UNIVERSITY OF SOUTHAMPTON

A provenance-aware crowd-sourced system for monitoring CO_2 emissions

by

Pavlos Polianidis

A thesis submitted in partial fulfillment for the
degree of Master of Science

in the

Faculty of Physical and Applied Sciences
Electronics and Computer Science

Supervisor: Dr Luc Moreau

Examiner: Dr Leslie Carr

September 2012

"Most of the time we know how much things cost. Our financial sense of proportion allows us to make good choices Our carbon instinct needs to be just like the one we have for managing money"

Mike Berners-Lee

UNIVERSITY OF SOUTHAMPTON

Abstract

Faculty of Physical and Applied Sciences
Electronics and Computer Science

Master of Science

by Pavlos Polianidis

Human activities have brought severe damages to the environment, over the last century. We all have witnessed that for the past few decades, the anthropogenic environmental impacts have led to severe climate changes. It is therefore more than obvious for us to seriously consider our impact on the environment. The most widespread way to achieve that is to use the term Carbon Footprint for quantifying that impact. There is a vast amount of tools that help us track our carbon footprints. However, the majority of them fail to engage users to insert all the required data for calculating their individual carbon footprints. Furthermore, for some of the tools the calculations are made based on various kinds of estimation, which makes the computed values less accurate and misleading. Finally, according to our best knowledge, there is no a single carbon footprint calculator that allows users to verify and reproduce the computed values.

This report describes the design and implementation of a crowd-sourced web (and mobile) application for monitoring individual and group carbon emissions, caused by travelling activities. The application leverages the provenance technology to allow users to validate and repeat the computational processes.

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

For/Dedicated to/To my...

Chapter 1

Introduction

In January 2008, the Higher Educational Funding Council for England (HEFCE) announced that carbon dioxide emissions would be one of the primary criteria for allocating capital funding to the UK universities. Hence, universities have to have plans to reduce their CO2 emissions, through minimizing waste and changing travel habits. In particular, the University of Southampton[1] plans to reduce its CO2 emissions by 10.400 tonnes by 2020. Thus, a system for monitoring and tracking CO2 emissions is indispensable, so that effective measures can be taken.

This project involves the design, development and deployment of a provenance-aware crowd-sourced system, which will allow any member of an organization, such as Universities, to monitor his carbon emissions. Several tools already exist for that purpose. However, there are several limitations that hinder users from using them. The major one is that it is somewhat difficult to engage users to put much effort in recording their activities (e.g. travels). Some applications try to minimize or even eliminate that effort by applying some quirks. That comes though, at the expense of accurately computing carbon emissions. Other applications track users' geo locations, in order to make educated estimations of the transport means they use. However, this brings several privacy issues onto the table. Finally, a noticeable shortcoming is that there is no application that allows users to review the history of the figures that it computes. In other words, users cannot examine the derivation of the value for their carbon emissions. This feature is vital, since it would allow users make trust judgments about the results computes by the application. Furthermore, comparisons among different calculation methods could be conducted, by following the derivation history of the value in question.

The main objective of the project described in this report is to design and develop an online web application that will, to some extent, consider the aforementioned issues, with emphasis given to the provenance technology.

[1] http://www.southampton.ac.uk/carbonmanagement/our_plan/

To sum up, the project involves:

- The design and development of a web-based application which users will use to add the trip they make on daily basis. The application will compute individual or groups (within an organization) carbon footprints. A dedicated web page will summarize those computations and present them in a user-friendly manner (e.g. charts). Additionally, the provenance of those values would be possible to be viewed.
- The development of a mobile version of the application, so that users can use it while their travelling.
- The design, development and deployment of a server which will control the whole process of storing trip and computing the corresponding carbon emissions.
- The design and development of a provenance software component, which will carry all the provenance-related tasks.

We start in chapter 2 by performing a depth review of the literature found in the provenance and carbon footprint bibliography. We then describe a set of applications that are similar to our application; pros and cons for each are also provided. The chapter ends with a brief description of the tools and libraries that were used during this project.

In chapter 3 a discussion about the application's design phase is given. Various kinds of UML diagrams showing in detail the design of the application, are presented.

Chapter 4 describes the provenance technology in the context of our application. Benefits for embracing a provenance software component are outlined.

The implementation phase is described in chapter 5. This includes several screen shots, showing both the web and mobile application in action. Furthermore, we discuss the evaluation of the application via unit testing and present the results of the accessibility and usability tests. A critical assessment and future worked are highlighted at the end of the chapter.

Finally, a conclusion is given in chapter 6.

Chapter 2

Background

2.1 Introduction

In this chapter we will present the background research pertaining to the theory that underpins the development of the our web application. More specifically the theory from provenance and carbon footprint bibliography is presented. The chapter ends with a description of similar application and the tools that were utilized for the implementation of the application.

2.2 Provenance of Electronic Data

A significant factor when buying a piece of artwork is the knowledge of the derivation of that item, as well as, the chronology of the ownership. This has been extensively described with the term provenance and is important for various reasons, chief of which is the fact that it can help us trace the whole history of an object and thus determine the authenticity and establish its historical importance. Ultimately, one can use this information to determine the value of a work of art. The same concept has been applied to digital resources or digital information that is generated by computer applications[1]. More specifically, in computer science, we refer to provenance of a piece of data, as the process that led to that specific piece of data[2]. Moreover, any other data or hardware or user interaction that took part in the computational process, belong to the provenance of that piece of data. In general though, the description of such derivation may be represented in any form based on users' personal interest[3].

