as follows: the summation process will obtain the sum of the readings from the four sensors, then the results will be divided by 4, which is the number of sensors in the room, in order to obtain the result. If all these items of information have been recorded, the debugging process can accordingly check if the summation process has included all the input data or if the summation result has been divided by the correct number of sensors. Therefore, two process records and one data record will be added to the provenance structure and the collection policy of these processes is shown in expression 3.4:

$$\forall \textbf{(process, event): event = start} \wedge \textbf{start.condition - true} \Rightarrow [\textit{record}\textbf{(process}_1\textbf{, …, process}_n\textbf{)} \wedge \textit{record}\ \textbf{(data}_1\textbf{, …, data}_n\textbf{)}]$$

The WSN query information and the input data stream information will be the same as mentioned before in the coarse grained form. However, the query process triggers the summation process and creates a TriggeredBy edge. The process primitive is used for representing the summation process as follows:

Summation process (ID, input, output, owner, category, purpose, start time, end time), which expresses the following constituents of the model:

- **P2** is the unique identifier of the summation process
- **Owner** is the WSN query ID p1
- **Category** is the category of the process, which is the summation
- **Purpose** is the purpose of the summation process which is calculated as the sum of the four readings
- **Annotation** is the number of input streams
- **Start time** is the time stamp when the process starts
- **End time** is the time stamp when the process finished

The derived stream of the summation process is expressed by:
Output data stream (ID, owner, category, source, start time, end time) which express the following:

- **S5** is the unique identifier for the data stream
- **Owner** is the ID of the WSN query (p2)
- **Category** is the category of the data output of the summation process, which is the derived stream
- **Source** is the process number (WSN query ID p1) that generates the derived stream
- **Start time** is the timestamp when the derived data stream starts
- **End time** is the timestamp when the derived data stream stops

This output stream has two roles, since it has a dependency relationship with two processes: it is an output of the summation process and an input for the division process. A WasGeneratedBy edge connects the output stream with the summation process, a UsedBy edge connects it with the division process in the graph, and it is stored as a dependencies relationship in the database. The process primitive represents the division process:

Division process (ID, input, output, owner, category, purpose, start time, end time), which expresses the following constituents of the model:

- **P3** is the unique identifier of the process

- **Owner** is the WSN query ID p1
- **Category** is the category of the process, which is division
- **Purpose** is the purpose of the division process which is to calculate the average
- **Annotation** is the number of sensors for division
- **Start time** is the time stamp when the process starts
- **End time** is the time stamp when the process finishes

The derived stream of the division process is expressed by:
Output data stream (ID, owner, category, source, start time, end time), which expresses the following:

- **S6** is the unique identifier for the data stream
- **Owner** is the ID of the WSN query (p3)
- **Category** is the category of the data output of the division process, which is the derived stream
- **Source** is the process number (summationID p2) that generates the derived stream
- **Start time** is the timestamp when the derived data stream starts
- **End time** is the timestamp when the derived data stream stops

The following provenance graph (figure 3.10) is the graphical representation of the provenance structure for the fine-grained form, which expresses that user1 (Agent) queries the average usage (Process), which triggers the summation process (process). The summation process generates the sum (stream5 derived output artifact) from the four input streams (input artifact) stream1 of sensor1, stream2 of sensor2, stream3 of sensor3 and stream4 of sensor4. The output stream of the summation process (stream5) has two causal dependencies: one is the WasGeneratedBy relationship (output artifact) to the summation process and the other is the UsedBy relationship (input artifact) to the division process, which is used to generate the average of the room usage (stream6 output artifact).

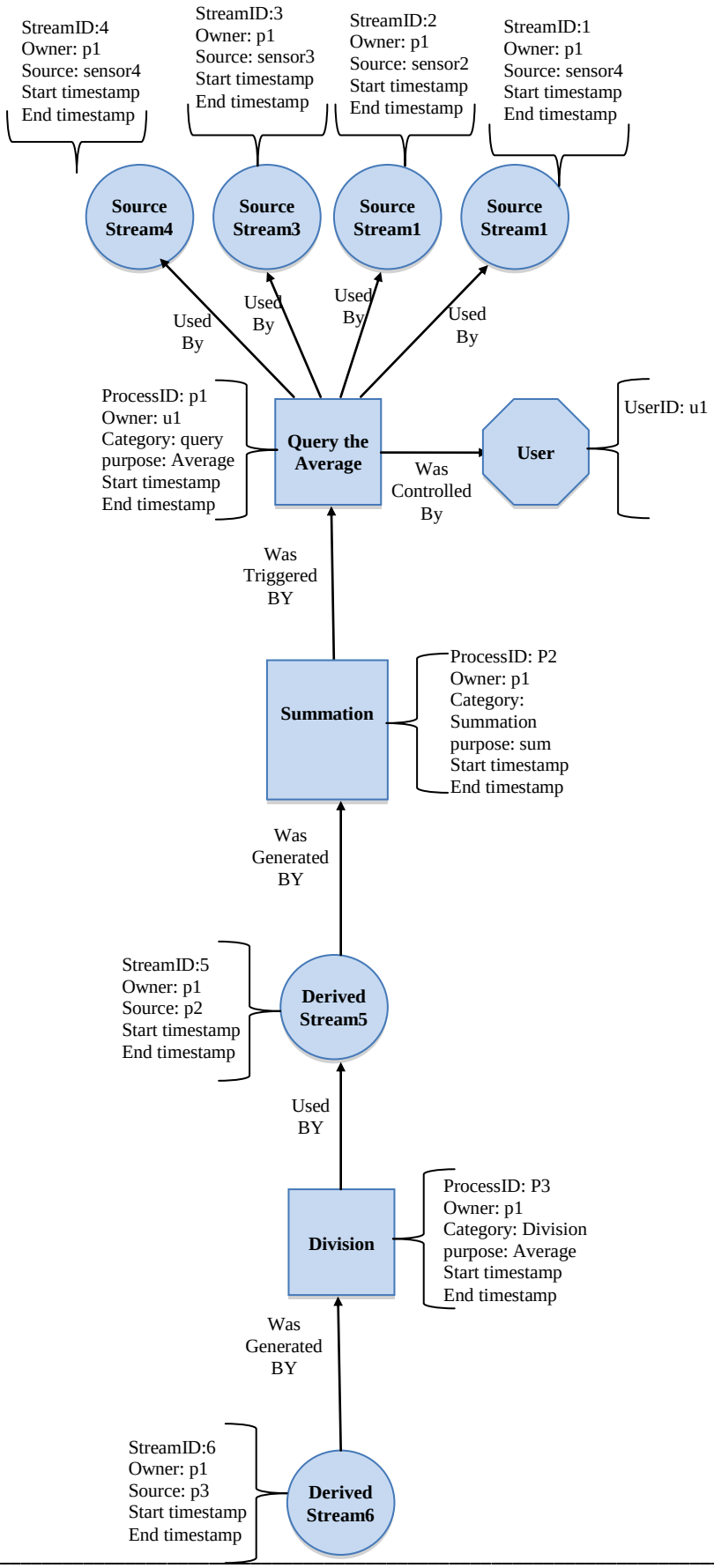


Figure 3.10: Provenance graph of fine-grained form

3.3.2 Scenario 2

The second scenario is for auditing, when the provenance information has to be collected for checking random probes, which means the provenance information of any interesting event that occurs needs to be recorded. The scenario for this example is monitoring the usage of a user, by checking the reading of the sensor that measures the electricity usage of that user, and recording the provenance information of the major events that occur in a five minute period. The scenario is that the sensor is measuring a plug for the user: after one minute the user connects a device to the electrical plug and starts using the electricity, a sensor fails after 30 seconds and is fixed one minute later, then the device stops using electricity in the fourth minute.

Therefore, the major events are: when the user starts to use the electricity (the sensor reading greater than 0); when the sensor fails (the reading is null); when the sensor starts work normally again (the reading become normal); and when the user stops using the electricity (the reading is equal to 0). The collecting policy states that:

∀ (streamID, event): event.condition = true
⇒[record event]

The provenance graph consists of a data stream and the events that occur in the data stream. The data stream log properties are expressed in:
Data stream (ID, owner, category, source, start time, end time)

- **S1** is the unique identifier for the data stream
- **Owner** is the ID of the WSN query for monitoring the sensor reading
- **Category** is the category of the data, which is the source stream
- **Source** is the sensor number (sensor1) that generates the stream
- **Start time** is the timestamp when the stream starts

- **End time** is the timestamp when the stream stops

When the user starts to use the electricity, the sensor reading is changed from 0 to 4, which is the condition of increase event:

IF r_n.value > r_{n-1}.value

THEN event = Increase

At this acceptance state, according to figure 3.2, the information about the event needs to be recorded. This event information is expressed by the Event primitive:

Event1 (ID, Owner, category, time) where:

- **ID** is the unique identifier of the event (1)
- **Owner** is the provenance unique identifier of the stream where the event accrues (s1)
- **Category** is the category of the event, which increases in the sensor reading
- **Time** is the time stamp of the event (timestamp1)

According to figure 3.3, when the sensor gives a null reading, the condition of Failed state is satisfied, and at this state the event is expressed by:

Event2 (ID, Owner, category, time) where:

- **ID** is the unique identifier of the event (2)
- **Owner** is the provenance unique identifier of the stream where the event accrues (s1)
- **Category** is the category of the event, which is Sensor failed
- **Time** is the time stamp of the event (timestamp2)

The machine transition is in state Stationary until the reading becomes a normal reading. Then it moves to Back state, which is the third event and is expressed by:

Event3 (ID, Owner, category, time) where:

- **ID** is the unique identifier of the event (3)
- **Owner** is the provenance unique identifier of the stream where the event accrues (s1)

- **Category** is the category of the event, where the reading is back to normal
- **Time** is the timestamp of the event (timestamp3)

The last event, when the usage stopped, means the reading = 0 and that implies:

$r_n.value < r_{n-1}.value$

With this condition satisfied, then the event is decrease. The event that has to be recorded is expressed by:

Event4 (ID, Owner, category, time) where:

- **ID** is the unique identifier of the event (4)
- **Owner** is the provenance unique identifier of the stream where the event accrues (s1)
- **Category** is the category of the event, which is decrease
- **Time** is the timestamp of the event (timestamp4)

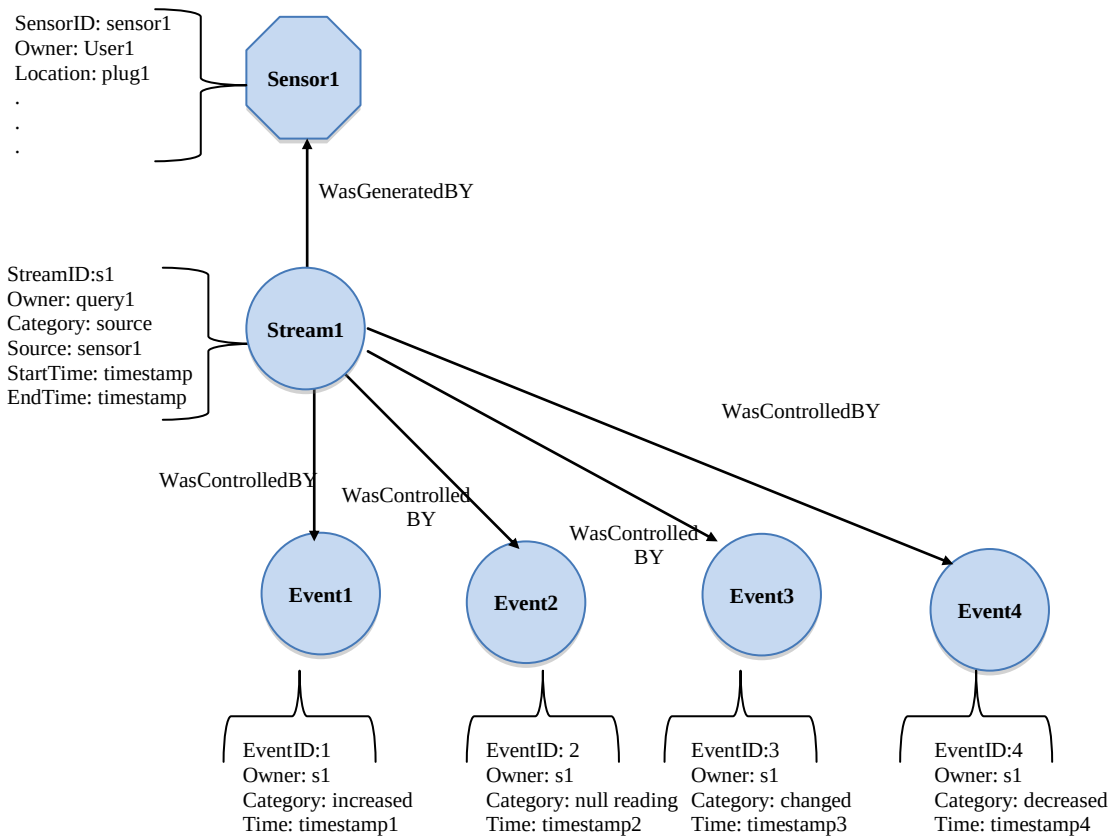


Figure 3.9: A provenance graph of events recording structure

The provenance graph in Figure 3.9 illustrates that stream1, which is generated by sensor1, shows the first event that occurs (usage starts when the reading increases) at timestmp1. In the mentioned period, at timestamp2 a sensor fails, and at timestamp3 the sensor goes back to giving a normal reading. At timestamp4, the electricity usage stops (reading =0).

3.4 The policy language description

This proposed policy language specifies the various kinds of information that need to be recorded as provenance information in such pervasive systems, when the filter predicate has occurred and conditions of the event of interest are fulfilled. It is similar to a policy-based framework such as Ponder2 (Twidle et al. 2009) or PMAC (Agrawal et al. 2005), which focus on a condition-action or an event-condition-action paradigm. There are two basic policy types: obligation policies, which specify the action that must be performed by the system when certain events occur; and authorization policies, which define the activities that can be performed on the target by the subject – or in other words it is an access control policy. However, this specific language defines composite events and their attributes, which are used in the provenance filter for filtering events and recording the information of certain events as provenance information. Therefore, it is a combination of an event monitoring that needs to process massive streams of events in real-time and obligate policies that specify the action that must be performed by the system.

For example, when a user is interested in finding matches to an event pattern such as finding upward raise events, which define when there are three consecutive correlated readings and reading n > reading n-1> reading n-2. The language has to define the event patterns that need to be detected and the event trigger reaction, which specify the obligated action and what information needs to be recorded as provenance information of the event.

In this proposed language, the causal relationship between the model entities (e.g. process and artifact or process and agent) are implicitly recorded by

defining new meta events, except some relationships such as the dependency relationship between the input and output stream with their related process, which are represented by the UsedBy and WasGeneratedBy relationship in the OPM causal graph.

The causality relationship can be decorated with time information. Instantiation time for occurrence of events and occurrence of starting or ending of process is observed and added to the metadata. Time can be useful in validating causality claims and can be comparable if it is measured according to a single clock or synchronised clock. Many wireless sensor network applications need the local clocks of sensor nodes to be synchronised. However these clocks are subject to clock drift. Therefore, many networking protocols require a common view of time to exist and be available to all nodes in the network at any particular time (Sivrikaya et al. 2004). Moreover, increasing research proposes synchronisation methods explicitly designed for sensor networks. In a wired sensor application, such as in our use case, a computer clock is used for time observation when any of the intended occurrences occur.

In some cases, when a bug is discovered in the provenance system, which needs changes in the data in order to correct it, fixing the problem and recurring the path of the metadata requires an ad-hoc human recording systems involvement.

The necessary policies to collect the metadata are an application-specific scenario. The domain requirements can call for a different kind of metadata and a different level of detail. For example, when debugging on deployment, the transformation process and sensors must be named, while for auditing, more detailed information is needed.

The design and implementation of the proposed policy language puts an emphasis on flexibility and interactivity, where the administrator interacts with the system by defining new events and creating new policies, or cancels an existing one. Another consideration was in addressing the challenges that arise

with stream data, and ensuring low overheads in tracking and collecting provenance. However, it has the limitation of not recording the provenance of policy creation or deletion, which could form valuable information when reviewing the provenance information recorded. Another concern is whether the causal relation needs to be explicitly noted. In our policy grammar, as mentioned before, most causal relationships are implicitly recorded and this is needed for a recursion query in order to retrieve the complete components of the provenance information. The reason behind this is to save storage space, but a recursion query is expensive. Therefore, a greater analysis is needed in comparing which cost is more acceptable. In the policy grammar set up at the moment, this doesn't quite happen except implicitly by defining new meta events.

3.5 Summary

This chapter defines the requirements on the policy language of the provenance collection in a pervasive computing application, which is based on an event alert and represented by it. The policies specify the different kinds of information that need to be recorded as provenance data and they describe how events are filtered. Then the chapter provides an overview of the open provenance model and presents the three basic entities of the provenance information, which are process, entities and agent,. It also introduces the graphical notion for a provenance graph, which describes the causal dependencies between these entities, represented by an edge. It then explains how the proposed model is mapped to the OPM's entities and its causal dependencies. Two case studies are demonstrated in order to explain how the proposed provenance collection policies can be applied and connected to the structured graph model. Finally is a description of the proposed policy and event collection language.

CHAPTER 4 ACCESS CONTROL POLICY

This chapter starts by analyzing the requirements of provenance access control and introduces the Role Based Access Control model. The next section develops the access control language model based on the provenance model that supports fine-grained access and privacy policies. At the end a use case example is presented to explain how the proposed access control language is applied.

4.1 Desiderata for Fine-grained access control

Pervasive computing applications may form part of a critical information infrastructure, and it is expected to be confidential and trustworthy. Provenance information of such applications may reveal critical information about the owner or the process and actions performed on the data. In some applications, the data is more important and needs to be protected more than the provenance; in other cases the provenance is more critical than the data. In the use case scenario, the question "*What amount of electricity has been used by a user?*" asks about the data. The questions "*How has this amount been used?*", "*When was this amount used?*", or "*What machines have been used?*" relate to the provenance information. The electricity usage by a user is abstracted as data items for which provenance information is collected. The provenance of the usage data is the detailed

information that has been collected about the electricity usage such as the plug or switch that has been used, the time and the duration. Each user's usage or real time WSN query may generate a provenance record. Provenance records may contain private information about users such as their electricity usage or personal habits. When gaining access to electricity usage information, the details about the user's private activities can be inferred. Therefore, both data and their provenance are sensitive and have different concerns as to why they need a different granularity level of access control. Given that the reasons for accessing the data and its provenance differ, we have chosen to separate the mechanisms for protecting the provenance from data.