Figure 2.1 displays the tag cloud for the literature research presented in this section.

FIGURE 2.1: Provenance Tag Cloud

2.2.1 Provenance in Scientific Workflows and Curated Databases

Substantial work on provenance of data has been undertaken by the database, workflow[4][5] and e-science communities. In particular, the provenance of scientific results can provide proof about the correctness of a result and determine the amount of trust one can place on it[1][6]. Representative examples are workflows in the scientific domain. A workflow (e.g. myGrid/Taverna[7], Kepler[8], visTrail[9], chimera[10] etc), essentially, has been proved to be a successful way to perform complex data processing tasks. Figure 2.1, illustrates an example of a typical workflow. Boxes are used to indicate data processing steps and arrows illustrate the data flow. Each task may take input data from the preceding task, user interactions or external sources (database, external tools). During a typical workflow run, the means (e.g. input data, user interactions, user parameters etc.) that are involved in the actual derivation of a result, are not recorded whatsoever. However, the lack of such provenance information makes the outcome of complex analysis difficult to interpret and reproduce. Thus, making these systems provenance-aware will address several issues they suffer from. First of all, scientists will be able to evaluate the correctness of the final workflow output and avoid the duplication of efforts. Furthermore, such provenance information might cater with means for quick troubleshooting or even optimize the whole workflow process[11].

Provenance of data was regarded as an important aspect of the databases storing scientific data. The majority of such databases are usually views of some bigger databases that store raw experimental data. In particular, several scientific stores are populated by the result of queries made to other databases, and are manually updated by several other experts. This sort of databases are commonly known as curated databases[13], due to the number of people and/or systems involved in the data insertion and update process. Thus, we can come to recognize

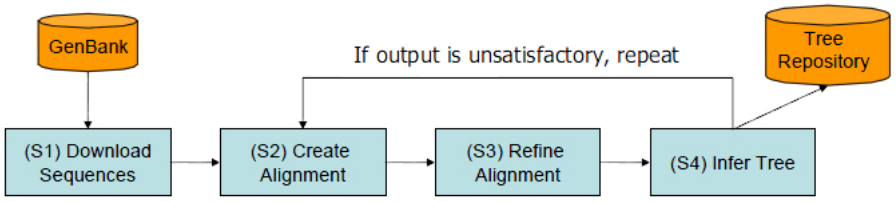

FIGURE 2.2: Scientific Workflow.[12]

that keeping provenance information can help to a great extent to understand the origin of a piece of data, and particularly the process that led to that piece of data. Eventually, we will be able to determine the accuracy, integrity and trustworthiness of data[14]. In the scope of databases, there are two types of provenance that can be observed: "where-provenance", "why-provenance"[15]. The former identifies the tuples that were involved in the production of a query result; whereas the latter helps identify the original location a piece of information was copied from.

Consider the following example

Table 1 Students

Sid	Name	Gpa
1	John	50
2	Paul	60
3	Mark	72

Query Q:

```
SELECT name, gpa
FROM students
WHERE gpa > 70
```

Table 2 Q(Students)

Name	Gpa
Mark	72

Where-provenance can answer to a question such as, where the value gpa ' 72 in the tuple $(mark, 72)$ comes from. The answer would be, that it comes from the field gpa of the tuple with id $= 3$, in the students table. Similarly, the why-provenance identifies that the tuple $(3, Mark, 72)$ was the one that contributed to the tuple $(Mark, 72)$ of the query result. There is a third type of provenance which is called "how-provenance"[16]. While why provenance identifies the source tuples that justify a query result, how-provenance goes a step further and tries to describe how those tuples were involved in the creation of that result.

2.2.2 Provenance Techniques

In this section we present a classification of different methods that have been used to support data provenance. Figure2.3 summarizes the five main aspects according to which we can classify

the techniques for data provenance that have been proposed for use in individual domains[17].

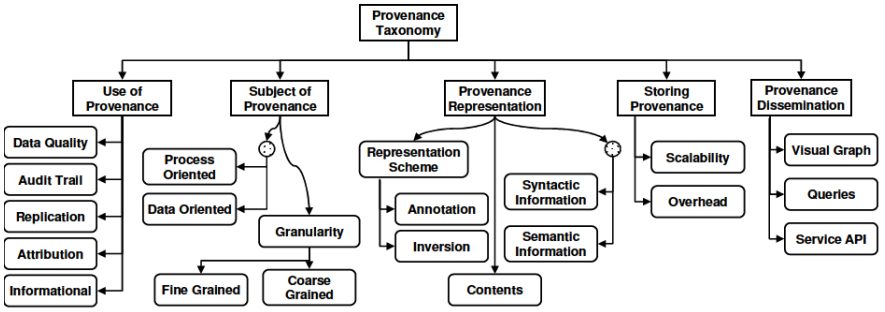

FIGURE 2.3: Taxonomy of Provenance[17].

2.2.3 Subject and Granularity of Provenance Information

When designing a provenance-aware system, one should specify the subject of provenance - the provenance of what? - as well as the level of information detail.

Yogesh, et al.[17] introduce two models of provenance: in a data-oriented model, the provenance of data is compiled explicitly in the form of metadata. For example, a directed acyclic graph (DAG)can explicitly present the provenance of a piece of data, by simply describing the process that led to its current state. Contrary to that, the process oriented model suggests that the deriving processes are entities for which provenance is recorded. Hence the data provenance is implicitly determined by examining the inputs and outputs of those processes.

The granularity refers to the smallest piece of information that provenance is tracked for. For example, in a relational database we can decide to track provenance to the level of rows or cells. Essentially, the system requirements need to determine the level of granularity.

2.2.4 Provenance Representation

There are two major approaches for computing data provenance[1]: (i) non-annotation (or inversion method) approach and (ii) annotation approach.

Consider the example in figure2.4. Q is the transformation function (i.e. query) that acts upon an input database with the aim to generate an output database. This is an example of a non-annotation approach whereby the provenance of the data product (i.e. output database) is computed by analyzing the input and output database, and the definition of Q (e.g. by analyzing the underlying algebraic structure of the query[7]). On the other hand, the annotation approach is slightly different. Provenance is determined by collating extra information in the

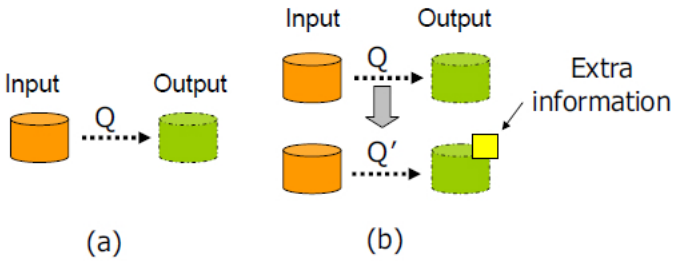

FIGURE 2.4: (a) Non-Annotation (NA) approach, (b) Annotation (A) approach[1].

form of annotations and descriptions about other resources. In this approach, provenance is serialized in a machine readable format (e.g. xml, rdf/xml).

2.2.5 Provenance Dissemination

Regardless the representation format of provenance, one should be able to view and act upon this captured information. A system needs to provide access and present provenance in both, a user and machine friendly manner. A common way is to visually represent provenance graphs. However, there is a significant caveat. Big provenance graphs are not intuitive and make it difficult for a user to interpret. Hence, this approach is not scalable. A better solution is to support a query engine which can extract smaller parts of a provenance graph.

2.2.6 Provenance open data models

In the previous section we examined how provenance systems have been integrated within scientific workflows and database management systems. Such systems, though, have full control of the information they process and store and thereby they track provenance within their own scope. Architecture like that, promotes tight coupling between various systems components (e.g. provenance system, execution environment etc.)[18]

In the era of distributed systems, interoperability is a significant factor that needs to be considered. To that end, systems should adopt mechanisms and techniques that utilize open and technology agnostic models. Several approaches for promoting interoperability rely on an infrastructure that supports several provenance stores[19][20] which offer long-term persistent storage of provenance information. Nevertheless, the representation of the provenance information should be described in a coherent way, regardless of the technologies involved.

The intent of this section is to summarize some early attempts for designing a common provenance model, as well as, the current W3C provenance data model (by the time of this writing, the W3C provenance working group[1] has published a working draft of the specification)

2.2.6.1 Provenance in Service Oriented Architecture (SOA)

Service oriented architecture (SOA) is an architectural style that has been successfully used in distributed systems. The major advantage of a SOA based system is the loose coupling between various software components, which minimizes the system development time, as well as, the maintenance cost. Applications following the SOA approach are mainly composed dynamically by utilizing services available in a network[21] . Thus, the need of a shared and technology-agnostic data model is critical for such systems to be able to manage provenance information.

One of the early attempts to integrate provenance systems into SOA applications was the PA-SOA approach[22]. According to the PASOA model, provenance information is described with the notion of provenance assertions (p-assertions)[3]. P-assertions are, essentially, assertions made by individual services about their involvement in a process execution. Furthermore, they are the main constituents of a process documentation, which in order, provides with description about what happened at execution time (e.g. which algorithms, data sets or services were involved).

Something that is noteworthy at this point is that the full details of the process that brought a data item to its current state, can considerably be huge (theoretically can trace back to the big bang[2]). For example, the full provenance of a result produced by a service that calculates the carbon footprints of individuals might include descriptions of the algorithms that were used, users' input, carbon emission calculation methods, data compiled by external devices (e.g. Gps) and so forth. However, this amount of information might be frustrating for a user who wants to obtain the provenance of a specific piece of data. This fact unveils the need of a query engine where users can make customized queries, so that they can get the information they need and avoid information overload.

During a process execution, there are various tasks carried out. For example, different services might interact with each other by exchanging messages (e.g. SOAP messages) or a single service might apply different transformation functions to input data. As a result, the content of p-assertions can vary. We have three distinct categories of p-assertions based on the content of the provenance information that is being captured.

Interaction p-assertions It is an assertion that describes how data flows among the various services in a SOA system. An interaction p-assertion consists of an interaction key that

[1] http://www.w3.org/2011/prov/wiki/Main_Page

identifies an interaction (i.e. source and destination services), as well as, the content of the message itself (e.g. content of SOAP message)

Relationship p-assertions Assertions about how data slows throughout a single service. They can essentially, describe the function or algorithm that was applied to input data, in order to produce the output message.