Provenance records will be available for access and query by different participants. However, some records cannot be revealed to everyone. In our use case application, users will have access to detailed data of their usage only and can decide who other than their supervisor and manager can access this provenance information. In other words, users can see the usage information of other users only if they have received permission from the owner. Supervisors have more detailed information about the data of users under their purview; on the other hand, they can access only the data of users from other purviews. The manager is made aware of the big picture and has full access to the data and the provenance information of the subordinates that he/she manages. Therefore, the individual user has associated access depending on the level this user is granted.

Provenance access control has been considered as one of the primary components in provenance systems. One of the challenges in provenance access control is the need for an access control language that supports fine-grained policies, privacy and preferences. It is not sufficient for the storage facility to provide multiple copies of provenance records depending on the principle of authority, due to the size of provenance information and the potential large number of participants. For example, one copy contains one field and another copy contains various fields. In order to provide finer grained control over exactly which participants can access which details of the provenance information, and to

overcome the problem of storing multiple copies of provenance records, one of the solutions that can be used is to assign each participant (user) to an appropriate role based on their particular responsibilities and qualifications (Ferraiolo, Barkley et al. 1999). Roles can be created for a job function or job title in relation to the authority required for meeting the goals of the organization, and these roles can be associated with permission or access rights. The role is an intermediary that brings a collection of users and a collection of permissions together (Sandhu 1996). A permission or privilege is an approval of access to data resources, or approval of a particular executable part of a program to be performed on data objects.

4.2 Mapping to Role Based Access Control

In the use case application, by applying a Role Based Access Control (RBAC) model, the user can grant access only to resources that he/she has permission for. The model has four components, as shown in figure 4.1:

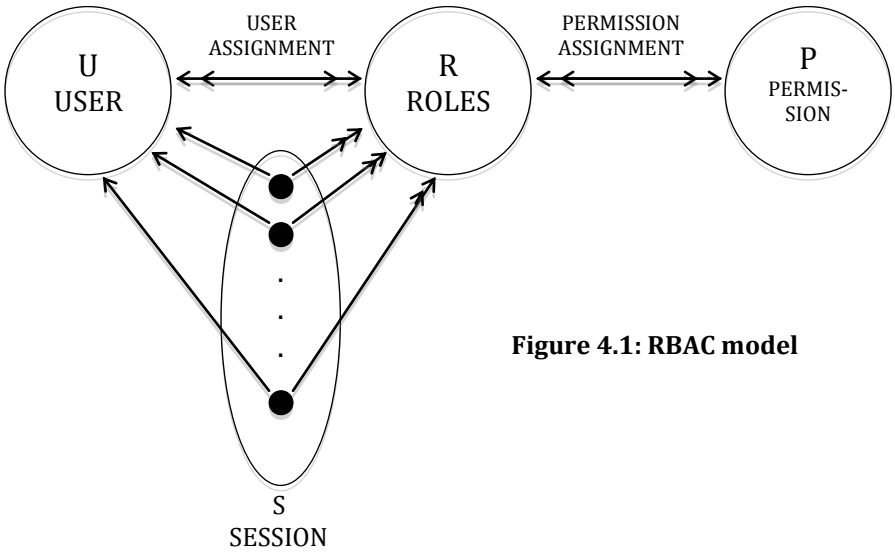


Figure 4.1: RBAC model

- User (U) is a human being or it could be generalized to include intelligent autonomous agents such as computers or a network of computers; or a human being who needs to obtain access to the resources and perform actions on data. Each user is assigned to one

or more roles depending on the user's authority and responsibility. A relationship between users and roles is many-to-many, which means a role can be associated with one or more users, and a user can be associated with one or more roles depending on his/her authority

- Roles (R) can be defined as a combination (Schaad, Moffett et al. 2001) of official position and an attribute as shown in table 4.1. An official position could be that of a supervisor, manager, electricity user or an outside party such as an Electricity Company or Analysis Company. Attribute represents an additional description of a user such being a supervisor of the area or a friend of a user.

Role	Official Position	Attribute
A	Electricity User	Normal
B	Electricity User	Owner
C	Electricity User	Friend
D	Supervisor	Normal
E	Supervisor	Section
…	…	….
…	…	….
K	Manager	System manager
…	…	…
X	Third Party	Electricity Company
Y	Third Party	Analysis Company
…	…	…

Table 4.1: Roles specification

- Permission (P) is an approval of a particular mode of access to a data object. It confers the ability of the holder to perform some actions, which could be from a very coarse grain mode such as access to all records, to a very fine grain one such as access to a particular field in the records (Ferraiolo, Barkley et al. 1999). In

our application, the provenance data is stored in a relational database. Therefore the nature of permission is accessing tuples, attributes and relations or tracking the dependencies of data in order to view the provenance information. The relationship again could be many-to-many between roles and permissions. Assigning permission to roles and users to roles provides flexibility and granularity control on user action. Table 4.2 defines the semantics of permission in the use case.

Permission	Access right
1	The usage of the user
2	The detail provenance of a WSN query
3	The input and output of the WSN query
4	The usage of a sensor
5	Location provenance of a sensor
6	All provenance of a sensor
7	Event of a stream
8	Event of a process
…	…

Table 4.2: Permission specification

- Session (S) is established when the user is authenticated and it involves a set of processes which act on behalf of a user and is permitted by the roles (Barkley 1997). A user may have multiple sessions but a session in only associated with a single user. Therefore the relation between user and session is one to many.

Since the role is based on the combination of job position (J) and an attribute (A), the total number of roles would be the product of every job position and every attribute:

R = J * A

However, the actual number is a subset of the theoretical number, as the user cannot combine it with supervising the same area or a different area to define a role.

$R \subset J * A$

It was mentioned in Schaad, Moffett et al. (2001) that the oral estimation of the number of roles, discussed at the RBAC2000 workshop, is approximately 3-4% of the user population.

All components of RBAC have to be under the control of the system administration. In order to reduce errors in administration, the assignment of access rights to a role and mapping users to a set of roles has to be at a different level, not under a single security officer. For example, a permission administrator assigns all permission numbers that are allowed to each role; the supervisor of the section has access control to all the provenance information of users of that section and some general information on users from other sections. A role administrator maps each user to a role or sub-set of roles that he/she is a member of.

The model $RBAC_1$ (Sandhu 1996) introduces role hierarchies in which one role is superior to another. In this application a role hierarchy can be defined as shown in figure 4.2, when a section supervisor is one of the electricity users in that section. It is the capability of one role to inherit another role, or in other words, the parent role inherits permission from all children roles.

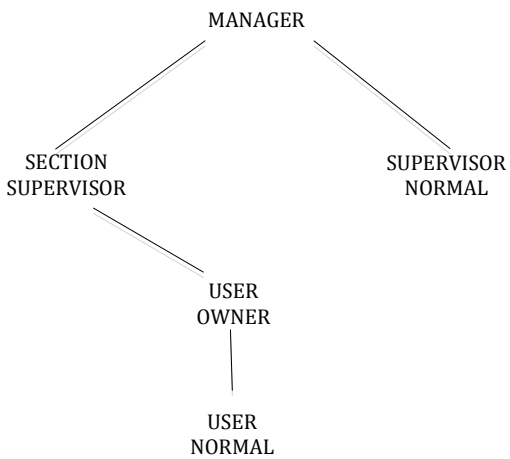


Figure 4.2: Role hierarchy

4.3 Access control policy

Definition: Access Control Policy: *a set of rules (authorization requirement) that answer the question of who is allowed to perform which actions.*

User authentication is performed externally and is outside the scope of this section. A Role Based Access Control model (RBAC) is used in order to provide finer-grained control over exactly which participants can access which details of the provenance information, and address the requirement discussed in the previous section.

The basic concept of RBAC is that users are assigned to roles and roles are assigned to permissions. *Permissions* are defined as privileges associated with roles for each data object. The relationship between users and roles is many-to-many, which means users can be assigned to multiple roles and roles can have different users, and there is the same relationship between roles and permission. Users are mapped to roles and acquire permission in a *session.*

eXtensible Access Control Markup Language (XACML) is a standardized RBAC specification language that provides the core RBAC specification and defines XML based policy framework. XACML defines three policy elements: permission, restriction and obligation. These policies define rules by connecting a set of subjects (actors) with a set of targets (data) and specifying the conditions of the rule. When the conditions are met, the rule results in a given effect: 'Allow' or otherwise 'Deny'. In some rules, an obligation needs to be met before permission is granted.

XACML is used as a starting point for this model. In the application, roles are created for a job function or job title regarding their authority.

```
ROLES ←
{users, supervisor, manager, administrator, third party}
```

User: electricity users

Supervisors have users under their purview

Manager is made aware of the subordinates that he/she manages.

Administrator assigns users to a set of roles and permission to roles

Third Party is an outside party such as an electricity company or analysis company

The following plain-language rules are to be enforced:

Rule 1: A user, identified by his number, may read any provenance record for which he/she is the designated user.

Rule 2: A user can access some static provenance information of his sensor location.

Rule 3: A user may access a WSN query result and the corresponding input of a friend without the detailed process that the data goes through.

Rule 4: A supervisor may access any provenance record for those under his/her purview.

Rule 5: A manager has full access to the data and the provenance information.

Rule 6: An administrator shall not be permitted to read any provenance record, as he/she works only as an application controller.

Rule 7: A third party has access to specific parts of provenance information depending on their level of involvement. For example, an analysis company can access the information about the electricity usage of a group of user in a room, in daytime or night-time.

According to these rules, an additional description (attribute) is needed for roles in order to specify the fine granularity level of access that can be assigned to each role with different attributes. For instance, a *User*, which is an electricity user, can have different attributes as shown in table 4.1. A *User* with the attribute *Normal* has access to the total usage of the other users. A *User* with the attribute *Owner* has access to all the provenance information specified in rule numbers 1 and 2. A *User* with the attribute *Friend* has access to some of his/her friend provenance information specified in rule number 3.

Provenance information is recorded as tuples in a table. In our case, each primitive described in chapter 3 is stored in a table, therefore four tables are created: a table for stream provenance information; a table for process provenance information; a table for event provenance information; and a table for dependencies relationships. In order to provide a granularity level of tuple or a granularity level of table, permission can be defined as access to a field in a table tuple.

Rules 1 and 2 illustrate a simple rule with a single condition: the user is the owner of the data. Table 4.3 shows the permission access for the user with the condition required

Permission	Condition
Stream table Process table Event table Location field in sensor table	UserID = OwnerID

Table 4.3: Owner user (permission and condition)

Rule 3 shows restrictions on the process applied to the data and any changes. A user can access only some information of his/her friend WSN query input and output. However, the permission is on condition of being a friend, as shown in table 4.4.

Permission	Condition
Stream table (WSN Query input and result)	UserID has a friend attribute User friend id = Ownerid

Table 4.4: User friend (permission and condition)

Rule 4 includes all the permissions that are allocated for a user role, in addition to other permissions assigned to a supervisor role, but under a condition of being a supervisor of the area where the user is. Table 4.5 shows the permission that is allocated to a supervisor if the user is under his purview.

Permission	Condition
Stream table Process table Event table Sensor table	Owner ∈ Users in the supervisor purview

Table 4.5: Supervisor (permission and condition)

Rule 5 shows a full permission without any condition and Rule 6 shows full restriction.

Rule 7 shows part of permission allocated according to the role attribute. In this example, the condition is that the attribute is an analysis company.

The system's RBAC has two states. A persistent state must be available throughout the lifetime of the system. Its components are the policy definition, which includes user and role identifiers, role hierarchies, user assignment and permission assignment. A soft state, which is throwaway, can be reconstructed from the persistent state. This state includes the session state, which consists of active user sessions and their currently active roles. The process of a session state is shown in the following algorithm:

Algorithm 4.1: Algorithm for implementing the proposed access control model

```
1: sid  ←  SESSIONS [REQEST.UID];
2: rolessid ← ROLES [sid];
3: att ← request.att
4: r ← mini. rolessid
5: for all r ∈ rolessid  do
6:       found ← DFS(att, perms)
7:       perms ← PERMS[r]
```

```
8:        cond ← perms.cond
9:        if match = true then
10:             effect (REQUEST.query)
11:             return
12:       end if
13: end for
14: deny (MSG);
15: return
```

Algorithm 4.1 shows the algorithm used for permission evaluation. The algorithm uses the incoming request as input and uses attributes contained in it (UID) to identify the set of active roles (lines 1-2). The request attribute is for identifying and evaluating the role condition (line 3). The least privileged role is required to perform a search (line 4). Finally, a depth first search (DFS) is performed on the role hierarchy defined by the policy starting at the current role (line 5). If the target role (defined by the permission) is found during the DFS and the condition is satisfied, the message is allowed. Otherwise, if all active roles are exhausted, the request is denied (line 14).

4.4 Applying access control policy in a use case example

In our first use case scenario, where the user sends a WSN query requesting the average usage of the room including four sensors, the detailed provenance information recorded are: the reading of the four sensors (input stream), the information about the WSN query, the information about the summation process with its input and out dependencies, and the information about the division process with its input and output dependencies. The user can access the result data and all detail provenance information of his WSN query, since the condition (the user is the owner of the data) of the first rule and the second rule has been met:

Rule 1: A user, identified by his/her number, may read any provenance record for which he is the designated user.

Rule 2: A user can access some static provenance information of his/her sensor.

Therefore, the user (User1) who sends the provenance query is the owner of the four sensors and these sensors are measuring his electricity usage. The roles assigned to this user are:

Role A, which has the official position of electricity user and attribute Normal
Role B, which has the official position of electricity user and attribute Owner

In the case where User1 has a friend (User2) from another section and Role C, which has the official position of electricity user and attribute Friend, is assigned to User1 and his/her friend (User2), this means that User2 has access to the provenance information of the WSN query such as time, input and to show the result without the detailed information of the WSN query provenance information, since he/she satisfies the Rule3 condition of being a friend of User1 (the WSN query owner):

Rule 3: A user may access a WSN query result and the corresponding input of a friend without the detailed process that the data goes through.

Emp	Role	Official Position	Attribute	Description
U1	A	Electricity User	Normal	Electricity user
U1	B	Electricity User	Owner	Sensor and WSN query owner
U1	C	Electricity User	Friend	U1 is a friend of U2

Table 4.6: Assign user to role

Role	Permission	Access right for role
A	1	The usage of the user
B	2, 4, 5	The detail provenance of a WSN query, The usage of a sensor, Location provenance of a sensor
C	3	The input and output of the WSN query

Table 4.7: Assign role to permission

Table 4.6 shows the assignment of user1 to roles, specified in table role specification in table 4.1, according to the application of Rule 1, 2, and 3. Table 4.7 shows the permissions that are associated with each role. Therefore, from the two assign tables: U1 has the permission specified in table 4. 3 and the permission specified in table 4.4.

A supervisor, where user1 is under his purview, is assigned to Roles D and E. Role D has an official position of Supervisor and attribute normal. Role E has an official position of Supervisor and attribute section. He/she can have access to all provenance information of user1 WSN query and usage, because he/she met the condition of Rule 4:

Rule 4: A supervisor may access any provenance record for those under his/her purview.

However, he/she cannot access the provenance information of any WSN query and usage of User2, because User2 is not under his/her purview.

Emp	Role	Official Position	Attribute	Description
S1	D	Supervisor	Normal	Normal supervisor for U2
S1	E	Supervisor	Section	Supervise U1

Table 4.8: Assign supervisor to roles

Role	Permission	Access right for role
D	1, 3	The usage of the user, The input and output of the WSN query
E	2, 4, 6	The detail provenance of a WSN query, The usage of a sensor, All provenance of a sensor

Table 4.9: Assign role to permission

From tables 4.8 and 4.9, the normal supervisor has access to the usage and the WSN query input/output of any other user not under his/her purview, where the supervisor of the section has access specified in table 4.5 of users in his/her section.

The manager satisfies the condition in Rule 5 and has access to all the provenance information of User1 and User2 without any condition.

Rule 5: A manager has full access to the data and the provenance information.
The manager is assigned to Role K, which has the official position of manager and attribute System Manager.

In the second use case scenario, the provenance information of any event is recorded for five minutes as provenance information of that stream. The stream is generated by a sensor; therefore the owner of the sensor and his/her supervisor only has access to that provenance information according to rule 1 and rule 4 respectively. Permission numbers 7 and 8 from table 4.2 are assigned to role B for the user and to role E for the supervisor.

4.5 Summary

This chapter discussed the need for an access control model that supports the fine-grained and privacy policies. Then it introduced the role based access control model and reviewed its four main components: roles, users, permissions and sessions. RBAC was presented as a solution for providing the requirement access control to provenance information. The proposed access control language has been influenced by XACML, which is a standardized RBAC specification language that provides the core RBAC specification. The access control policy is a set of rules that answers the question of who is allowed to perform which actions. These rules are translated into permissions and conditions of the proposed access control model, which have been applied and explained in the case study.

CHAPTER 5 MAPPING TO IMPLEMENTATION

In this chapter, we describe the experimental setup required to evaluate the proposed system and briefly discuss some implementation aspects required for simulating the use case system and the sensor component. Following this, the focus is on the prototype implementation of the proposed provenance solution and access control.

5.1 Experimental setup

The experimental setup for evaluating the proposed collection model requires an application that combines the monitoring of an electrical energy usage feature with the sensor middleware functionality. Any sensor applications need middleware to query a reading from sensors or to process that data then store the result in a database. The provenance service tracks and records provenance information by interacting with the middleware. Several solutions for middleware systems have already been implemented and assessed (Horré et al. 2007). However, these middleware systems and existing monitoring applications require a real sensor network and are not efficient enough for evaluating the requirements of the proposed provenance collection and access control.

TinyDB, sensor Query Processing in TinyOS, can connect to TOSSIM: a simulator for the TinyOS network, but it does not support multiple users and, consequently, is unable to apply the access control model. Therefore, the solution lies in implementing a sensor middleware with the essential functionalities, and simulating a system for the monitoring of electrical energy usage, as mentioned in the use case section.

Furthermore, most sensor network simulators and test-beds are for examining and comparing issues in the sensor network design such as the routing, protocols and performance. They do not support any real-time contact with the sensor readings. In our case, the main concern is, therefore, to work with real-time sensor readings and the operations applied to these data, without going into the detailed design of sensor networks. This has led us to design a simple sensor component that generates events in order for these to be detected by the proposed provenance collection model.

5.1.1 Simulated application architecture

The architecture of this simulated application is, as discussed in the background chapter, based on three components: the WSN query planner service component, the WSN query execution engine component and the simulated sensor component.

5.1.1.1 WSN Query planner

A WSN query planner provides a declarative language for specifying queries, where the user describes what he/she wants by selecting the set of attributes and conditions such as the time period, selected sensors and operation. Queries can be real time queries, data obtained directly from a sensor, or data retrieved from a database. The incoming WSN query is parsed and translated into an internal representation. The WSN query planner accepts the WSN query requests and deploys the optimized queries to the query execution engine, where they are executed.

In order to illustrate the query planner, an example of a query and an optimized query are presented. Figure 5.1 shows a simple query, which requests a sensor reading. It is not complex and does not need a plan for executing.

```
SELECT Reading
FROM sensor1, sensor2 …
```

Figure 5.1: A simple query

A complex query may require aggregation and grouping that need a plan for executing, which is called query optimization. An example of such a query is shown in figure 5.2. The query requests the average usage of light in each room for five minutes.

```
SELECT Average (light)
FROM sensors
GROUP BY roomno
DURATION 5min
```

Figure 5.2: A complex query

Bemana (2012) has proposed a new method for executing optimized queries by defining three rules:

1- Do select before all operations
2- Do project after selection, before other operations
3- After following rules 1 and 2 you can join operator.

Therefore, the query planner executes this query by following these rules. The execution requires selection of the readings of the sensor that measures the electricity usage of light switches, then grouping of the readings of each room, then calculates the average.

5.1.1.2 WSN Query execution

The WSN query execution component receives the transformed queries from the WSN query planner and executes them for the duration of their lifetime.

It supports standard query features such as filtering, joins, grouping and aggregation.

The WSN query execution component is configured to collect data from a number of sensors. It obtains the sensor reading using the ID of the sensor and the analogue input number that the sensor is connected to. It then time-stamps the reading and gives it a sequence number, then forwards it to the database. The current AC is calculated from the sensor reading and the usage amount is sent to the user. The reading could also be sent as WSN queries result in a case where it was the result of a user's WSN query. In the case of sending the result to the user, a buffer interface is supported by the WSN query execution for the WSN query result. A buffer interface is where the result tuples are temporarily stored in a buffer and sent to the web page when the result set count is completed. This method of interaction is supported in order to reduce the webpage reloading process.

5.1.1.3 Simulated sensor component

In order to understand how the electricity usage is measured, a ccurrent sensor was attached to the outlet that it is sensing to measure the electrical energy consumed. The current sensor, i-Snail-VC, is a self-powered AC current transducer, and it provides a 0-5V DC analogue signal proportional to the AC current flowing through the device wire window (Phidgets 2011). The sensor is connected, using a cable included with the sensor, to an analogue input on a Phidget Interface Kit that is used to measure continuous quantities of the AC current. The formula for converting the sensor value into AC Amps(RMS) is:

AC Amps(RMS) = `sensor value` / 10.

And the formula to convert Amps to watts is:

Watts = `Amps * volt`

The experiment was performed on one sensor. However, according to the information collected, a simple simulator has been implemented to simulate the

sensor component. The reading is collected from the simulator periodically, when a user sends any queries. Any process applied on the data will be performed in the execution service. The simulator generates events for the purpose of the evaluation, such as where the out of range reading is corrupted, the sensor fails by sending a null reading, the reading is increased or the reading is decreased.

5.1.2 Application functionalities

Generally, the required functionalities may differ depending on the environment, the sensor used and how the sensor data is analysed. The following essential functionality requirements for the implemented system have been partly derived from existing sensor middleware systems (Madden et al. 2003, Loo et al. 2006) and partly from existing energy monitoring systems (Kappler et al. 2004, Harris et al. 2007):

***WSN Querying**:* the system provides a means of querying real-time sensor data, as well as offline database data, in an efficient way. A query can aggregate, filter and transform one or more data streams on behalf of the user, and generate a new stream.

***Presentation**:* keeping the user up to date with the system by presenting sensor or usage data.

***Sensor access**:* sensors have a communication protocol and different ways to connect, such as by a serial connection, or USB. In this system the sensor is connected to the board using a cable, and the board is connected to the computer through a USB cable (Phidgets). The system maintains a routing table with entries for all sensors containing a sensor ID and the analogue input number that the sensor is connected to. However, in our use case the application is connected to the simulated sensor component.

***Sensor discovery**:* sensors can be added to or removed from the system. The number of sensors is changed in the database, and accordingly the content of the routing table needs to be adapted in addition to the sensor count.

***Sensor specification**:* the system knows the sensor's characteristics such as output structure and location, as well as how to identify the sensor. The system records this information and also when changes to the configuration occur.

***Fault tolerance**:* with a sensor, data could be corrupted or communication could fail. The system should take this into account and react appropriately, by regularly scanning for missing sensors, and sending an alarm message in case of any fault.

Shared execution: many users may be involved in analysing sensor data. The system allows multiple users to access it at the same time and provides access control on sensor data.

Storage: a stream management system usually needs three types of data storage (Golab et al. 2003). Temporary storage should be in the memory for storing windows queries or caching. Disk space will be used in recording historical data and aggregated results, while static storage such as a relational database is used to store sensor metadata such as location, manufacturer and output specification.

5.1.3 Mapping to implementation

In order to include all the functionalities discussed earlier, the simulated application is implemented as a web server that allows for multithreaded execution in order to allow multiple users to access the system with an authentication security. The application provides a facility for adding or removing sensors and detects some of the fault tolerances such as when a sensor is missing.

In normal operation, the system runs a long-lasting WSN query for collecting the readings from all sensors. The routing table is populated in order to obtain access to sensors. The number of sensors is adapted according to the number of sensors registered. The reading is collected each second; it has a time-stamp and is given a sequence number and is then sent to the database. The sensor reading value is converted to the AC current and displayed to the user, and is then sent to the database.

A user can query data from sensors by building a query and submitting it. The WSN query planner accepts the query, translates it and sends it to the query execution engine. A WSN query planner service provides a declarative language

for specifying queries where the user describes what he/she wants by selecting the set of attributes and conditions such as the time, selected sensors and operation. The WSN query execution engine receives the query and starts to execute it. The result could be from the sensor or from the database, according to the exact query specification, and is sent back to the WSN query services in order to be presented to the user.

TinyDB extends and implements a query-based interface for extracting information from a network of TinyOS sensors. TinyDB query language is based on SQL, which refers to as TinySQL, and consists of a set of attributes to select, a set of aggregation expressions, a set of selection predicts for filtering, and a grouping expression for partitioning the data before aggregation. Aggregation is commonly used in a sensor environment. The five basic data aggregations are: count, min, max, sum, and average (Madden et al. 2002).

Our query interface has borrowed some TinyDB features, which are listed below:

- A set of attributes to select such as sensor IDs, or the duration of the query
- A set of aggregation expressions such as Summation and Average
- A grouping expression such as group by room number or sensor type.

A graphical interface, a query window, has been implemented for building queries and choosing the attributes and aggregation expressions to apply on the data as shown in figure 5.3.

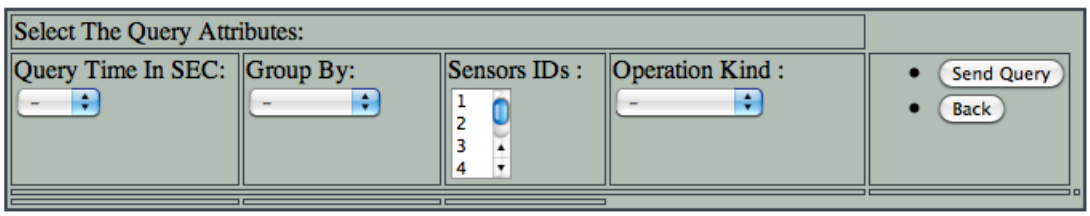


Figure5.3: Query windows of UGI

As shown in figure 5.4, the query time specifies the interval time in seconds, which is the query duration. A Group By drop list shows the available grouping expressions, which group by room number, by user ID or by sensor type. The sensor IDs list all sensors registered with the system and allow for selecting many items. The operation menu lists the available aggregation expressions, which are Summation and Average, and Reading is to get the reading of the sensor.

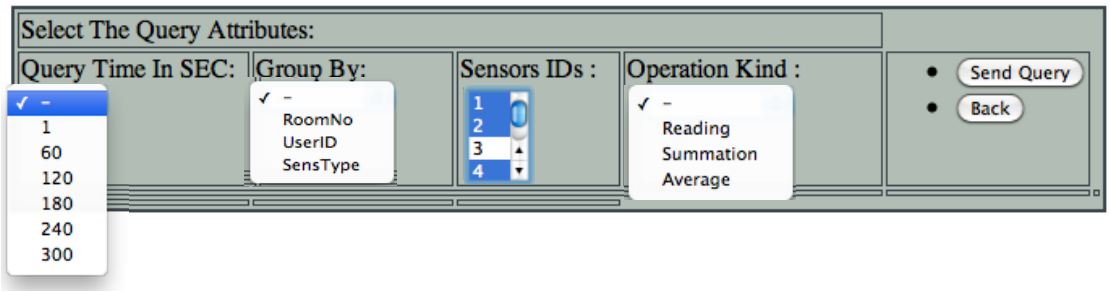
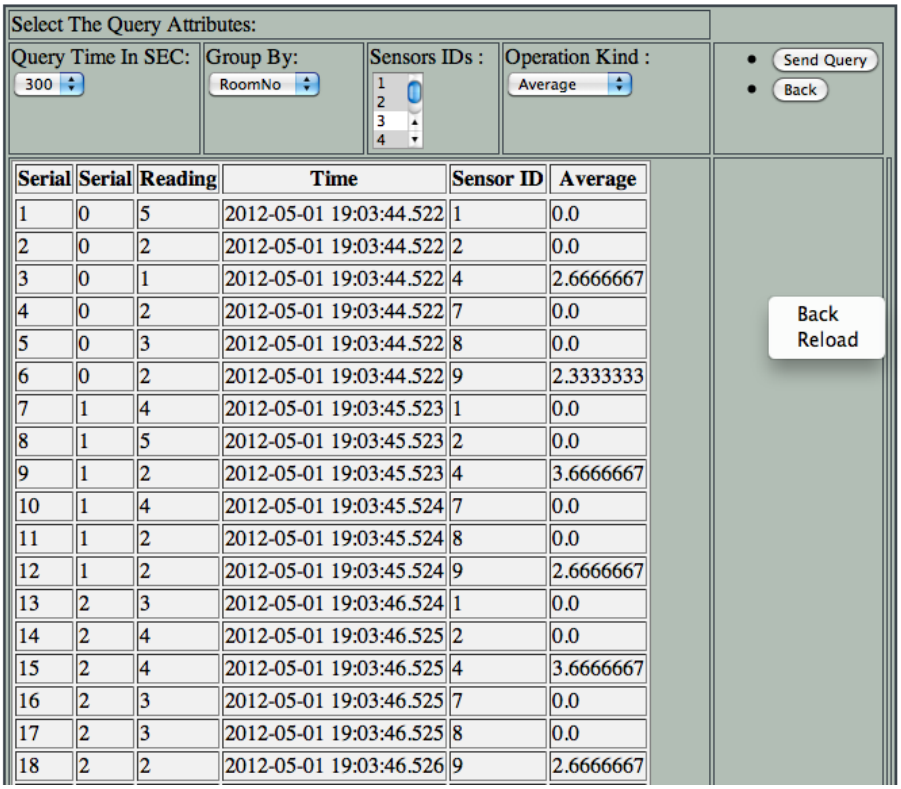


Figure5.4: Options to be selected for each attribute

For example, when the user requests the average reading of sensor numbers 1, 2, 4, 7, 8, and 9 group by the room number, this means the average is calculated for the sensor according to the room number. Therefore, if sensor numbers 1, 2, and 4 are in room no. 1, and sensor numbers 7, 8, and 9 are in room 2, the average is calculated for each room. The SQL statement is as follows:

```
SELECT RoomNo, Average
FROM Sensors
GROUP BY RoomNo
QUERY DURATION 5 min
```

Figure 5.5 shows the query window that constructs this query. The duration time is 300 seconds (5 minutes), the sensor has been selected by selecting the sensor IDs, a room number was selected as a group filtering and Average was selected as an operation expression.



Serial	Serial	Reading	Time	Sensor ID	Average
1	0	5	2012-05-01 19:03:44.522	1	0.0
2	0	2	2012-05-01 19:03:44.522	2	0.0
3	0	1	2012-05-01 19:03:44.522	4	2.6666667
4	0	2	2012-05-01 19:03:44.522	7	0.0
5	0	3	2012-05-01 19:03:44.522	8	0.0
6	0	2	2012-05-01 19:03:44.522	9	2.3333333
7	1	4	2012-05-01 19:03:45.523	1	0.0
8	1	5	2012-05-01 19:03:45.523	2	0.0
9	1	2	2012-05-01 19:03:45.523	4	3.6666667
10	1	4	2012-05-01 19:03:45.524	7	0.0
11	1	2	2012-05-01 19:03:45.524	8	0.0
12	1	2	2012-05-01 19:03:45.524	9	2.6666667
13	2	3	2012-05-01 19:03:46.524	1	0.0
14	2	4	2012-05-01 19:03:46.525	2	0.0
15	2	4	2012-05-01 19:03:46.525	4	3.6666667
16	2	3	2012-05-01 19:03:46.525	7	0.0
17	2	3	2012-05-01 19:03:46.525	8	0.0
18	2	2	2012-05-01 19:03:46.526	9	2.6666667

.

.

1777	296	5	2012-05-01 19:08:40.911	1	0.0
1778	296	4	2012-05-01 19:08:40.912	2	0.0
1779	296	2	2012-05-01 19:08:40.912	4	3.6666667
1780	296	1	2012-05-01 19:08:40.912	7	0.0
1781	296	3	2012-05-01 19:08:40.912	8	0.0
1782	296	1	2012-05-01 19:08:40.912	9	1.6666666
1783	297	5	2012-05-01 19:08:41.912	1	0.0
1784	297	1	2012-05-01 19:08:41.913	2	0.0
1785	297	4	2012-05-01 19:08:41.913	4	3.3333333
1786	297	3	2012-05-01 19:08:41.913	7	0.0
1787	297	3	2012-05-01 19:08:41.913	8	0.0
1788	297	5	2012-05-01 19:08:41.913	9	3.6666667
1789	298	1	2012-05-01 19:08:42.913	1	0.0
1790	298	4	2012-05-01 19:08:42.914	2	0.0
1791	298	4	2012-05-01 19:08:42.914	4	3.0
1792	298	1	2012-05-01 19:08:42.914	7	0.0
1793	298	5	2012-05-01 19:08:42.914	8	0.0
1794	298	1	2012-05-01 19:08:42.914	9	2.3333333
1795	299	4	2012-05-01 19:08:43.915	1	0.0
1796	299	2	2012-05-01 19:08:43.915	2	0.0
1797	299	3	2012-05-01 19:08:43.915	4	3.0
1798	299	3	2012-05-01 19:08:43.915	7	0.0
1799	299	5	2012-05-01 19:08:43.915	8	0.0
1800	299	5	2012-05-01 19:08:43.915	9	4.3333335

Figue 5.5: Query construction window with its result

5.2 Provenance subsystem

The application architecture can be extended to capture provenance by including the provenance component as shown in figure 5.6.

The Provenance Subsystem (Provenance component) is the implementation of the provenance model in checking, collecting and storing the provenance information of the measurement data and queries. When a new WSN query is submitted, the WSN query service parses it and sends it for query execution. The WSN query execution engine executes the query and records the sensor real-time data in a relational database. During the execution, data streams are subject to rate and accuracy changes (Vijayakumar 2006). The WSN query execution engine contacts the provenance services when the condition of the extended policy of stream event collecting is satisfied. The provenance service detects the event of interest that is specified in the policy and captures the changes in the stream and the association effect dynamically. As provenance data has to be permanently stored and maintained, the updated information with its time-stamp is stored as tuples in relational data tables. In addition, if a WSN query or any changes associated with it are specified in the collecting policy as an event of interest, the provenance service should go through the same process.

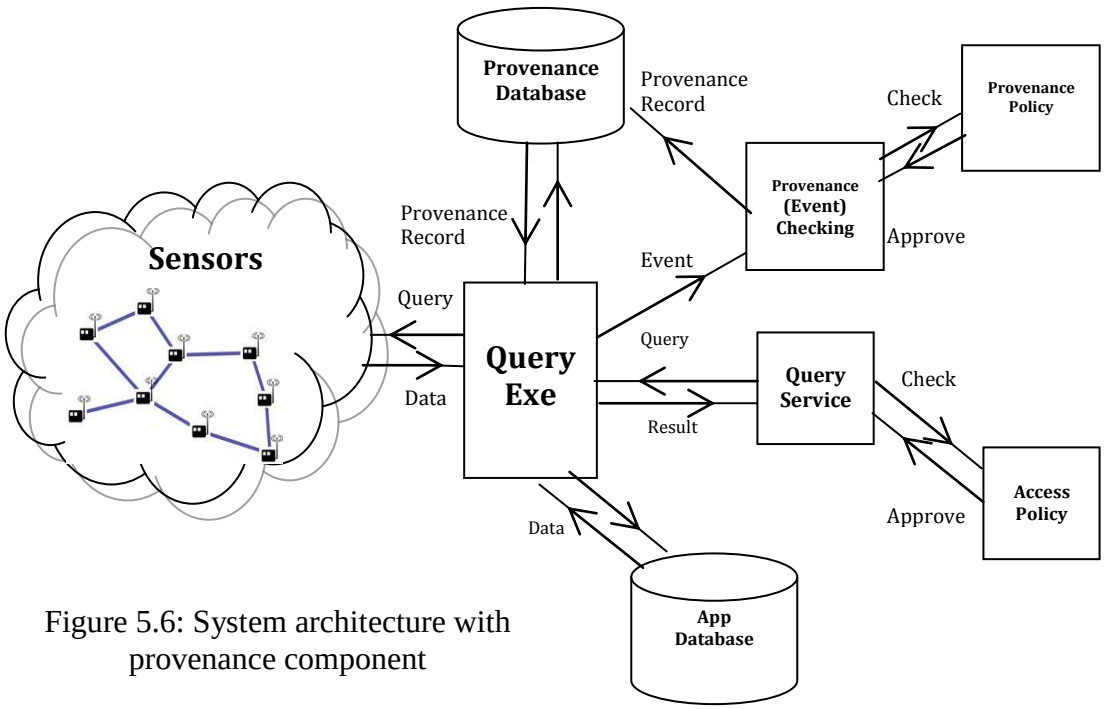


Figure 5.6: System architecture with provenance component

The query interface of the provenance component must support complex query facilities to search, analyse and reason over the collected provenance information. Different levels of query capability are required, such as a recursive query and a distributed query. Each stream or process registered with the system is uniquely identified by an ID. The full provenance information of a usage stream or a process can be retrieved using its own ID or WSN query ID, while a subset of an event associated with a stream or process can be identified by any information related to the event such as a type of change, or a change timestamp.