Service-state p-assertions During a single execution, a service has a specific internal state. For example, information such as, CPU time used by a service, available space on the disk, user logged in, local time, might be useful to make various interpretations about the computational result. P-assertions demonstrating all these information are called service-state p-assertions.

The aforementioned p-assertions describe the steps in the history of a process execution and can be visualized as directed acyclic graphs (see figure5.8)

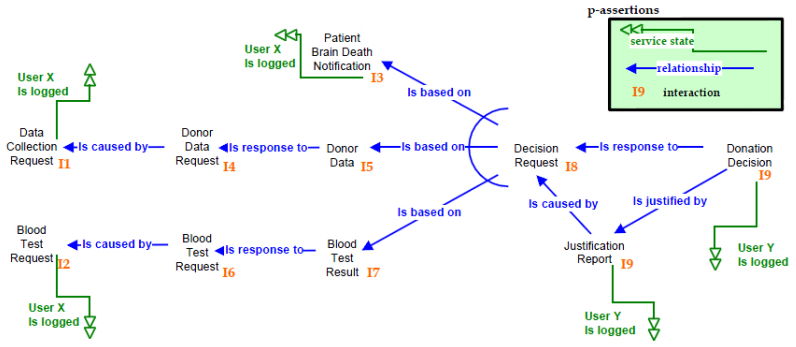

FIGURE 2.5: Provenance Directed Graph[3]

Provenance life-cycle In this section we briefly describe the different phases that constitute p-assertions life cycle[23]. More specifically, p-assertions pass through four distinct phases, which we outline below.

1. **Creating p-assertions:** As we have already mentioned, services are responsible for creating p-assertions. While execution precedes each service can only describe its own involvement in the process execution.

2. **Recording p-assertions:** A provenance store, as described earlier, offers a long-term persistent storage for p-assertions. Hence, after they are created, p-assertions have to be stored in a provenance store, for future use.

3. **Querying p-assertion:** A provenance store offers a helpful mechanism for obtaining the provenance of a specific piece of data. More specifically, a user can compose queries

that will bring the p-assertions according to the users need. In essence, the query will be executed over a process documentation that embraces all the p-assertions.

4. **Managing p-assertions:** P-assertions may be stored in a provenance store for a long period of time. They therefore may need to be managed over the course of time.

A complete provenance system has to support all those phases of the provenance life-cycle.

A Conceptual Provenance Architecture In the previous section we outlined the four main phases of the provenance (or p-assertion) life cycle. For a system to be able to support those phases, a logical architecture that will consist of appropriate system components, has to be designed. Groth et al.[23] suggest generic provenance system architecture (see figure2.6).

In this architecture, we can observe four different actors that are involved in the provenance life-cycle

Application actor It is the system component that processes the business logic of the application.

Provenance store This is the central point in the architecture. As we have already described, it is the responsibility of the provenance store to persist and provide access to the recorded provenance information.

Querying actor This is the part of the system that communicates with the provenance store by issuing provenance queries.

Recording actor While the querying actor sends queries to the provenance store, the recording actor populates it by submitting p-assertions.

Asserting actor This is the actor that creates appropriate p-assertions while the execution proceeds.

Managing actor An actor that performs managing tasks in the provenance store.

It is obvious that the provenance store is the key actor in the provenance lifecycle. It interacts with several other system components and therefore should provide appropriate interfaces to facilitate those interactions.

- A *recording interface* which gives access to recording actors so that they can submit p-assertions.