5.2.1 Data model

In the use case scenario, the provenance information consists of two parts: base provenance information and dynamic provenance information. The base provenance information is the information about sensors, networks, users, and administrative information, which is gathered when these are first registered with the system and is amended only when they are added or removed. The dynamic provenance information is the information about the usage and the changes, which is gathered during the measurement of the usage by sensors or during the execution of the process. The three atomic units of dynamic provenance collection are: streams, processes and events, while dependency is recorded to connect the input and output data to the process. Streams could be base streams, which are generated by sensors, or derived streams. A process is a transformation applied to the data, while an event can be related to processes or streams.

The provenance information of a stream contains the sources that generate it, the owner of the stream, the stream start time and the stream end time. For the process, the provenance information contains a list of input streams that the process has been applied to the type of process, start time, end time and the derived stream as an output. The event information contains its type, its time-stamp and where it happened.

The additional provenance information about sensors, phidgets and networks can be used to describe the environment when the action happened. This information could be added to a process as an annotation within the provenance graph, in order to record the state of a process. Sensor provenance characteristics may include hardware information such as: type, lifetime, and phidget kit interface; and software information such as: time period for each reading, time-stamping and reading package. The user information contains the number of sensors assigned to measure that user's usage and other information required by the application.

The dynamic provenance information needs to be stored in order to be used and queried to reconstitute the provenance of some data or process; therefore, it is saved in a relational database management system for the advantages discussed in section 2.2.3. Provenance information can grow to an immense size and needs to be supported by efficient management tools. MySQL has been considered because it is a popular choice for use in many high profile, large-scale World Wide Web products such as Wikipedia, Google, Facebook and Twitter (MySQL 2011). It is an open source, which works on many different system platforms and provides a full-featured database management system.

A table is created for each of the three provenance entities and for the dependency relationships:

- The process table is used to store the provenance information of the process and each row represents the attribute information of a process primitive
- The stream primitive attribute is stored in a stream table, which is used to store the provenance information of a source or derived streams
- The event table is used to store event information specified in the event primitive
- The dependency table is used to store the dependency relationships with the specific attribute in the dependency primitive.

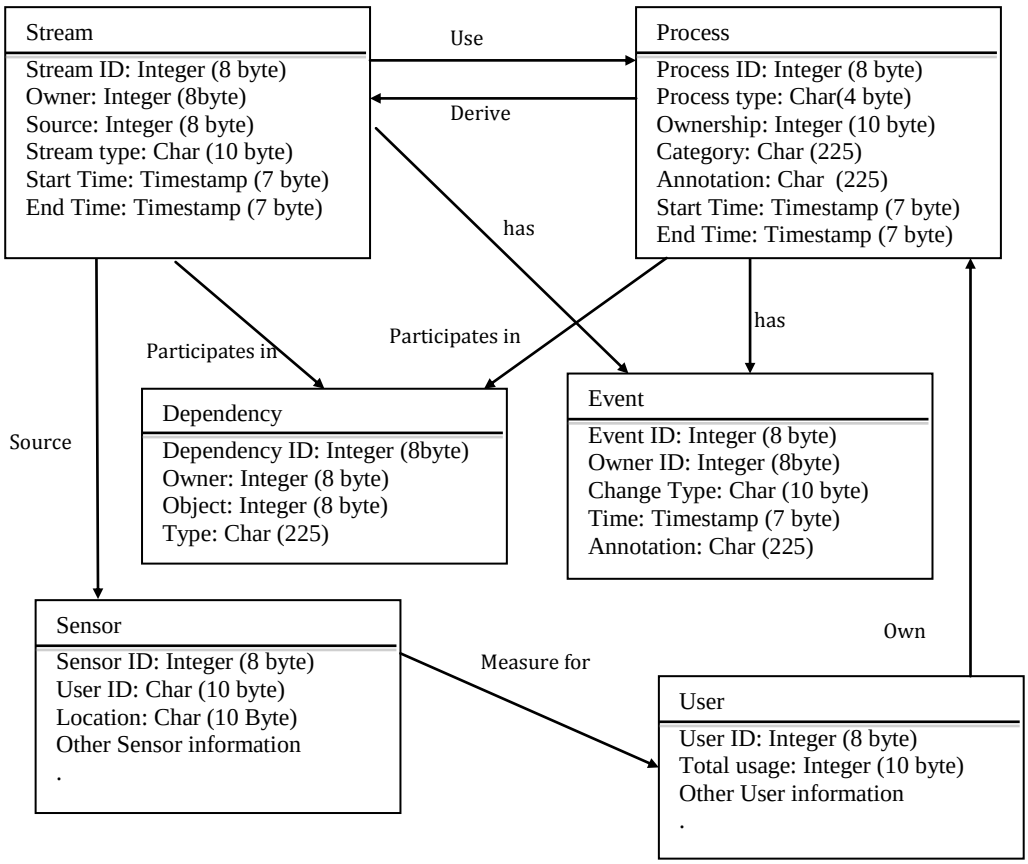


Figure 5.7: Provenance Data Model

The information in these tables needs to be connected to information in the sensor and user tables. A stream contains change events and participates in process input and output. A process has changes, accepts inputs, and generates a new stream. Sensors generate source streams, and sensors and queries have an owner. The data model of the proposed solution consists of the major entities, as shown in figure 5.7, to keep track of the provenance information.

5.2.2 Implementing the collecting model

Recording provenance is controlled by collecting policy definitions as specified in Chapter 3. These policies can be created and be active or inactive at any time in the application lifetime and controlled by the system administrator.

To validate the set of policies in the recording phase of stream events, we built a domain specific language that is used to define events of interest and their attributes using ANTLR (Parr 2007). Grammar is the highest-level construct of ANTLR, which is a list of rules describing the structure of the policy language. ANTLR automatically analyses the grammar and generates the lexical analyser and parser. *Lexical analysis* is the first phase in the translation, which breaks up the incoming stream into *tokens*. *Parsing* is the second phase, which operates on these tokens and tries to recognise the sentence structure.

By recognising these policies, ANTLR generates executable DFAs that match the stated policies, and these are used to filter the incoming stream of reading for event detection. The whole code is written in Java, which requires an ANTLR jar file to run.

Two grammar files are created: one file for recognising the definition of the event (Appendix 1), and the other for recognising the policy of capturing provenance (Appendix 2). The following is an example of an increase event definition:

```
event IncreaseEvent {
      Long totalChange;
}
ReadingEvent[n].value > ReadingEvent[n-1].value
=>
IncreaseEvent.totalChange        =        ReadingEvent[n].value-
ReadingEvent[n-1].value;
```

This definition implies that when the current reading is greater than the previous reading the difference value is caught. The grammar file reads this file and translates it to create the DFA states. An example of a policy definition for capturing the provenance information of this specific event is shown below. The policy states that if the difference between the current reading and the previous reading is greater than 2 then the provenance information of that event is captured.

```
policy Increase {
IncreaseEvent.totalChange > 2
} capture IncreaseEvent
```

The event file and the policy file are a well-defined set of sentences that represent the language. ANTLR builds a recognizer which checks that the sequence input sentences from these files follow the rules of the language described by their grammar file. According to the definition of the event, an array list is created for its states transition table. The array list start with a start state, and has an acceptance state depending on the translation of the event expression and the specification of its condition and attribute. The event attribute defines the number of the value that needs to be held during the checking process. In our example, one value is being held, which is the previous value (n-1) in order to compare it with the current value (n). Each state has its attribute and transaction, which is stored in another array list. At the beginning, the start state becomes the current state. When the first reading is put in, the compiler checks its attribute and according to that the transaction will be decided, which could mean moving to the next state, staying in the same state, or whatever it is according to the event state transition table, and this will be the new current state.

In the example, according to the parser event file, an array list of the DFA states is created and the accept state is reached when the attribute of the parse policy file has become true, as shown in figure 5.8.

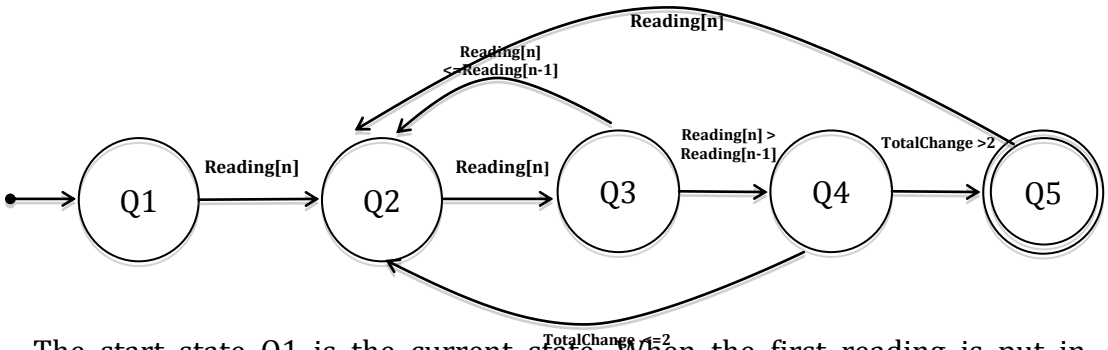


Figure 5.8: DFA for event increased and policy increase

The start state Q1 is the current state. When the first reading is put in, the transition moves to state Q2, and it become the current state. When the second reading is put in, the transition moves to state Q3. State Q3 had to make a comparison in order to decide what the next state would be.

IF Reading.event[n].value > Reading.event[n-1].value THEN moves to state Q4

ELSE moves to state Q2 hold the last reading and wait for the next reading

If the current state is Q4, the action is:

IF TotalChange >2 THEN moves to Q5, which is the accept state

ELSE moves to state Q2 waiting for the next reading

At state Q5, the event provenance information is captured. Then the current state will be Q2, hold the last reading and wait for new reading input.

The recording policies for events associated with a WSN query are dependent on actions such as WSN Query Submit, or WSN Query Stopped, which do not require specific language like the events associated with streams. Therefore, when this action happens, the provenance service checks the policy and performs according to this. In relation to the first scenario example, when the user sends a WSN query required to calculate the average of the consumption energy of his/her room, which includes four sensors, for one hour: when the WSN query instance is detected, the provenance component will check the active policies and filter this event according to that. The function Confirm is used for the filtering:

```
Confirm (event) IF condition
```

In this case, the event is WSN Query Start, so the function will be:

```
Confirm (query start) IF (policy is active)
```

If the policy is active, then the function will return (True) and the provenance information is recorded within the context of the WSN query instances. The recording model will annotate the three types of entities and their relationship, as illustrated in the following algorithm 5.1. The query process

primitive is mapped to the Insert Process function that records the WSN query information specified in the primitive and stores them in the process table in the database. The data primitive, for each input stream and output stream, is mapped to the Insert Stream function and saves all the input and output stream information in the stream table. The dependency relation between the input stream and the process, and between the process and the output stream is recorded in the dependency table by the Insert Dependency function.

If the policy of recording detailed information is active, then the summation process and the division process with their input and output stream and dependencies will be recorded.

Algorithm 5.1 The Provenance Collection Graph

QI = {qi}: a set of queries instances,
G = {*(A, P, AG, US, GN,T, D, C* }: the provenance graph in OPM.

```
1:    for each query instance qi in QI do
2:        if policy = true then
3:            add agent AG to G,
4:            for each process instance process i in qi do
5:                add process Pi to G,
6:                add dependency "ControlledBy" between AG to Pi in G,
7:                add dependency "Trigger" between Pi-1 to Pi in G,
8:                for each input data input j of process i do
9:                    add artifact Aj to G,
10:                   add dependency "USedBy " between Pi and Aj in G,
11:               end for
12:               for each output data output k of process i do
13:                   add artifact Ak to G,
14:                   add dependency "GNeratedBy" between Pi and Ak in G,
15:                   add dependency "Derived" between Aj and Ak in G,
16:               end for
17:           end for
18:           for each event in qi do
19:               if event condition is satisfied then
20:                   add artefact An to G
21:                   add dependency "Derived" between Aj and An in G,
22:               end if
23:           end for
24:       end if
25:   end for
26: output G.
```

The algorithm uses the submitted WSN query as an input and uses it to identify the set of active recording policies (line 2). If the policy is active, the provenance service records the WSN query information with its input and output (lines 4 -17). For each event associated with streams or process, the provenance services record the events that match the condition of the stated policies (lines 18-23).

5.2.3 Implementing the provenance query component

Two models are used for querying provenance information: retrieval query and filter query. Retrieval query is used when the complete provenance information of a process or artifact is involved, while the filtering model is used for searching for specific provenance information using information filter criteria. These two models require provenance systems to support complex queries such as nested sub-queries and aggregation (Glavic et al. 2010). Different levels of query capability are required, such as a recursive query and a distributed query. However, a distributed query was not used in our provenance component, since not all the provenance data was distributed in different database services.

In our use case, the provenance query was planned to implement an interface that allows us to query the provenance information of a query, a stream, a sensor, a process, and a user's usage, as shown in figure 5.9. However, only the query provenance information has been implemented with all options shown in figure 5.10.

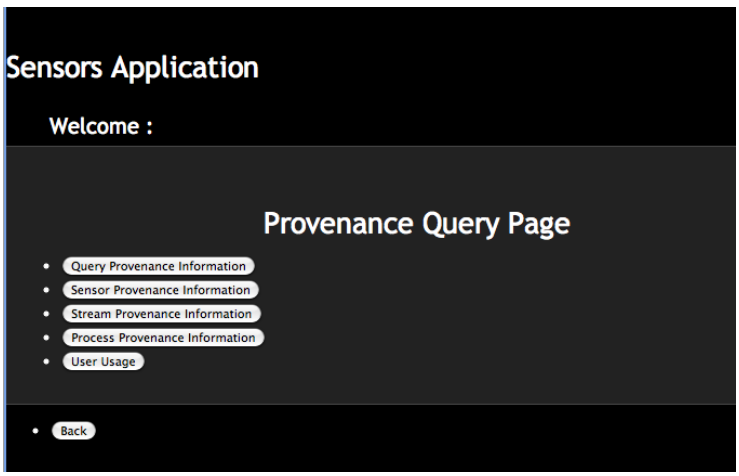


Figure 5.9: Provenance query Page

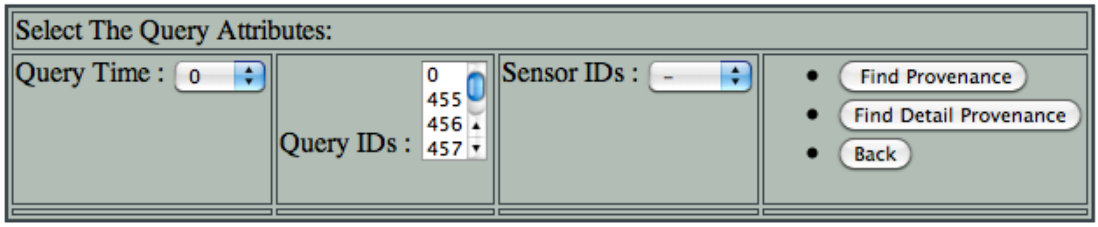


Figure 5.10: Provenance query construction window

Choosing the query ID can retrieve the complete provenance of that query. The query provenance information can be filtered by choosing the duration time of the query or the sensors that generate the source stream used by the query. For example, when the user requests the provenance information of a query that lasts for 1 hour, in this case we have a nested provenance filtering query.

```
SELECT *
FROM Dependency,
     (SELECT *
      FROM ( SELECT *, end_time-start_time AS Duration
             FROM Process
           ) AS InnerQuery
      WHERE Duration <=60
      ) AS Queryreq
WHERE Queryreq.process_id = Dependency.owner
```

An example of using a recursive query is when the user requests the provenance information of all the processes that use the stream generated by sensor number 10 directly as an input, or any process that uses a stream derived from a stream generated by sensor number 10. The recursive query is over a table stream, that contains information about streams and the processes related to those streams. The query returns the process that directly or indirectly uses a stream generated by a specific sensor by recursion from the sensor ID. That is defined by a Union All with an initialization full select that seeds the recursion and an iterative full-select that contains a direct reference to itself in the FROM clause.

```
SELECT *
FROM Process,
           (SELECT Stream.owner
            FROM Stream
            WHERE Source =sensorID AND type = 'source'
```

```
            UNION ALL

            SELECT Stream.owner
            FROM Stream, (SELECT *
                                    FROM Stream
                                    WHERE Source = sensorID
AND type = 'source'
                                    ) AS InnerQuery
            WHERE Stream.Source= InnerQuery.owner) AS
NextQuery

WHERE NextQuery.owner = process_id
```

5.3 Access control model

As mentioned before, provenance information needs its own right of access. The proposed access control model is based on the RBAC specification, which can provide fine-grained control over a tuple or attribute in the provenance database. As the application uses MySQL for store provenance information, the access control model is also built in MySQL.

5.3.1 Data model

In order to implement the proposed access control in MySQL, all the data is stored in tables. These tables need to be created in order to define the three main entities of the RBAC: the user, role and permission:

- Table for *User*, who needs to obtain access to the provenance data. Users could be the application users or outside third parties.
- Table for the *Roles* specification, which matches the application requirement as mentioned in Table 4.1.
- Table for the *Permission* specification, which defines the semantics of permission as described in Table 4.2.

Four additional tables are needed to specify the assignment, relationship and conditions:

- Table for the *Role-User* assignment, which assigns each user to a role, based on their responsibilities and authority, as in Tables 4.6 and 4.8. The

assigned relationship between the user and their role is 'many-to-many', which means the user can be assigned to many roles and one role is assigned to many users.

- Table for the *Role-permission* assignment, which assigns roles to the permission and specifies which access is approved for each role. Again, the relationship between role and permission is 'many-to-many', which means a role can be assigned to many permissions and a permission can be assigned to many roles.
- Table *Condition*, which specifies the condition of each permission according to the policy rule, as mentioned in Tables 4.3, 4.4 and 4.5. The Condition table is needed because some permissions can be allowed by more than one condition; for example, in our case, the WSN query input/output can be accessed if the user is the owner of the WSN query or a friend of the owner or a supervisor of the owner.
- Table *Relationship*, which specifies the relationship between users, such as which user is a friend of which other user. This table is used to check the condition of some permissions; for instance, the permission is allowed if the provenance query sender is a friend of the data owner.