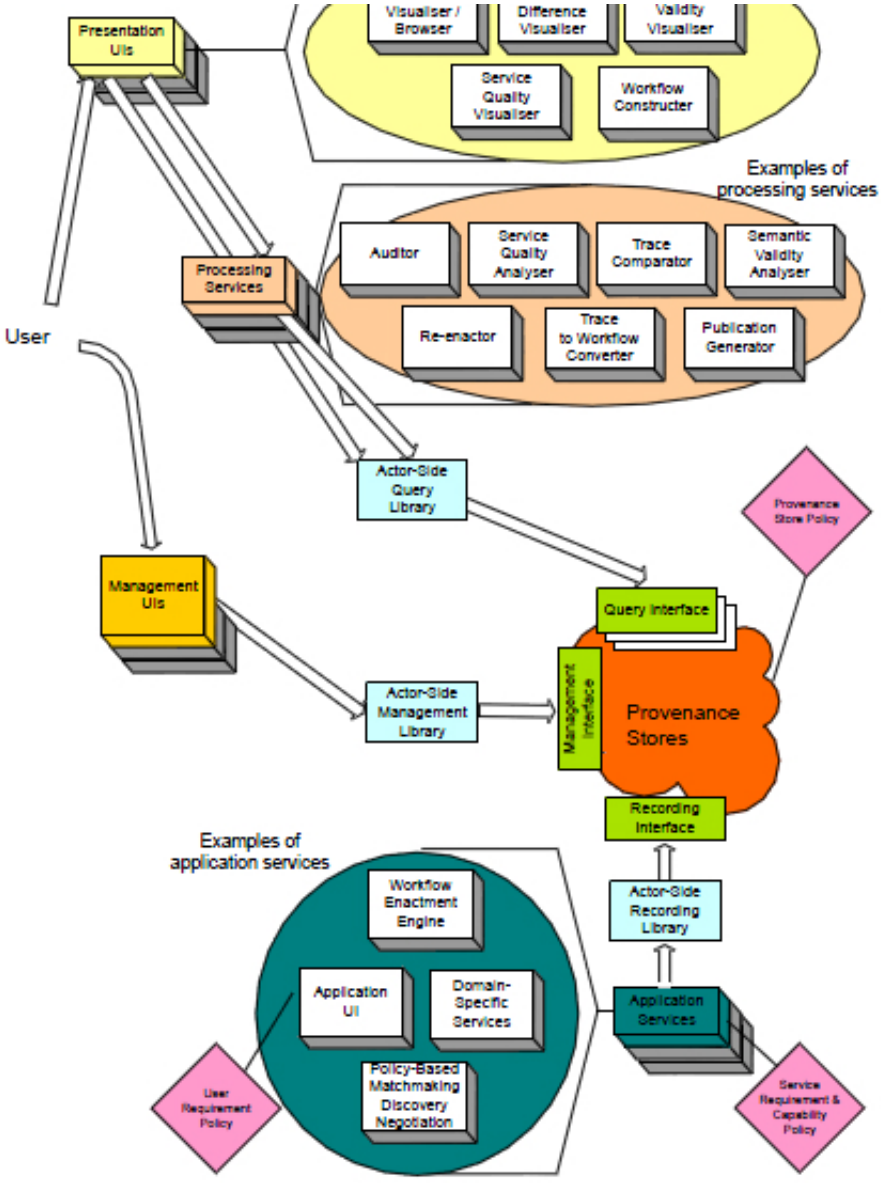

FIGURE 2.6: Architecture of Provenance-Aware Application.[23]

- A *query interface* which is the entry point for query actors so that they can issue provenance queries and get back query results.
- A *management interface* which allows managing actors to perform managing tasks in the store.

2.2.6.2 Open Provenance Model

The provenance data model we presented earlier, describes a shared model for capturing, recording, exchanging and managing provenance information. However, it is primarily bound to SOA applications, that is, to message exchanging systems. The models that we discuss in this section is the first purely technology agnostic provenance data model.

The Open Provenance Model (OPM)[24][25] addresses different provenance interoperability issues by introducing the notion of the provenance interoperability layer[2].

Figure2.7 illustrates a big system which consists of several individual applications. Each of these applications is designed to be provenance aware and makes use of its own provenance store. In a system like that, where information flows across different applications, the history of the derivation of a piece of data (i.e. provenance) might reside across several provenance stores. As a result, one would need to query several provenance stores, to form the whole process execution chain. It is the job of the inter-operability layer to conceal the technology diversity of individual applications and expose provenance data in a uniform manner.

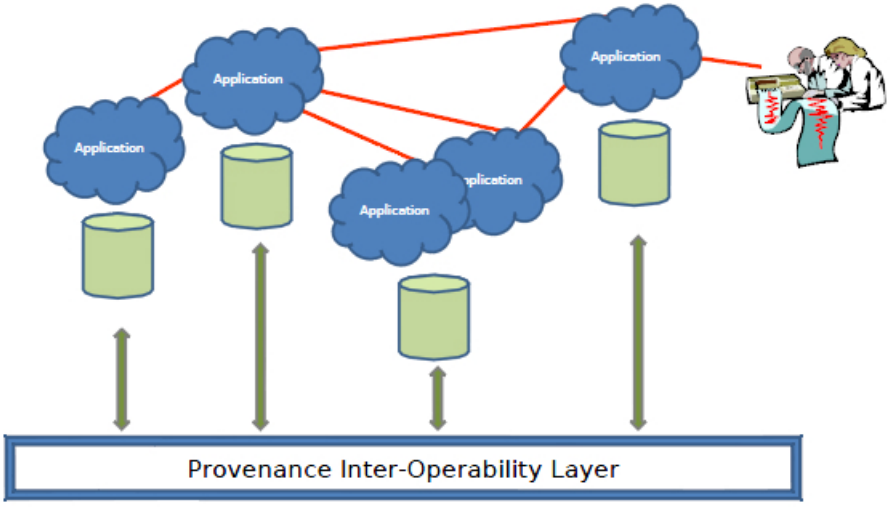

FIGURE 2.7: Cross-System Provenance.[2]

The open provenance model utilizes a graph structure to represent a set of causal dependencies[24] among its nodes. A set of such dependencies can explain how a digital piece of data, physical

resource (e.g. bicycle) or a conceptual entity (e.g. immaterial entity, idea etc.) came to be in a particular state at a specified time.

OPM suggests three types of nodes, which we now present:

Artifact A digital, physical or conceptual entity can have different sets of characteristics, at different moments. We call this specific state of an entity, an artifact. An artifact can be viewed as a snapshot of an entity at a given moment; for example, a pdf file at a specific time. States are immutable in that they cannot be modified. Alterations to any characteristic of an entity at a given time result in the generation of a new artifact.

Process A process refers to an action or a sequence of actions that result in the generation of new artifacts; for example the process of embedding images into a pdf file produces a new pdf file, enriched with multimedia characteristics (i.e. images).

Agent Agent is an entity which triggers a process. In other words, it is the responsibility of an agent to initiate or terminate a process. Following the example with the pdf file, an agent might be a software component that initiated the process of adding images into the new file.

Nodes in an OPM graphs are connected with each other via causal dependencies. Such dependencies are illustrated with directed edges from the source bode (or effect) to the destination node (or cause).