Figure 5.11 shows all the tables and the relationship between them. These tables, which are to maintain the state, are required for permission evaluation. The *Role-User* table, which contains a mapping of the user identifier and the role identifier, maintains the set of active roles of each user session. This role identifier indexes into the *Roles* table. The *Role-permission* table maintains the permission assignment and the role numbers.

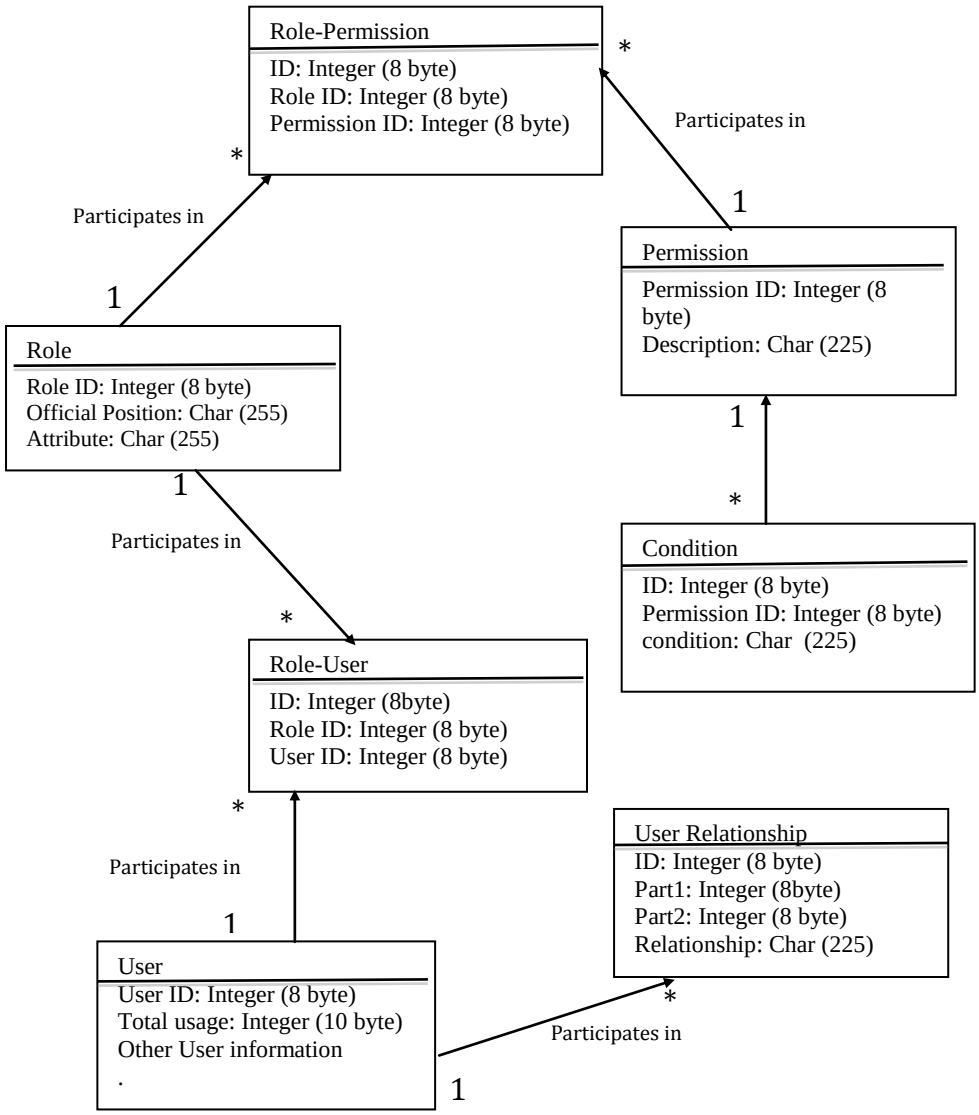


Figure 5.11: Access control data model

5.3.2 Mapping to implementation

When implementing the access control model, a permit function is needed for permission evaluation. Each request query to the provenance database, which is sent by a user, needs permission evaluation and responses of ALLOW or DENY. When user1 signs into the application and starts a new session, he/she has all the access rights of all roles to which he/she is assigned. The roles activated for that user's session, as mentioned in Table 4.6, are A and B, therefore all permissions assigned to these two roles are activated.

According to the first scenario example, user1 sends WSN query1, which requests the average to be calculated. If user1 sends a provenance query requesting the detailed provenance information of WSN query1, then by following algorithm 4.1, the Permit function will go through all the permissions in each role and check. If the permission is found, the condition for accessing the detailed provenance information is that the user who requests the information is the owner of the WSN query. In this case, the condition is met in permission 2 of Role B, because user1 is the owner of the WSN query. Therefore, the function will return ALLOW and the user will have access to the information he/she has requested.

The second example, which records the provenance information of a user1 sensor data stream, the stream has four events recorded in a period of time. In the case where user2 logs onto the system and requests the provenance information of a sensor stream related to his/her friend user1, the permit function will check all the permissions and detect permission 7 for role B. However, the condition for this permission states that the user who requests the information has to be the owner of the sensor that generates the stream. Therefore, the function will return a DENY message, because the condition is not valid and the loop of checking all the permissions of each role has finished and none of the conditions has been satisfied.

5.4 Summary

This chapter has discussed the experimental setup requirements for evaluating the proposed provenance collection model and access control. The electrical monitoring system and sensor network have been simulated and implemented in order to be used for the evaluation stage. The provenance component can be added to the application architecture with little modification. The provenance subsystem architecture is based on collecting and data models. The data model of the proposed solution consists of six major entities to keep track of the provenance information. The collecting model is based on an event alert and these events are filtered according to the policy language specified in Chapter 3.

The implementation of the collecting model involves building a small domain-specific language for capturing events associated with streams using ANTLR. The access control data model consists of seven entities which are used to maintain the required information for applying RBAC. The implementation is mapped to the implementation algorithm in Chapter 4.

CHAPTER 6 EVALUATION

In this chapter, we describe how the first set of experiments were set up to evaluate the overhead of the proposed collecting model and to measure its scalability. The second set of experiments were set up to evaluate the time response of the provenance query. The storage overhead was evaluated by a numerical analysis.

Several techniques have been used to meet the provenance requirements for individual domains. Based on a survey of the literature on provenance, provenance systems can be analyzed and compared according to several taxonomies (Simmhan et al. (a) 2005). These taxonomies are based on why the provenance is recorded, what it describes, how it is represented and stored, and the way to disseminate it. The subject of provenance and its representation affect the cost of the collecting process, while the manner in which this information is stored is important in relation to its scalability.

The management of provenance incurs costs for its collection and storage (Simmhan et al. (b) 2005). Therefore, the main parameters for evaluating our proposed system are collection and storage overheads and provenance query performance.

In order to evaluate the provenance recording and querying performance, the simulated use case has been built that queries a simple sensor simulator, as mentioned in the previous chapter. The simulated sensor network works as sensors that send a reading periodically when a user sends any queries. The simulated application consists of a WSN query service, WSN query execution engine and provenance service. The user builds his/her WSN query at the WSN query service; the WSN query execution engine connects the WSN query service and the sensor simulator. The provenance service detects the event of interest and records its provenance information.

The simulated sensors and the application are written entirely in Java and hosted on MacBook Pro with Intel Core 2 Due 2.66 GHZ and 4 GB Memory. The application is deployed within a Tomcat 6.0 web server container and uses a MySQL database.

The experimental evaluation was performed in different stages and for different purposes. The first set of experiments measured the provenance-collecting overhead and measured the scalability of the provenance service in tracking and recording the provenance information. The second set was for evaluating the response time of the provenance query. The storage overhead is evaluated using a numeral analysis.

6.1 Provenance recording evaluation

6.1.1 Collecting overhead

In our system, provenance management is transparent, which means the users are freed from manually recording the provenance information. The system middleware will automatically capture the provenance of the sensor stream data, and this should be during the data creation process (Simmhan et al. (a) 2005). Instrumentation could incur a performance loss (Vijayakumar et al.2006), so ideally, a collection overhead could be imposed on the normal functionality of the system, and has to be reasonable.

The collection process overhead can be defined by measuring the time taken to record the provenance when an event of interest occurs. For instance, when a sensor reading changes, which means a user starts to use electricity, the provenance component sends this information such as the change value and the time to the database. When the reading goes down, which means the electricity stops being used, the time and the information about the change is recorded. Any other changes that happen during the usage, such as packet loss or connection failure, should be recorded also as provenance information. The time of recording all the provenance information of electricity usage compared to the overall execution time of the system is the overhead of provenance collection.

The other case is to record the time that is taken to record the provenance information of a WSN query. For the purpose of this experiment, the WSN query service generates a WSN query and deploys it to the execution engine. The WSN query time is measured by recording the time it takes from starting to send the WSN query until it has finished, which includes the time it takes to record any provenance information. The overhead is to compare the provenance recording time with the query execution time without recording the provenance information.

6.1.1.1 Process provenance

The first experiment is to measure the time taken to register the WSN query information at the provenance service. Registering a WSN query includes checking the validation of the policy and recording the WSN query as a process in a process table, then recording the information of each sensor stream in the stream table. This is the standard process for each WSN query. In order to measure the time that the system takes to check and register the WSN query information with the provenance service, the WSN query service generates 120 WSN queries sequentially that request a reading from a sensor. The time is measured from submitting the WSN query until it finishes by displaying the reading. We assume that the simplest WSN query is for obtaining the current reading from one sensor, and the WSN query and the stream are the only events

that need to be recorded. In chart 6.1, the X-axis shows the recording time in milliseconds (MS) and the Y-axis shows the serial number of the WSN query. The recording time is between 5 and 10 milliseconds.

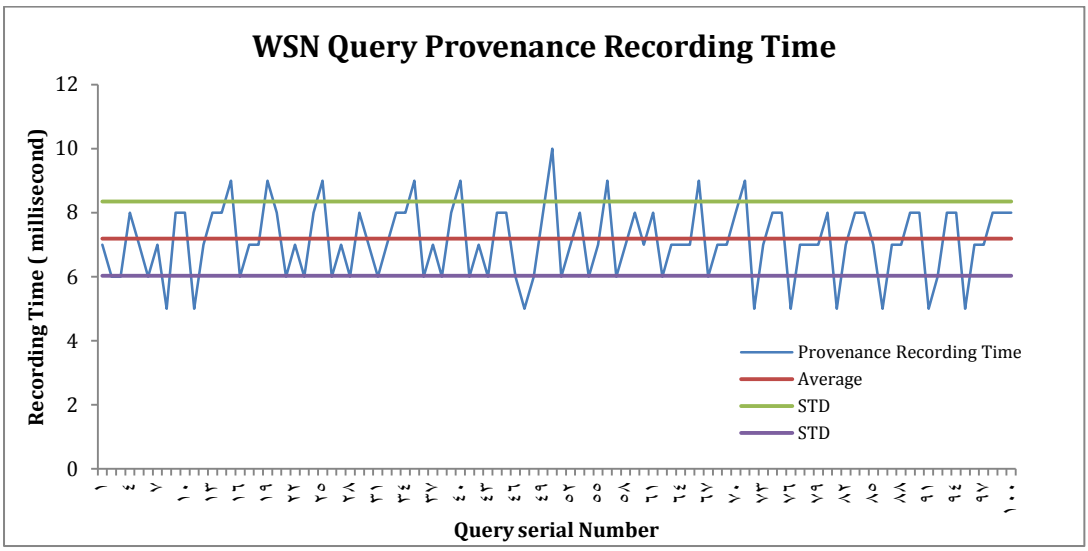


Chart 6.1: WSN Query provenance recording time

The range can be calculated as:

Range = highest value which is 10 – lowest value which is 5 = 5

The mode is 8 milliseconds, which has the highest frequency accruing of 95 out of 120, in the frequency distribution.

To calculate the recording time standard deviation, the difference of each time from the mean has to be calculated, and the result of each has to be squared.

The average is 7.19

The standard deviation is 1.16

The above average standard deviation is 8.35

The below average standard deviation is 6.03

Table 6.1 below shows the overhead of registering the WSN query, which imposes an overhead of 7.19 MS on the overall query execution, about 10.8% of the total time, which is reasonable (Vijayakumar et al. 2006, Groth et al. 2005).

Measure	Average (MS)	STD (MS)
WSN Query execution time	66.34	4.63
Recording WSN query information	7.19	1.16

Table 6.1: WSN Query Provenance Overhead

The time required for recording query provenance is the time for checking the policy and contacting the database for persistent storing of the query information. By breaking down the cost of recording the provenance information of the query, the result shows that only 0.6.5 milliseconds of the total time for provenance recording, which is 7.19 milliseconds, is for performing the checking process. The higher percentage, which is not under control, is used to locate the database server, to establish a communication channel with it, and for exchanging information; it is called the database connection overhead. This overhead is not fixed and depends on different factors such as data traffic.

6.1.1.2 Event provenance

The second experiment is to measure the time for monitoring and checking an event and then recording the provenance information. A single WSN query was set up and run, and a single stream was recorded. Events were generated every five seconds for the provenance service to check the policy and record the event information. In this experiment, events were changes in the reading value and null reading, as the overhead time was similar in these cases. The experiment lasted for 10 minutes and 100 events were generated. The time that the provenance service took to record event information was between 1 and 5 milliseconds. The chart 6.2 shows the recording time of 100 events, the X-axis shows the time in milliseconds and the Y-axis represents the event serial numbers.

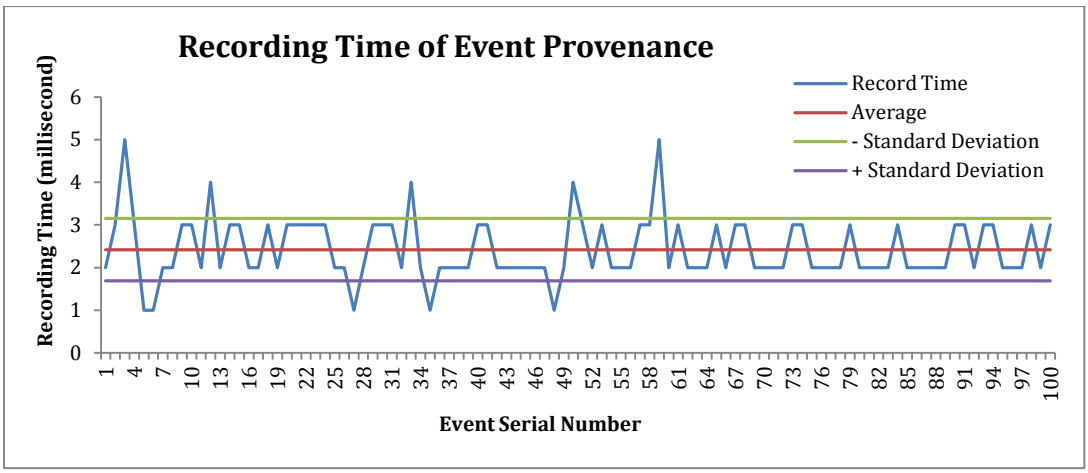


Chart 6.2: Event provenance recording time

The range = highest value which is 5 – lowest value which is 1 = 4

The mode is 2 milliseconds, which has the highest frequency accruing of 65 out of 100, in the frequency distribution.

The average is 2.42

The standard deviation is 0.62

The above average standard deviation is 3.15

The below average standard deviation is 1.69

As these results show, there are vast differences between event and query provenance recording, in both time consumption and storage overhead. The average time required to record query information is 7 milliseconds, while 2.42 milliseconds is the average time for recording the event provenance information. Query provenance has a higher overhead in time and storage. That is because recording query information involves recording a query as a process provenance, the reading of each sensor as an input stream; and a derived stream, in the case of the query requiring a transformation on the input streams as an output stream. The query that is used in the evaluation of recording query information is an example of the minimum query information, which is a query of one reading of one sensor. This involves recording the information of the query in the process provenance table, and the reading of the sensor in a stream provenance table. Two tuples are recorded and the database has been contacted twice. The event

provenance, which records changes that have occurred in the stream data, requires the recording of one tuple in the event provenance table.

6.1.2 Collecting scalability

Performance scalability is the ability of the system to scale with an increased number of sensors and events. The experiment will measure the overall execution time measured from when the WSN query is sent until the result is displayed (Groth et al. 2005). The time includes detecting the event condition and updating the database in order to keep the system updated with the event provenance information. The scalability performance can be done by analysing the process time to record events information with an increasing number of events and the size of the data recording (Groth et al. 2005). The experiment can show an increase in the execution time with an increasing number of sensors. Accordingly, it can show the scalability of provenance collecting to an increasing rate of event recording.

In the experiment, in order to show big number of sensors, the WSN query was for calculating the average of different numbers of sensor readings, starting from 10 sensors and going up to 100 sensors with an increase of 10 sensors each time, and for different configuration recordings:

- Without provenance recording
- With recording provenance information of the WSN query, sensor stream, the average as a process, and the dependency.
- With detailed information of the process for calculating the average such as: the summation process and division process, and the input and output of each process.

Chart 6.3 plots the execution time in milliseconds on the X-axis and the increasing number of sensors on the Y-axis. The observations are as follows:

- Overall, the different execution times remain linear with the number of sensors to be processed. Each plot has a correlation coefficient greater than 0.99;

- Overall, the overhead of provenance performance remains less than 20%;
- The granularity coarseness has an acceptable cost. However, the developer has to schedule the required granularity that offsets the overhead.

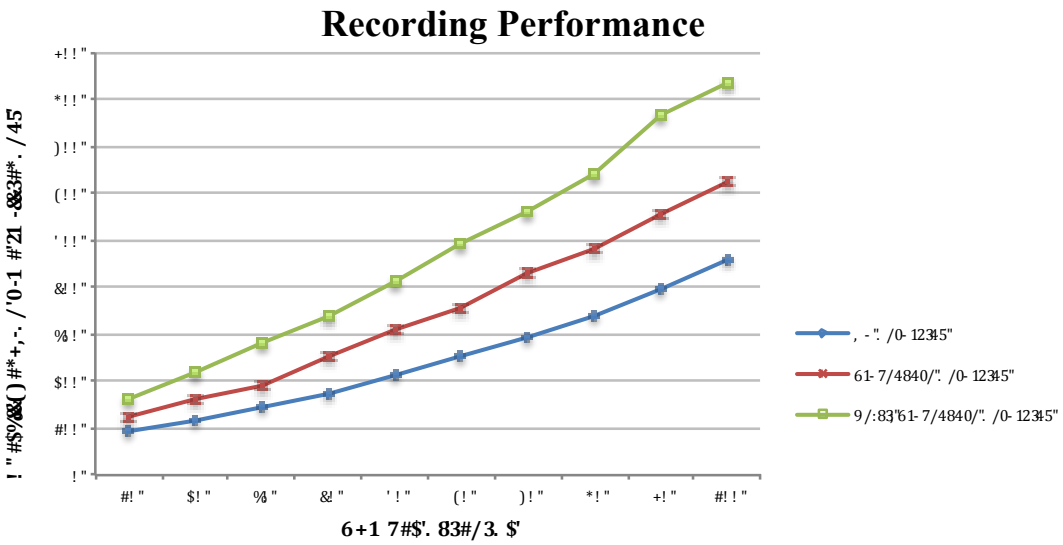


Chart 6.3: Recording performance

In order to show big number of sensors, other set of experiment is performed for calculating the average of different numbers of sensor readings, but starting from 100 sensors and going up to 1000 sensors with an increase of 100 sensors each time, and for the same configuration recordings as above.

Chart 6.4 plots the execution time in milliseconds on the X-axis and the increasing number of sensors on the Y-axis. The observations are as follows:

- Overall, the different execution times show still linear behavior with the number of sensors to be processed. Each plot has a correlation coefficient greater than 0.99;
- The overhead of provenance performance is between 20% and 40%. The overhead is increased with the increased number of sensor. The overhead start to increase at recording the provenance of a query

involved with 300 sensors. However, 4.89 seconds is reasonable time when dealing with 1000 of sensors;

- The detailed provenance recording almost cost double provenance recording overhead starting at dealing with 300 sensors. However, granularity coarseness still has an acceptable cost of 5.88 second when collecting details information of a query deals with 1000 sensors.

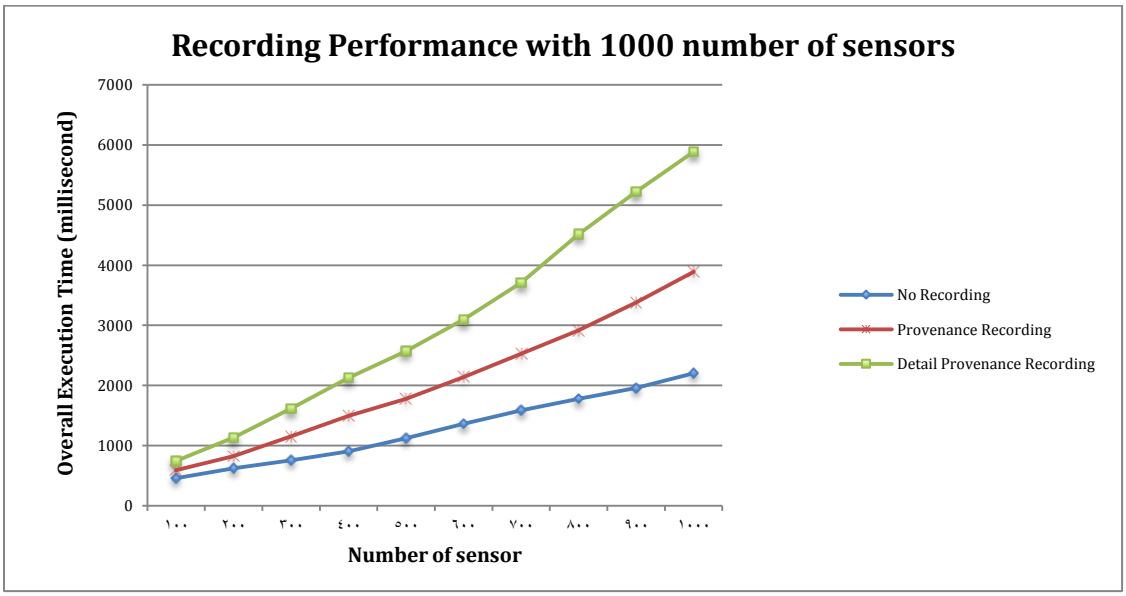


Chart 6.4: recording performance with 1000 sensors

6.2 Provenance query performance evaluation

Users can use the provenance information and search the database in order to locate the information they are interested in. The provenance query performance is evaluated by measuring the request response time. The time that the system takes to process the provenance query and search the provenance database then send the result of the user provenance query is called the request response time. Two factors can affect the provenance response time: the size of the provenance store and the size of the result. Therefore, the response time is measured against the different number of records contained in the store (Groth et al. 2005). Secondly, different queries with different result sizes are examined in

order to determine how these factors can increase the time required to search and respond (Groth et al. 2005, Simmhan et al. 2006b).

The size of the data result is built on three levels of provenance documentation details: a data provenance, a process provenance with its dependency of input and output, and the details of all processes that are performed on the input for a specific output.

The experimental use case provenance query was the provenance information of an average result. The provenance information was, for example: the streams that contribute to this result, the sensors that generate the stream, and the events that accrue in a stream. More detailed information, such as the process that the calculation goes through, includes: the summation process and its input and output, the division process and its input and output, and the process that uses this result (the result was its input).

6.2.1 Size of interaction record

The first experiment is to measure the response time for querying the provenance information of an average process and to perform a comparison against the number of interaction records contained in the store. The time taken to retrieve the provenance information is dependent on the size of the store. The response time has been measured when querying simple provenance information such as the input data, the output data and the average process. The detailed provenance information includes the input data, the summation process, the result of the summation process, the division process, and the output of the division process.

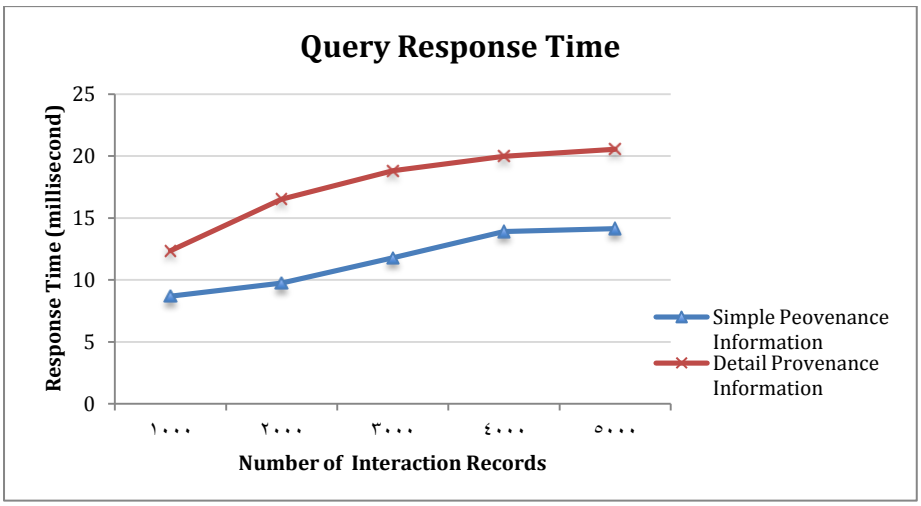


Chart 6.5: Response time with the increase in the store size

The time to query the store for these two levels of provenance information was measured against the increased number of records in the store starting with 1000 of interaction records and increasing toward 5000 records. Chart 6.5 shows linear behaviour with the size of the store, which means with the increased size of the store the response time is also increased. The size of the interaction records with the response time of querying simple provenance information has a correlation coefficient of 0.98, while it has 0.94 with the response time of querying detailed provenance information. The findings also show a reasonable response time even with the increase of store size, since with 5000 records in the store and the detailed information request, the response time is 20.54 MS.

6.2.2 Size of result records

The second experiment is to examine how the increase in the result size can affect the response time. The number of records in the store was 30,000 records, while the results started from 10 records and increased towards 100 records. The first case was performed by increasing the result size while the number of SELECT statements remained the same, while the other case was performed by increasing the number of records in the result and SELECT statements. As can be seen from Chart 6.5, increasing the number of records in the result has a low increase in the response time, while the increase in the SELECT statement in the second case takes more time to retrieve the required

result. Thus, the increase in the result size has little impact on the response time if it does not require many interactions with different tables.

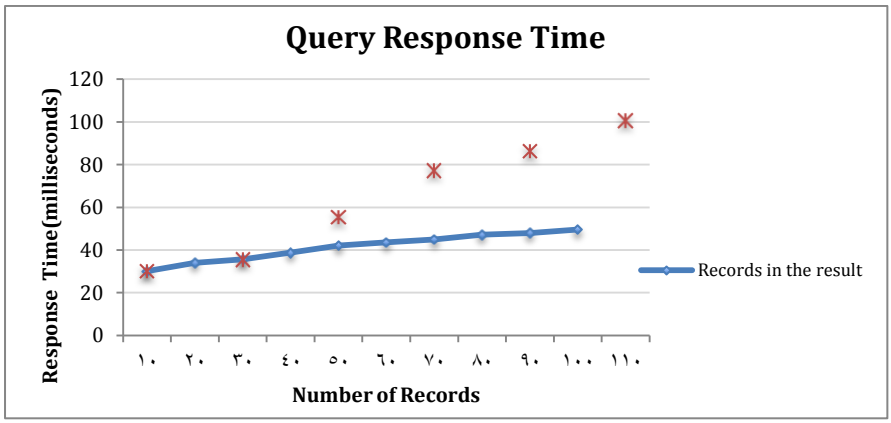


Chart 6.5: Response time with increase in result size

6.3 Storage overhead evaluation

The size of provenance information can grow larger than the data it describes and this has an effect on storage cost. The storage overhead of the provenance collection can be examined by measuring the size of the provenance records when any event accrues. Each event involves the creation of one or more provenance records, and each record is one row in a provenance database table. As we use MySQL for the provenance storage, each row needs a fixed number of bytes depending on the attribute type of each record. For example, a row in a stream table needs 26 bytes, as shown in Table 6.2.

Provenance systems can scale with the number of datasets, their granularity and depth of lineage (Simmhan et al. 2005b). The annotation method has been used with some restrictions in order to address the problem of high storage and process overheads. In pervasive systems, provenance storage scale with the number of sensors, the number of events, their granularity, and the depth of the lineage. The number of events needed for provenance recording is domain dependent and related to the granularity approach that the system adopts. The storage cost of recording this information increases exponentially with the depth of provenance. In our solution a fine-grained model has been used which could lead to outsize storage. However, it has

been reduced by recording just the event of interest, which is specified in the policy language, instead of all events over all time. Moreover, the dependency relationship between events is not explicitly recorded in order to minimize the consumption storage. However, that may require a recursive inspection of the provenance of each event in order to assemble the complete provenance.

Attribute	**Type**	**Storage required**
Stream ID	INT	4 byte
Source	INT	4 byte
Start time	Timestamp	4 byte
End time	Timestamp	4 byte
Owner	INT	4 byte
Type	var char	10 byte
	Total	26 bytes

Table 6.2: Storage required for stream provenance

This section will present a simple numerical analysis of the storage overhead of the annotation-based provenance vs. our proposed solution. Clearly, each WSN query must carry the ID of all the input and the output streams, in addition to the WSN query information. TVC (Wang et al. 2007) records provenance information based on stream segments, which are bounded by time intervals, rather than at the data element level. Over a specific time interval *t,* the output is periodically generated and the provenance of the process is recorded. When using the annotation approach in this instance, the overall overhead storage will be the total overheads of all streams of provenance produced in *t* additional to the cost of total per output overhead. If a WSN query lasts for T of time, which includes many time intervals *t,* then the total overhead of the provenance of the WSN query is calculated as follows:

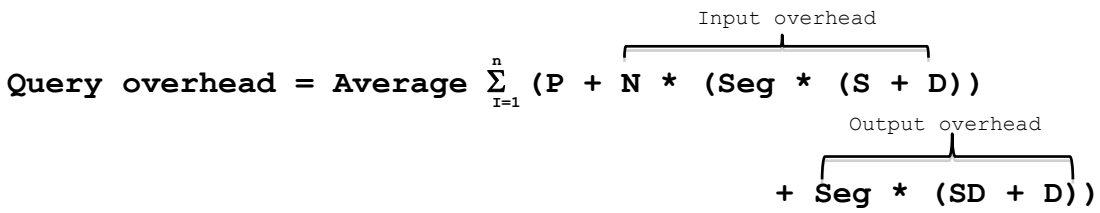


Figure 6.6: Query overhead of the annotation approach

Table 6.3 below lists the various mathematical symbols used in the analysis. The overhead of the annotation approach when recording WSN query provenance is the average of the total overhead of n number of query provenance overheads. The query overhead is the size of process provenance (P), and the input and the output overhead. The input overhead includes: the number of segments in the WSN query time (Seg) multiplication the size of each input stream (S) and its related dependency (D) for all sensors involved in the query (N), while the number of segments in the WSN query (Seg) multiplying the size of the derived stream (SD) by its dependency (D) is considered as an output overhead.

For our proposed solution, the provenance of all base streams and the derived stream will be collected only once in the WSN query lifetime. The stream segment is not in fixed time – it could be one second or one minute, either more or less, depending on the WSN query duration conditions. For the dependency, the provenance is recorded with each stream. With the extended recording policy there is a significant saving in per WSN query provenance, as shown below. The overhead of the query provenance in our proposed solution is the average of the total size of only the query concerned. The query overhead is still the size of process provenance (P), and the input and the output overhead. However the input overhead is only the size of the input stream (S) and its dependency relationship (D) for each of the sensors (N), since there is one segment for each query. For the same reason, the output overhead is the size of the derived stream (SD) and its dependency size (D):

$$\text{Query provenance overhead} = \text{Average} \sum_{I=1}^{n} (\text{Interest } (P + \overbrace{N * (S + D)}^{\text{Input overhead}} + \overbrace{SD + D}^{\text{Output overhead}}))$$

Figure 6.7: Query provenance overhead of the proposed language

Symbol	Meaning
N	No. of sensors
T	Time period
P	Size of process provenance
S	Size of stream provenance
SD	Size of output stream
D	Size of dependency provenance
E	Size of event provenance
Seg	No. of stream segments in T

Table 6.3: Mathematical symbols for provenance analysis

In order to record the changes in the monitoring environment, an event of interest in a stream is recorded when it occurs. So, the size of change events associated with a stream depends on the number of times it happened. Therefore, the size of stream event overhead is the average of the total size of all event provenance size (E) of all sensors (N) as shown:

$$\text{Overhead of stream event} = \text{Average} \sum_{I=1}^{n} (E * N)$$

Figure 6.8: Stream event overhead

The following chart represents the data in the provenance record size.

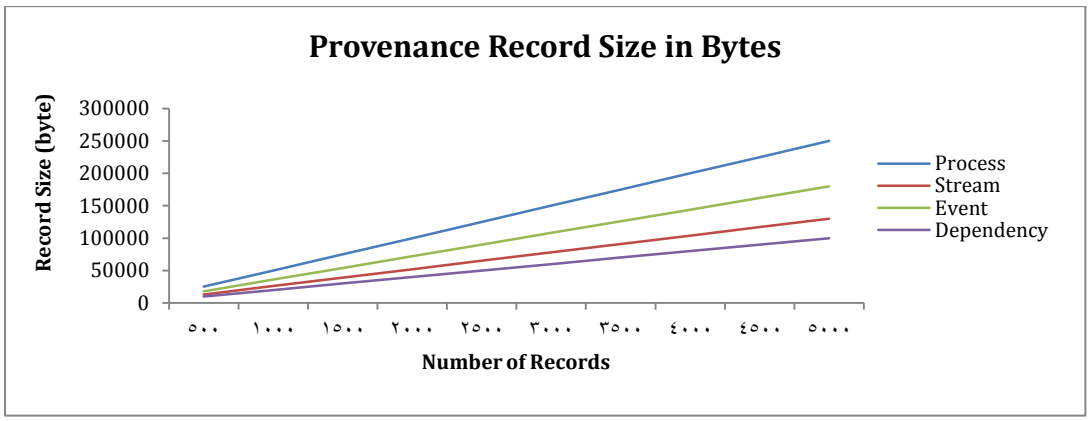


Chart 6.9: Provenance record size in Bytes

In Chart 6.6, it can be clearly seen that the size of the provenance information of the process records has the highest number of bytes. However, the process information will only be recorded for each WSN query and any operation performed on the data if it is specified in the policy. This indicates that 250KB of storage is required for 5,000 records of process provenance. The dependency information is recorded only when an operation is performed on the data and does not require many bytes: 1MB is enough for 50,000 dependency records.

The stream record has been inserted for each sensor in the WSN query and an output for each operation, therefore the number of stream records is increased by the increase in the number of sensors: 5,000 records of stream information requires only 130KB.

The number of event records depends on the number of changes accruing in the stream or the process during the WSN query, which cannot be predicted. However, 1MB is the size that fits for 28,000 records, and that is rational enough.

6.4 Summary

This chapter described the experimental setup for evaluating the proposed model. The first set of experiments was for evaluating the collecting model by examining the collection overhead and the performance scalability. The collection overhead was measured by evaluating the overhead of recording WSN query provenance and event provenance. The findings demonstrated a reasonable

overhead on the system functionality. The system performance scalability was examined by measuring the time required to record the provenance with an increased number of sensors and information to collect. The experimental result showed a linear increase. The second set of experiments was for evaluating the provenance query response time, which was examined with the increased number of provenance stores and with the increase in the result size. The result showed a linear increase in the response time with the increase in the provenance store, and the total time was still acceptable. The increase in the size of the result had little impact on the response time; however, a higher impact occurred when it required many interactions with many tables in the database. Finally, the provenance storage overhead was evaluated by comparing the storage size required in our proposed model with the annotation model, and showed a significant saving.

CHAPTER 7 CONCLUSION

This chapter presents a summary of the overall work, followed by a summary of its contribution. At the end, it discusses future work and concludes with closing remarks.

7.1 Summary of the work and its contribution

Pervasive computing is a model of human-computer interaction where information processing integrates into everyday activities. It is a rapidly developing area and has many potential applications from domestic ubiquitous computing to environmental monitoring and intelligent spaces. Most of these applications involve responding to real time events in the environment as well as the need sometimes for decision-making, and require provenance recording in order to justify any action and draw an accurate conclusion. However, pervasive applications present challenges when integrated with provenance support. Provenance is well studied in the field of scientific workflow and database, but is still an exploratory field in pervasive computing.