The OPM model introduces an initial set of causal dependencies which are illustrated in figure2.8. This is a graphical representation of the causal relationships between artifacts, processes and agents. Artifacts are represented by ellipses; processes are represented by rectangles; and agents by octagons.

We observe a set of five types of causal dependencies based on participants.

Used A relationship between a process and an artifact denoting that the process used the artifact to produce a result (e.g. a new artifact). The formal definition suggests that the availability of the artifact is critical for the process to complete the execution.

wasGenerateBy It is a relationship between an artifact and a process, indicating that the artifact was generated by the process. In other words, the existence if the artifact is due to the execution of that process.

wasTriggeredBy Figure2.8 shows an edge wasTriggeredBy from a process p2 to a process p1. The semantics of this relationship defines that there was some unknown artifact that was generated by some process p1 and was used by a process p2.

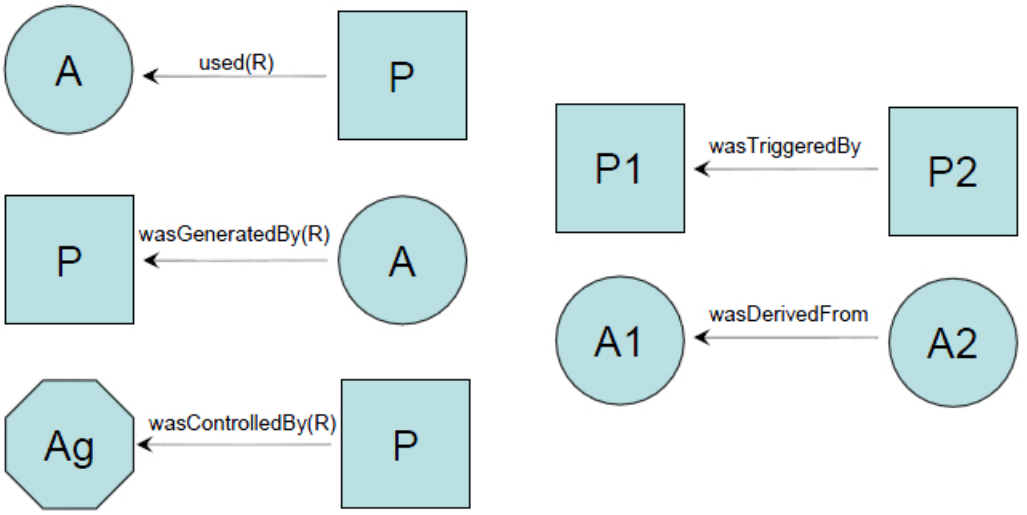

FIGURE 2.8: Provenance Causal Dependencies[24]. Note that past tense aligns with the notion of process history documentation.

wasDerivedFrom An edge wasDerivedFrom from an artifact node A2 to another artifact node A1 denoted that there was a process that used an artifact A1 to generate an artifact A2. It is a useful relationship, when the actual process of a transformation is unknown.

wasControlledBy It is a causal dependency between a process and an agent. It simply indicates that the execution (i.e. the start and end) of a process is carried out under the control of an agent.

Note the letter R in the "used", was controlled by" and "was generated by" edges. This is a bit of extra information that is used in the aforementioned relationships to characterize the role of each entity (i.e. artifacts, agents) that is connected to the same process. More specifically, in a "used" relationship, the same process may use several artifacts. However, the nature of each usage might be different; for instance, an online process for searching for news based on a keyword and a specified source uses the keyword with the role "keyword to search" and the source web site with the role "web site to search". Similarly, a process can generate several artifacts with different role each. Following the previous example, the process can produce a news article and a set of additional sources. The former will have the role "article associated with the keyword" whereas the latter will have the role "additional sources to look at". Finally, a process can be controlled by multiple agents, each with a distinct role. In the example of the online news fetcher, the process for searching news can be initiated either by a user via a user interface, or by an external application via an application programming interface (API). The Roles in this case might be "internal actor" and "external actor" respectively.

One of the concepts associated with provenance that we have been constantly highlighting, is the ability to specify the scope of the provenance that is being captured. A provenance of a piece of data can be expressed in various levels of detail. Furthermore, it can be viewed from different angles and standpoints. To that end, OPM introduces the notion of accounts. An **account** is regarded as a graph coloring, indicating different provenance sub-graphs which provide details at different level of abstraction.

The concept of accounts is fairly useful and caters users with multiple descriptions of the provenance of the same piece of data. A final note about OPM concerns the extensibility capabilities of the model. Stronger interpretations of the causal dependencies we described in this section can be used to serve application-specific requirements. A common way to achieve that would be to design an ontology with subclasses of those causal dependencies.

2.2.6.3 The W3C Provenance Working Group

The open provenance model was a good initiative to tackle with inter-operability issues pertaining to the provenance dissemination across multiple platforms. However, as is the case of many widely used technologies, there should be a consensus among different bodies (e.g. industry, academia etc.) about the final specification. For this reason, the provenance working group was formed in the World Wide Consortium (W3C). The group followed the work that was started by the Provenance Incubator Group[2]. The ultimate goal of the group is to reach an agreement upon the specification of provenance on the World Wide Web.

At the time of this writing, the provenance working group has published a set of specifications which are still on the "working draft" step of the W3C specification approval process. In particular, the specifications that have already been released are:

PROV Data Model A data model for describing all the parties that are involved in the generation of a data item[26].

PROV Ontology A OWL 2 web ontology for expressing the PROV data model. In essence, it is the "vocabulary" that is used by the PROV data model to represent provenance information[27].

PROV Notation Specification which aims to design the means for a human friendly representation of the information expressed by the Prov data model[28].

PROV Constraints A document that defines a set of constraints that assures the validity of PROV instances (statements). Further, it specifies inference rules for reasoning over

[2] http://www.w3.org/2005/Incubator/prov/wiki/W3C_Provenance_Incubator_Group_Wiki

PROV instances. This can result in the creation of additional statements implied from the explicitly stated provenance information[29].

PROV Access and Query A document that describes the means to leverage the existing web infrastructure, in order to query and obtain provenance information[30].

In this section, we will present the PROV data model (PROV-DM). For additional information about the rest specifications, we refer the reader to the official web site of the W3C provenance working group[3]

PROV Data Model Provenance can be viewed from different standpoints, and therefore various kinds of information can be stored in provenance records[31].

The provenance data model identifies three distinct perspectives of provenance:

- **Agent-centered provenance:** emphasis is given on the entities that took part in the generation or manipulation of the data item in question; for example, considering the provenance of a video file in a blog post, we might want to know the person who recorded that video, the person who edited it, and the user who posted it on the blog.
- **Object-oriented provenance:** a different perspective suggests that we might want to identify the origin of different parts of a document; for example, in the same blog post, we may want to know that the video was taken from YouTube and the images from Flickr.
- **Process-oriented provenance:** finally, we may want to highlight the actions that were carried out to generate a data item; for example, a visual graph illustrating the connections I have in a social network site may have been generated by invoking a service to pull data from the social network, which then are processed by a JavaScript library for visualization.

Regardless the various viewpoints that provenance can be examined, a good data model should provide all the means needed to capture adequate information. To that end, the PROV-DM introduces some key concepts, which we outline in the following lines.

An entity is the equivalent to an artifact in the open provenance model. It may refer to a physical, digital or immaterial thing, such as a blog post, a car or a decision. The second, constituent is the activity. This is equivalent to a process in OPM model and represents the derivation of an entity. In particular, it refers to the process of generating a new entity or the transformation of one entity to another; for example, the process of calculating the carbon emissions of an individual based on some input data is regarded an activity. Finally, an agent

[3] http://www.w3.org/2011/prov/wiki/Main_Page

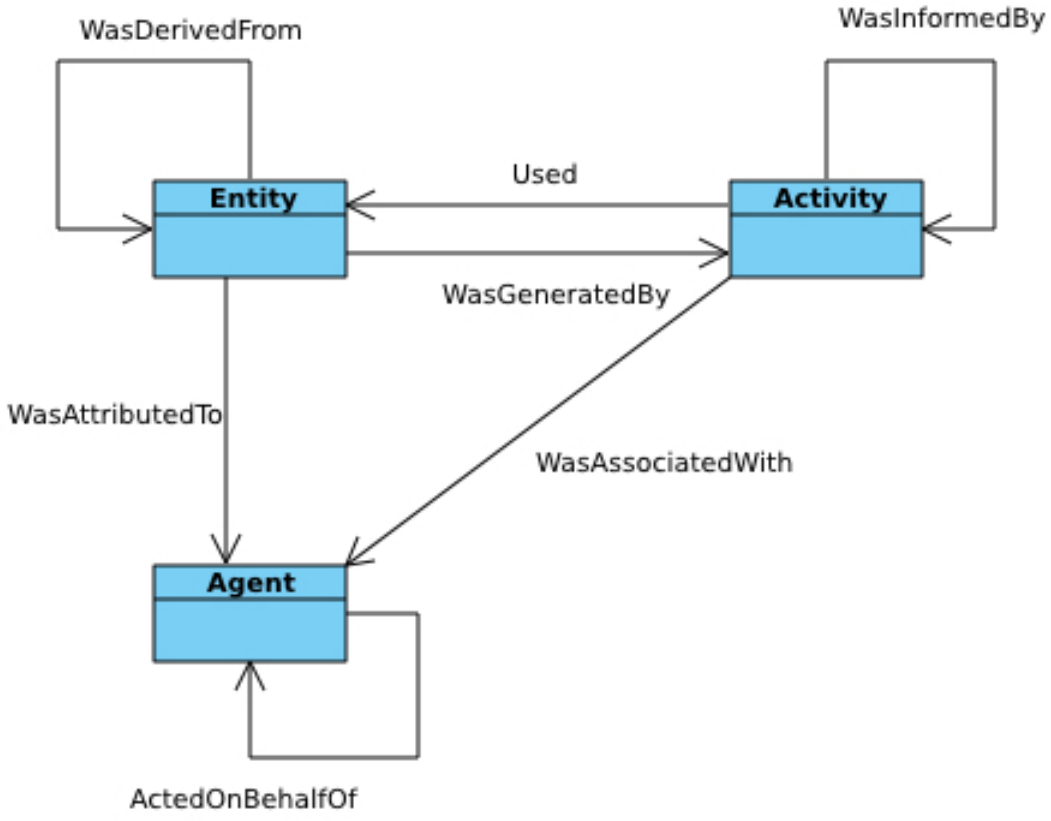

FIGURE 2.9: PROV-DM[31]. Note that past tense aligns with the notion of process history documentation.

in PROV-DM is assigned a degree of responsibility for an activity that is being carried out. The concept of an agent is similar to that of an actor in the OPM model, in that it represents a "thing" that is responsible for controlling the execution of an activity. An agent can be a human being, an organization, software or any other entity that may initiate an activity; for example, a JavaScript visualization library can be an agent that is responsible for imitating a graph visualization activity.