This work discusses the need for provenance in such an application and introduces a model for supporting provenance by making two contributions. The first contribution is in defining a provenance policy language for recording the provenance information. The recording policy specifies the different kinds of

information that need to be recorded with the ability to identify and address challenges that are unique to such a system, which is brought about by the dynamic nature and high rate of data streams. The proposed collection model, according to the provenance recording policy language, captures and stores provenance data based on an event alert, which is considered as a dataset for provenance collecting instead of a data element. The data model identifies four main units: streams, processes, their relationship to one another, and changes that occur in the stream and the process, which are used to maintain the construct of provenance information. This solution has been mapped to the Open Provenance Model structure in order to generate a compliant provenance structure and allow for its interoperability with other systems.

The second contribution is in introducing a fine-grained access control over the details of provenance information by defining an access control policy. It is a set of rules (authorization requirement) that specifies who is allowed to perform which action and so avoids the problem of storing multiple copies of provenance records depending on the principle of authority. The proposed access control model is mapped to RBAC (Role Based Access Control), which supports provenance specific requirements such as supporting both fine-grained policies and personal preferences. The basic concept of RBAC is that users are assigned to roles and roles are assigned to permissions. Roles are created for job functions or job titles regarding their authority, and permission is an approval of a particular access to a data object. The standardized RBAC specification language defines three policy elements: permission, restriction and obligation. These policies define rules by connecting a set of subjects (users) with a set of targets (data) and specifying the conditions of the rule. In order to provide a granularity level of access, permission can be defined as access to a field in a table tuple.

The model is validated by implementation in a use case of an energy monitoring system. Two examples from the system lifecycle have been discussed in detail, which are debugging on deployment and auditing for correct operation. The two examples show how the provenance information which needs to be

recorded for such functionality is specified in the collection language policies. Accordingly, the provenance graph structure has been developed. The proposed access control policy has been applied on the two use case scenario in order to illustrate how roles should be allocated for access to the provenance information and how permission is allowed or denied. In order to evaluate the proposed solution, a simple sensor simulator and a simulated application have been built. Provenance collection overhead and provenance query performance were quantified by experimental evaluation, while the storage overhead has been evaluated by a numerical analysis.

7.2 Future work

As provenance management is advancing rapidly, several directions can be potentially extended to this work in order to increase its reliability and usage in the future. The following are some of the possible extensions that can be added:

- Sensornets are becoming widely deployed and sharing sensor data online across multiple parties has become much more common. The process of transforming sensor data online is called republishing, and can involve a variety of processes and multiple users (Park 2008). The provenance, in this case, is defined as a record of actions taken on particular sensor data over its transformation life cycle. Each use of data online may generate a provenance record. In general, a provenance record may include the identity of the principle, a log action (e.g., aggregation, filtering), a description of the environment when the action was performed (such as the time and software), and confidentiality information. Keeping track of the provenance chain of time-ordered records by addressing the special requirement of sensor data streams and cross-domain demand could create an interesting challenge to be tackled.
- Data is increasingly shared across organizations; it is therefore essential to share provenance information along with the data. OPM as a standard provenance data model can be a starter for collaborative provenance information on a large scale, which would enable a combination of provenance resources to be exchanged.

- The problem of access control for provenance will become more complex as data may cross multiple domain boundaries. An access control policy requires a language that combines different access controls from different sources such as organization high-level security policies, the policies of the different involved parties, and privacy laws and regulations. The aggregation of authorization decisions from different policies with different purposes could be a new area to explore.

7.3 Closing remarks

This work has focused on defining the language for collecting and securing provenance information, and finding efficient approaches for storing and representing it. The provenance component service can be extended to a larger context management service by providing a method to query this information effectively For example, using a method for queries time to be largely independent from the total provenance size (Kementsietsidis et al. 2009). Another service could be to provide a usage pattern for this information in the application domain. Furthermore, it can be extended by considering the implication of data collected from multiple administration domains instead of a centralized approach. This could involve efficiently federating the collecting, storage and retrieving of provenance information across these domains.

REFERENCES

Aberer, K., Hauswirth, M. and Salehi, A. (2006) "A Middleware for Fast and Flexible Sensor Network Deployment". *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)*, pp. 1199-1202. ACM Press, New York.

Agrawal, D., Calo, S. B., Giles, J., Lee, K.-W. and Verma, D. C. (2005) "Policy management for networked systems and applications," *in IFIP/IEEE Symp. on Integrated Network Management, Nice, France*, pp. 455–468.

Barkley, J. (1997) "Comparing Simple Role Based Access Control Models and Access Control Lists". *Proceedings of the Second ACM Workshop on Role-based Access Control*. Fairfax, Virginia, United States, ACM.

Bauer, A., Gore, R. and Tiu, A. (2009) "A First-Order Policy Language for History-Based Transaction Monitoring", in Leucker, M. and Morgan, C. (eds), ICTAC, *Lecture Notes in Computer Science*, vol. 5684, pp. 96-111.

Bemana, A. (2012) "optimized query processing for wireless sensor networks". *International Journal of Science and technology,* vol. 1(2): pp. 67–71.

Bernstein, P. A. and Bergstraesser, T. (1999) "Meta-Data Support for Data Transformations Using Microsoft Repository," in *IEEE Data Engineering Bulletin*, vol. 22, pp. 9-14.

Bhagwat, D., Chiticariu, L.,Tan, W. C. and Vijayvargiya, G. (2004) "An Annotation Management System for Relational Databases," *in VLDB*, pp. 900-911.

Biton, O., Cohen-Boulakia, S., Davidson, S. and Hara, C. (2008) "Querying and Managing Provenance through User Views in Scientific Workflows," in *Proceedings of ICDE* (to be published).

Black, K. (2004). *Business Statistics for Contemporary Decision Making.* Wiley, India.

Blount, M., et al. (2007) "Century: Automated Aspects of Patient Care", in *Embedded and Real-Time Computing Systems and Applications.* RTCSA, 13th IEEE International Conference.

Blue Line Innovations Inc. (2010)
Available at: http://www.bluelineinnovations.com. (Accessed Dec 2010).

Bose, R. and Frew, J. (2005) "Lineage retrieval for scientific data processing: a survey," in *ACM Computer Survey*, vol. 37. New York, NY, USA, pp. 1-28.

Braun, U., Shinnar, A. and Seltzer, M. (2008) "Securing Provenance". *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security (HotSec)*.

Brase, J. (2004) "Using Digital Library Techniques - Registration of Scientific Primary Data," in *Lecture Notes in Computer Science*, vol. 3232, pp. 488-494.
Buneman, P., Khanna, S., Tan, W. C. (2000) "Data Provenance: Some Basic Issues", in *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*.

Buneman, P., S. Khanna, and W.C. Tan (2001) "Why and Where: A Characterization of Data Provenance", in *Proceedings of the 8th International Conference on Database Theory.* Springer-Verlag, pp. 316-330.

Buneman, P. and W.-C. Tan (2007) "Provenance in databases". *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data 2007. ACM*, Beijing, China pp. 1171-1173. Available at: http://doi.acm.org/10.1145/1247480.1247646 (Accessed 3 May 2008).

Cameron, G. (2003) *Provenance and Pragmatics*, Workshop on Data Provenance and Annotation, Edinburgh.

Chalmers, D. (2007) "Classification and Use of Context"*. Pervasive Computing Course Notes.*

Chen, L., Tan, V., Xu, F., Biller, A., Groth, P., Miles, S., Ibbotson, J., Luck, M. and Moreau, L. (2005) "A Proof of Concept: Provenance in a Service Oriented Architecture"*,* in *Proceedings of the Fourth All Hands Meeting (AHM)*, September 2005.

Cheney, J. (2011) "A Formal Framework for Provenance Security", in *The 24th IEEE Computer Security Foundations Symposium*.

Chebotko, A., Chang, S. Lu, S., Fotouhi, F. and Yang, P. (2008) "Scientific Workflow Provenance Querying with Security Views". *WAIM*, pp. 349-356.

Chong, S. K., Krishnaswamy, S. and Loke, W. (2005) "A Context-Aware Approach to Conserving Energy in Wireless Sensor Networks". *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society.*

Clarke, D. G. and Clark, D. M. (1995) "Lineage," in Guptill, S.C. and Morrison, J.L. (eds.), *Elements of Spatial Data Quality*. Oxford, Elsevier Science.

Collins-Sussman, B., Fitzpatrick, B. W. and Pilato, C. M. (2004) "Version Control with Subversion". *O'Reilly Media*, first edition.

Concurrent Versioning System. http://www.nongnu.org/cvs/. Access date March 2012.

Cui, Y. and Widom, J. (2000a) "Practical Lineage Tracing in Data Warehouses", in *ICDE*, pp. 367-378.

Cui, Y., Widom, J. and Wiener, J. (2000b) "Tracing the lineage of view data in a warehousing environment". *ACM Transactions on Database Systems*, vol. 25(2): pp. 179–227.

Davidson, S., Ludäscher, B., McPhillips, T. and Freire, J. (2007) *Provenance in Scientific Workflow Systems.* Available at: http://sites.computer.org/debull/A07dec/susan.pdf (Accessed 20 April 2008).

Davidson, S. B. and Freire, J. (2008) "Provenance and Scientific Workflows: Challenges and Opportunities". *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. Vancouver, Canada, ACM.
Demers, A., Gehrke, J., Panda, B., Riedewald, M., Sharma, V. and White, W. (2007) "Cayuga: A general purpose event monitoring system",in *Proc. CIDR*, pp 412-422.

Dey, A.K. (2001) *Understanding and Using Context.* Available at: http://dx.doi.org/10.1007/s007790170019 Personal Ubiquitous Computing, **5** (1): p. 4-7

DIY Kyoto (2010)
Available at: http://www.diykyoto.com/uk/wattson/about (Accessed Dec 2010).

Ferraiolo, D, Barkley, J. and Kuhn, D.R. (1999) "A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet." *ACM Trans. Inf. Syst. Secur.* 2(1): 34-64.

Freire, J. Koop, D. Santos, E. and Silva, C. (2008) "Provenance for Computational Tasks: A Survey". *Computing in Science & Engineering*, 10(3) pp. 20-30.

Frew, J., Metzger, D. and Slaughter, P. (2008) "Automatic Capture and Reconstruction of Computational Provenance," *Concurrency and Computation: Practice and Experience,* vol. 20, no. 5, pp. 485–496.

Gaber, M. (2007) "Data Stream Processing in Sensor Network", in Gama, J. and Gaber, M.M. (eds.) *Learning from Data Streams: Processing Techniques in Sensor Networks*, Springer Verlag, pp. 41-48.

Gehrke, J. and Madden, S. (2004) "Query Processing in Sensor Networks". *Pervasive Computing, IEEE*, 3(1) pp. 46-55.

Glavic, B., Miller, R., Alonso, G. (2010) "Using SQL for Efficient Generation and Querying of Provenance Information".

Golab, L. and Ozsu, M. T. (2003) “Issues in data stream management,” *ACMSIGMOD Record*, vol. 32, pp. 5–14.

Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L. and Oinn, T. (2003) "Provenance of e-Science Experiments – Experience from Bioinformatics," in *Proceedings of the UK OST e-Science second All Hands Meeting*.

Groth, P., Miles, S. and Moreau, L. (2005) "PReServ: Provenance Recording for Services". *Proceedings of the UK OST e-Science second All Hands Meeting* (AHM'05), Nottingham, UK.

Groth, P., Jiang, S., Miles, S., Munroe, S., Tan, V., Tsasakou, S. and Moreau, L. (2006) *An Architecture for Provenance Systems.* Available at: http://eprints.ecs.soton.ac.uk/12023/. Accessed 12 May 2008.

Groth, P., Tan, V., Munroe, S., Jiang, S., Miles, S. and Moreau, L. (2006) *Process Documentation Recording Protocol*. Technical report, University of Southampton.

Gude, R. and Oster, M. (2007) “Provenance-CSL: A provenance client side library”. *Technical report, Fachhochschule Bonn-Rhein-Sieg, Fachbereich Informatik.*

Guitton, A., Skordylis, A. and Trigoni, N. (2007) "Utilizing Correlations to Compress Time-Series in Traffic Monitoring Sensor Network", in *IEEE Wireless Communication and Networking Conference (WCNC)*.

Gyllstrom, D., Wu, E., Chae, H., Diao, Y., Stahlberg, P. and Anderson, G. (2007) "SASE: Complex Event Processing over Streams (Demo)", in *CIDR Conference*, Asilomar, CA.

Harris, E., Krishna, R., Chalmers, D., Fitzpatrick, G., Stringer, M. (2007) "From Development in the Laboratory to Deployment in the Home: Trouble and strife with sensor networks", in *Proceedings of the 32nd IEEE Conference on Local Computer Networks*. IEEE Computer Society.

Hasan, R., Sion, R., Winslett, M. (2007) "Introducing Secure Provenance: Problems and Challenges". *Proceedings of the 2007 ACM Workshop on Storage Security and Survivability*. Alexandria, Virginia, USA. ACM.

Hasan, R., Sion, R., Winslett, M. (2009) "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance". *Proceedings of the 7th Conference on File and Storage Technologies*. San Francisco, California. USENIX Association.

Henricksen, K., Indulska, J. and Rakotonirainy A. (2002) "Modeling Context Information in Pervasive Computing Systems", in *Proceedings of the Conference on Pervasive Computing,* Zurich.

Hinton, H. and Stewart Lee, E. (1994) "The compatibility of policies", in *CCS '94: Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 258-269, New York, NY, USA. ACM

Hiramatsu, K., Hattori, T, Yamada, T and Okadome T, (2005) "Finding Small Changes Using Sensor Networks", *Proceedings of Ubicomp 2005 Workshop on Smart Object Systems*, 37-44, Tokyo, Japan.

Holland, D. A., Braun, U., Maclean, D., Muniswamy-Reddy, K.-K., Seltzer, M. I. (2008) "Choosing a Data Model and Query Language for Provenance", in *Int. Provenance and Annotation Workshop*.

Horré, W., Matthys, N., Michiels, S., Joosen, W. and Verbaeten, P. (2007) *A Survey of Middle-Ware for Wireless Sensor Networks*. Technical Report CW 498, Department of Computer Science, K.U.Leuven, Belgium.

Howe, B. and Maier, D. (2002) "Modeling Data Product Generation," *in Workshop on Data Derivation and Provenance*, Chicago.

Imran, M. and K. A. Hummel (2008) "On Using Provenance Data to Increase the Reliability of Ubiquitous Computing Environments". *Proceedings of the 10th International Conference on Information Integration and Web-based Applications \& Services*. Linz, Austria, ACM.

Jiang, X., Dawson-Haggerty, S., Dutta, P. and Culler, D. (2009) "Design and Implementation of a High-Fidelity AC Metering Network", in *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks*.

Kappler, C. and Riegel, G. (2004) "A Real-World, Simple Wireless Sensor Network for Monitoring Electrical Energy Consumption".*Springer*, Berlin / Heidelberg, vol. 2920, pp 339-352

The Kepler Project. Available at: https://kepler-project.org/ (Accessed 12 March 2012).

Kementsietsidis, A. and Wang, M. (2009) "Provenance query evaluation: what's so special about it? ", in *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09).*

Kim, C.H., Park, K., Fu, J. and Elmasri, R. (2005) "Architectures for Streaming Data Processing in Sensor Networks in Computer Systems and Applications". *The 3rd ACS/IEEE International Conference*.

Lange, R-J. (2010) *Provenance aware sensor networks for real-time data analysis*. Available at http://essay.utwente.nl/59473/1/scriptie_R_de_Lange.pdf (Accessed 12 May 2010).

Lanter, D. P. (1990) "Lineage in GIS: The Problem and a Solution," in *Technical Report*: National Center for Geographic Information and Analysis.

Ledlie, J., Ng, C., Holland, D., Muniswamy-Reddy, K., Braun, U. and Seltzer, M. (2005) "Provenance-Aware Sensor Data Storage". *Proceedings of the 21st International Conference on Data Engineering Workshops*, IEEE Computer Society.

Li, S., Balfe, S. , Zhou, J. , Chen , K. (2008) "Enforcing trust in pervasive computing". *International Journal of Systems Engineering*. vol. 1, pp. 96-110.

Lim, H., Moon, Y. and Bertino, E. (2009) "Research Issues in Data Provenance for Streaming Environments". *Proceedings of the 2009 ACM SPRINGL*, November 3, 2009, Seattle, WA, USA, pp. 58 - 62.

Liu, Y., Vijayakumar, N. N., Plale, B. (2006) "Stream Processing in Data-Driven Computational Science", in *Grid Computing, 7th IEEE/ACM International Conference*.

Loo, B., Condie, T., Garofalakis, M., Gay, D., Hellerstein, J., Maniatis, P., Ramakrishnan, R., Roscoe, T. and Stoica, I. (2006) "Declarative Networking: Language, Execution and Optimization", in *ACM SIGMOD*.

Madden, S., Szewczyk, R., Franklin, M. and Culler, D. (2002) "Supporting aggregate queries over ad-hoc wireless sensor networks". *In Workshop on Mobile Computing and Systems Applications.*

Madden, S., Hellerstein, J. and Hong, W. (2003) *TinyDB: In-Network Query Processing in TinyOS*. Available at: http://telegraph.cs.berkeley.edu/tinydb/doc/index.html (Accessed 20 Nov 2010).

Misra, A., Blount, M., Kementsietsidis, A., Sow, D. and Wang, M. (2008) "Advances and Challenges for Scalable Provenance in Stream Processing Systems". *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop*, IPAW 2008, Salt Lake City, UT, USA, June 17-18. *Revised Selected Papers*, Springer-Verlag, pp. 253-265.

Moreau L, Groth P, Miles S, Vazquez J, Ibbotson J, Jiang S, Munroe S, Rana O, Schreiber A, Tan V, Varga L. (2007) "The provenance of electronic data". *Communications of the ACM*. 51(4): pp. 52–58

Moreau, L., Chapman, S., Schreiber, A., Hempel, R., Rana, O., Varga, L., Cortes, U. and Willmott, S. (2004) *Provenance-based Trust for Grid Computing – Position Paper.* Available at: http://eprints.soton.ac.uk/262571/ (Accessed 2 Sep 2009)

Moreau, L., Ludäscher, B., Altintas, I., Barga, R. S., Bowers, S., Callahan, S., Chin Jr., G., Clifford, B., Cohen, S., Cohen-Boulakia, S., Davidson, S., Deelman, E., Digiampietri, L., Foster, I., Freire, J., Frew, J., Futrelle, J., Gibson, T., Gil, Y., Goble, C., Golbeck, J., Groth, P., Holland, D., Jiang, S., Kim, J., Koop, D., Krenek, A., McPhillips, T., Mehta, G., Miles, S., Metzger, D., Munroe, S., Myers, J., Plale, B., Podhorszki, N.,

Moreau, L. (ed.), Plale, B., Miles, S., Goble, C., Missier, P., Barga, R., Simmhan, Y., Futrelle, J., McGrath, R., Myers, J., Paulson, P., Bowers, S., Ludaescher, B., Kwasnikowska, N., Van den Bussche, J., Ellkvist, T., Freire, J., Groth, P. (2008) *The open provenance model (v1.01*). Technical report, University of Southampton (July 2008), Available at: http://eprints.soton.ac.uk/266148/ (Accessed 12 Oct 2010).

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., den Bussche, J.V. (2010) "The open provenance model core specification (v1.1)". *Future Generation Computer Systems*. In press, accepted manuscript (2010)

Muniswamy-Reddy, K. K., Holland D. A., Braun, B., Seltzer, M. (2006) "Provenance-Aware Storage Systems", in *Processing of the 2006 USENLX Annual Technical Conference*.

MySQL (2011) Available at : http://www.mysql.com/ (Accessed 1 Sep 2011)

Oxford Dictionaries (2009) Available at: http://oxforddictionaries.com/definition/provenance (Accessed 30 June 2009).

Park, U., and Heiddemann, J. (2008) "Provenance in Sensornet Republishing" *Springer* Berlin/Heidelberg, vol. 5272, pp 280-292.

Parr, T. (2007) *The Definitive ANTLR Reference: Building Domain-Specific Languages.* The Prgamatic Programmers, USA.

PASS (not dated) Available at: www.eecs.harvard.edu/syrah/pass (Accessed Sep 2009).

Phidgets (products for USB sensing and control). Available at: http://www.phidgets.com/products.php?category=8&product_id=3503 (Accessed Dec 2010).

Provenance Challenge. (2011) *The Fourth and Last Provenance Challenge*. Available at: http://twiki.ipaw.info/bin/view/Challenge/FourthProvenanceChallenge (Accessed Oct 2011).

P3 International (2010) Available at: http://tinyurl.com/436sw (Accessed Sep 2011).

Ratnakar, V., Santos, E., Scheidegger, C., Schuchardt, K., Seltzer, M., Simmhan, Y. L., Silva, C., Slaughter, P., Stephan, E., Stevens, R., Turi, D., Vo, H., Wilde, M., Zhao, J., Zhao, Y. (2006) *The First Provenance Challenge. Concurrency and Computation: Practice and Experience*. Published online. DOI 10.1002/cpe.1233.

Ray, S., Starobinski, D., Trachtenberg, A., Ungrangsi, R. (2004) "Robust location detection with sensor networks". *Selected Areas in Communications, IEEE Journal on*, 2004, vol. 22(6), pp. 1016-1025.

Ringelstein, C. and Staab. S. (2010a) "PAPEL: A Language and Model for Provenance-Aware Policy Definition and Execution", in *BPM 2010 – International Conference on Business Process Management*.

Ringelstein, C. and Staab, S. (2010b) "Provenance-Aware Policy Definition and Execution", in *BPM 2010 – International Conference on Business Process Management*.

Ringwald, M., Romer, K., and Vialetti, A. (2006) "SNIF: Sensor Network Inspection Framework" *Technical Report 535, Department of Computer Science*, ETH Zurich.

Romeu, J. L. (1999) "Data Quality and Pedigree," in *Material Ease*.

Roychowdhury, A., Falchuk, B. and Misra, A. (2010) "MediAlly: A Provenance-Aware Remote Health Monitoring Middleware", *8th IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

Sandhu, R., Coyne, E., Feinstein, H., Youman, C. (1996) "Role-Based Access Control Models." *IEEE Computer*, vol. 29(2), pp. 38-47.

Satyanarayanan, M. (2001) "Pervasive Computing: Vision and Challenges". *Personal Communication*, IEEE, vol. 8(4): pp. 10-17.

Schaad, A., Moffett, J. and Jacob, J. (2001) "The Role-Based Access Control System of a European Bank: a Case Study and Discussion". *Proceedings of the sixth ACM Symposium on access control models and technologies*. Chantilly, Virginia, United States, ACM.

Scheidegger, C., Koop, D., Vo, H., Freire, J. and Silva, C. (2007) "Querying and creating visualizations by analogy". *IEEE Transactions on Visualization and Computer Graphics,* vol. 13, no. 6, pp. 1560-1567.

Seltzer, M., Muniswamy-Reddy, K.-K., Holland, D.A., Braun, U. and Ledlie, J. (2005) "Provenance-Aware Storage Systems". *Technical report, Harvard University Computer Science Technical Report TR-18-05.*

Sheng, Q.Z., Nambiar, U., Sheth, A., Srivastava, B.**,** Maamr, Z. and Elnaffar, S. (2008) "WS3 International Workshop on Context-Enabled Source and Service Selection, Integration and Adaptation" *(CSSSIA 2008),* in *Proceedings of the 17th international conference on World Wide Web.* ACM: Beijing, China pp. 1263-1264

Simmhan, Y.L., Plale, B. and Gannon, D. (2005a) *A Survey of Data Provenance Techniques.* Available at:

http://www.cs.indiana.edu/l/www/ftp/techreports/TR618.pdf (Accessed March 2008).

Simmhan, Y.L., Plale, B. and Gannon, D. (2005b) "A Survey of Data Provenance in e-Science". *SIGMOD Rec.* vol. 34 (3), pp. 31-36

Simmhan, Y.L., Plale, B. and Gannon, D. (2006a) "A Framework for Collecting Provenance in Data-Centric Scientific Workflows", in *International Conference on Web Services*.

Simmhan, Y.L. , Plale, B. , Gannon, D. and Marru, S. (2006b) "Performance Evaluation of the Karma Provenance Framework for Scientific Workflows ". In L. Moreau and I. T. Foster (eds), *International Provenance and Annotation Workshop (IPAW)*, Chicago, IL, vol. 4145 of *Lecture Notes in Computer Science*, pp. 222–236.

Simmhan, Y.L., Plale, B. and Gannon, D. (2008) "Karma2: Provenance Management for Data Driven Workflows", to be published in *International Journal of Web Services Research*, vol. 5, no. 1.

Sivrikaya, F. and Yener, B. (2004) "Time synchronization in sensor networks: a survey", *IEEE Network* 18 (4) pp. 45–50.

SmartLabs Inc.(2010) Available at: http://tinyurl.com/34d23n8 (Accessed Sep 2011)

Szomszor, M. and Moreau, L. (2003) "Recording and Reasoning over Data Provenance in Web and Grid Services". *Springer* Berlin / Heidelberg, 2003, vol. 2888, pp. 603-620.

Tan, V., Groth, P., Miles, S., Jiang, S., Munroe, S., Tsasakou, S., and Moreau, L. (2006) "Security Issues in a SOA-Based Provenance System", in *Third International Provenance and Annotation Workshop*, Springer.

Tan, W.C. (2007) "Provenance in Databases: Past, Current, and Future", *IEEE Data Eng*. Bull. 30(4), pp.3-12.

The Taverna Project. Available at: http://taverna.sourceforge.net. (Accessed 12 March 2012).

Tolle, G., Polastre J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., Hong, W. (2005) "A macroscope in the redwoods". *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. San Diego, California, USA, ACM.

Twidle, K., Dulay, N., Lupu, E. and Sloman, M. (2009) "Ponder2: A policy system for autonomous pervasive environments," *Autonomic and Autonomous Systems, ICAS '09. Fifth International Conference*, pp. 330–335.

Vassiliadis, P., Bouzeghoub, M. and Quix, C. (1999) "Towards Quality-Oriented Data Warehouse Usage and Evolution," *LNCS* 1626.

Vijayakumar, N. and Plale, B. (2006) "Towards Low Overhead Provenance Tracking in Near Real-Time Stream Filtering", *Springer* Berlin/Heidelberg vol. 4145, pp. 46-54.

Vijayakumar, N.and Plale, B. (2007) "Tracking Stream Provenance in Complex Event Processing Systems for Workflow-Driven Computing". *Second International Workshop on Event-driven Architecture, Processing and Systems (EDA-PS'07*). Vienna, Austria.

The VisTrails Project. Available at: http://www.vistrails.org/index.php/Main_Page (Accessed 12 March 2012).

Wang, M., Blount, M., Davis, J., Misra, A., Sow, D. (2007) "A Time-and-Value Centric Provenance Model and Architecture for Medical Event Streams". *Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments*. San Juan, Puerto Rico, ACM.

Weiss, M., Guinard, D.(2010) "Increasing Energy Awareness Through Web-enabled Power Outlets", in *Proceedings. of MUM* 2010.

Weiss, M., Mattern, F., Graml, T., Staake, T. and Fleisch, E. (2009) "Handy Feedback: Connecting Smart Meters with Mobile Phones, in *Proceedings of MUM (The 8th International Conference on Mobile and Ubiquitous Multimedia)*.

Welsh, M., Myung, D., Gaynor, M. and Moulton, S. (2003) "Resuscitation Monitoring with a Wireless Sensor Network", in *Circulation: Journal of the American Heart Association*.

Widom, J. (2005) "Trio: A System for Integrated Management of Data, Accuracy, and Lineage", in *CIDR*, pp. 262-276.

Zhao, J., Goble, C. A., Stevens, R. and Bechhofer, S. (2004) "Semantically Linking and Browsing Provenance Logs for E- science," in *ICSNW*, pp. 158–176

APPENDIX 1

```
grammar Event;

tokens {
EXTENDS = 'extends';
IMPORT = 'import';
EVENT =  'event';
LONGTYPE =   'Long';
STRINGTYPE = 'String';
DOUBLETYPE= 'Double';
NULL = 'null';
EVENT = 'event';
AND = 'and';
OR = 'or';
COMMA =  ',';
BECOMES =    '=>';
LCURL = '{';
RCURL = '}';
}

@header {
package uk.ac.susx.inf.foss.provenance.event.parser;
import uk.ac.susx.inf.foss.provenance.event.Operation;
import uk.ac.susx.inf.foss.provenance.event.Event;
import java.util.List;
import java.util.ArrayList;
}

@lexer::header {
package uk.ac.susx.inf.foss.provenance.event.parser;
}

@members {
ArrayList<Condition> conditionList = new
ArrayList<Condition>();
ArrayList<AttributeAssignment> assignmentList = new
ArrayList<AttributeAssignment>();
}

definition   returns [DFACollector dfa]
:
importList*
type=event   {$dfa = new DFACollector
($type.evType,conditionList,assignmentList);}
;

event    returns [String evType]
:
EVENT type=ID {Event.legalEvents.put($type.text,null);$evType =
$type.text;}
(EXTENDS supertype=ID)?
```

```
LCURL
propertyList
RCURL
dfa?
;

propertyList : property*
;

property : (
LONGTYPE
| STRINGTYPE
| DOUBLETYPE
) ID ';'
;

dfa : conditionList BECOMES attributeAssignments ';'
;

conditionList    : cond=condition {conditionList.add(cond);}
(COMMA cond1=condition    {conditionList.add(cond1);}
)*
;

condition returns [Condition cond]
:
eventDefn  {cond = new ConditionEvent($eventDefn.ev);}
| attributeTest  {cond = new ConditionTest
($attributeTest.attr);}
;

eventDefn    returns [EventMatch ev]
: type=ID '[' temporalPattern ']' {ev = new EventMatch
($type.text,$temporalPattern.temp);}
;

attributeTest    returns [AttributeTest attr]
: arg1=eventAttr
op = ( '>' | '<' | '==' | '!=')   {$attr = new AttributeTest
($arg1.attr,$op.text);}
(arg2=eventAttr  {$attr.setArg2($arg2.attr);}
| STRING {$attr.setArg2($STRING.text);}
| INT    {$attr.setArg2(Long.parseLong($INT.text));}
| FLOAT  {$attr.setArg2(Double.parseDouble ($FLOAT.text));}
| NULL   {$attr.setArg2((String)null);}
)
;

eventAttr    returns [EventAttribute attr]
: ev=eventDefn '.' attribute=ID {attr = new EventAttribute
($ev.ev,$attribute.text);}
;

temporalPattern returns [Temporal temp]
:
```

```
| 'n'    {temp = new TemporalRelative(01);}
| 'n' '-' INT {temp = new TemporalRelative(Long.parseLong
($INT.text));}
;

attributeAssignments
:
attr0=attributeAssignment {assignmentList.add($attr0.attr);}
(','
attr1=attributeAssignment {assignmentList.add($attr1.attr);}
)*
;

attributeAssignment returns[AttributeAssignment attr]
:
type=ID
'.'
field=ID
'='
(
expr{$attr = new AttributeAssignment($type.text,
$field.text,$expr.expr);}
|STRING  {$attr = new AttributeAssignment($type.text,
$field.text,new ExpressionString($STRING.text));}
|INT{$attr = new AttributeAssignment($type.text,
$field.text,new ExpressionLong(Long.parseLong($INT.text)));}
|FLOAT   {$attr = new AttributeAssignment($type.text,
$field.text,new
ExpressionDouble(Double.parseDouble($FLOAT.text)));}
)
;

expr returns [Expression expr]
:
arg0=eventAttr {$expr = new ExpressionSingle($arg0.attr);}
|arg1=eventAttr op=( '+' | '-') arg2=eventAttr {$expr = new
ExpressionCompound($arg1.attr,$arg2.attr,$op.text);}
;

ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
;

INT : '0'..'9'+
;

FLOAT
:    ('0'..'9')+ '.' ('0'..'9')* EXPONENT?
|    '.' ('0'..'9')+ EXPONENT?
|    ('0'..'9')+ EXPONENT
;

COMMENT
:    '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
|    '/*' ( options {greedy=false;} : . )* '*/'
```

```
{$channel=HIDDEN;}
;

WS: (''
| '\t'

| '\r'
| '\n'
) {$channel=HIDDEN;}
;

STRING
: '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
{
String tmp = getText().substring(1, getText().length()-1);
setText(tmp);
}
;

fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;
fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
:   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
| UNICODE_ESC
| OCTAL_ESC
;

fragment
OCTAL_ESC
:   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
|   '\\' ('0'..'7') ('0'..'7')
|   '\\' ('0'..'7')
;
fragment
UNICODE_ESC
:   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
;
```

APPENDIX 2

```
grammar Policy;

tokens {
CAPTURE = 'capture';
POLICYTOKEN = 'policy';
TIMEID = '$time';
DURATION = 'duration';
STARTTIMEID = '$startTime';
WILDCARD = '*';
COMMA =  ',';
START =  'start';
END =    'end';
}

@header {
package uk.ac.susx.inf.foss.provenance.policy;
import uk.ac.susx.inf.foss.provenance.event.Event;
}

@lexer::header {
package uk.ac.susx.inf.foss.provenance.policy;
}

@members {
ArrayList<Expr> expressions = new ArrayList<Expr>();
ArrayList<Expr> startExpressions = new ArrayList<Expr>();
ArrayList<Expr> endExpressions = new ArrayList<Expr>();
ArrayList<String> events = new ArrayList<String>();
}

definition returns [Policy policy]
:
POLICYTOKEN name=ID
startConditions?
collectConditions
endConditions?

CAPTURE
events   {$policy = new Policy($name.text,startExpressions,
expressions, endExpressions, events);}
;

startConditions
:
START '{'
exp0 = expr {startExpressions.add($exp0.exp);}
(
COMMA
exp1 = expr {startExpressions.add($exp1.exp);}
)*
```

```
'}'
;

endConditions
:
END '{'
exp0 = expr {endExpressions.add($exp0.exp);}
(
COMMA
exp1 = expr {endExpressions.add($exp1.exp);}
)*
'}'
;

collectConditions
:
'{'
exp0 = expr {expressions.add($exp0.exp);}
(
COMMA
exp1 = expr {expressions.add($exp1.exp);}
)*
'}'
;

expr returns [Expr exp]
:
intExpr {$exp = $intExpr.exp;}
|
stringExpr {$exp = $stringExpr.exp;}
|
timeExpr {$exp = $timeExpr.exp;}
|
durationExpr {$exp = $durationExpr.exp;}
;

intExpr returns [ExprInteger exp]
:
attribute
op = ('>'|'<'|'=='|'!=')
val = INT {$exp = new ExprInteger($attribute.attr,
$op.text,Long.parseLong($val.text));}
;

durationExpr returns [ExprDuration exp]
:
DURATION '=' duration = INT
('minutes' {$exp = new ExprDuration(); $exp.setMinutes
(Integer.parseInt($duration.text));}
|'hours' {$exp = new ExprDuration(); $exp.setHours
(Integer.parseInt($duration.text));}
|'seconds' {$exp = new ExprDuration(); $exp.setSeconds
(Integer.parseInt($duration.text));}
)
;
```

```
stringExpr returns [ExprString exp]
:
attribute
op=('=='|'!=')
str=STRING {$exp = new ExprString($attribute.attr,$op.text,
$str.text);}
;

attribute returns [Attribute attr]
:
ev = (WILDCARD | ID) '.' att = ID {$attr = new
Attribute($ev.text, $att.text);}
;

timeExpr returns [ExprTime exp]
:
TIMEID
op=('>' | '<')
t = time
{$exp = new ExprTime($t.hour,$t.minute,$op.text);}
;

catch [PolicyException pe]{throw new RecognitionException();}
time returns [int hour, int minute]
:
hours = INT {$hour = Integer.parseInt($hours.text);}
':'
minutes = INT {$minute = Integer.parseInt($minutes.text);}
;

events :
ID+ {events.add($ID.text);}
;

ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
;

INT : '0'..'9'+
;

FLOAT
:    ('0'..'9')+ '.' ('0'..'9')* EXPONENT?
|    '.' ('0'..'9')+ EXPONENT?
|    ('0'..'9')+ EXPONENT
;

WS: (''
| '\t'
| '\r'
| '\n'
) {$channel=HIDDEN;}
;

STRING
```

```
: '"' ( ESC_SEQ | ~('\\'|'"') )* '"'
{
String tmp = getText().substring(1, getText().length()-1);
setText(tmp);
}
;

fragment
EXPONENT : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment
HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
ESC_SEQ
:   '\\' ('b'|'t'|'n'|'f'|'r'|'"'|'\''|'\\')
| UNICODE_ESC
| OCTAL_ESC
;

fragment
OCTAL_ESC
:   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
|   '\\' ('0'..'7') ('0'..'7')
|   '\\' ('0'..'7')
;

fragment
UNICODE_ESC
:   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
```