At this point we have to note that provenance can be expressed only for entities. However, we can make provenance assertions about an agent, should the agent be declared both as an agent and an entity. These are the main concepts of the PROV-DM data model. In addition to that, the data model encompasses several other relationships (for additional information, refer to the official document[26]).

Entity Generation and Usage An activity in PROV-DM can be related to an entity in two different ways. First of all, the effect of the actions that comprise an activity is usually the generation of new entities. Hence, entities exist due to activities existence; for example, assembling car parts brings a car into existence. Additionally, activities can use entities, often

during the process of generating other entities; for example, a visualization activity can use some datasets stored in a database, in order to create a chart. These two relationships are represented in the PROV-DM with the terms *wasGeneratedBy* and *used* respectively

Agents' Responsibility Agents can be associated with entities, activities or other agents. Relationships between entities and activities come in the form of responsibility that an agent has over them. We have already highlighted that agents are responsible for the execution of an activity. A responsibility might refer to an agent initiating or terminating an activity. For example, in PROV-DM we say that an activity was associated with an agent. Similarly, an activity can be responsible for an entity. This form of responsibility can be interpreted as the attribution of an entity to an agent. In other words, the was AttributedTo relation, from an entity to an agent, denotes that an agent was responsible for an activity that was the cause for this entity to be created. This is useful relation when the activity is unknown or of a little significance. Finally, PROV-DM provides mechanisms to express relations between two agents. In particular we say that an agent can act on other agent's behalf; for example, an employee can be designated by the company to perform some actions on their behalf.

Entity Derivation The last core relationship defined by the PROV-DM is a relation between two entities. The existence of an entity might often be due to some other entity that was used by the activity that generated it. In that case, it is said that one entity was derived from another entity. A special kind of such relation is the "was revision of"; for example, the PROV-DM specification may go through multiple revisions over the course of time. Such relation can be useful to examine the changes that had been brought across the different revisions of a document.

Roles Recall the notion of roles in the OPM model. The PROV-DM has adopted the same notion to characterize relationships between entities and activities. In particular, a role specifies how an entity was used or generated by an activity. Similarly, roles can specify the nature of agents' involvement in an activity. It is obvious that roles are application specific, and therefore are not specified by PROV-DM.

Plans and Accounts Activities may consist of numerous steps and procedures. For instance, an online tutorial for publishing links data on the web illustrates several steps that should be taken. In PROV-DM there is a specific term to represent this set of actions. In PROV-DM language, this term is called **plan**. Plans are entities, therefore we can describe provenance for them. Finally, one would wonder how we can trust the provenance information about a resource. In fact, this is a plausible question, since anyone can make provenance assertions

about any entity. The PROV-DM supports a means to address this issue, namely it includes the notion of accounts. In essence, an **account** is an entity that contains some provenance information, and because it is an entity, we can express a provenance for it. That is to say, provenance of provenance can be expressed.

```
Entity(ex:w3c-publication.pn, [prov:type="prov:Account" % % xsd:QName])
wasAttributedTo(ex:w3c-publication.pn, w3:consortium.)
```

In the above example, which is written in PROV-Notation, we have explicitly declared the "ex:w3c:publication.pn" provenance description. We then state that "w3:consortium" agent was responsible for its generation.

In this section we presented only the core principle of the PROV-DM specification. However, the whole specification is very rich and consists of many more concepts and definitions. For this reason, we would recommend the reader to visit the provenance working group web site and read the whole set of PROV specifications.

2.2.7 Security

A provenance-aware system is, essentially, capable of recording and storing provenance information about different assets that reside in the system. As we have already discussed, this sort of functionality is accomplished with the aid of a data model, which defines the way that provenance information is represented internally. Further, the presence of a provenance store guarantees the long-term storage of provenance information, as well as, provides a means for querying the store[32]. The result of a provenance query is a causal graph or provenance graph, which illustrates the provenance information in a way that can be viewed an analyzed.

Throughout the process described above, there is nothing that guarantees the quality and integrity of provenance information. To address this problem a framework that will consider all the security-related aspects of the provenance information, have to be devised. Such framework can leverage existing technologies (e.g. cryptography), that have been successfully applied on the web. We will start our discussion by introducing some fundamental concepts and technologies pertaining to the security in computer systems.

2.2.7.1 Fundamental concepts

A discussion about security issues concerning the communication between different entities, which exchange some sort of information, should start by defining some initial security requirements. More specifically, there are five requirements that a secure system must provide.

Confidentiality A system has to guarantee that all the information that is being exchanged between different entities is protected against eavesdroppers; for example, the information that we exchange when purchasing a product on eBay, should not be wire tapped by anyone.

Integrity A message sent from one entity to other should not be altered, whatsoever; for instance an order sent to eBay has to remain intact while traveling through the wire.

Authentication A mechanism that blocks unauthorized users from accessing the systems data and functionality. Access should be granted only to those users who can provide adequate information, in order to prove their identity; for instance In order to be able to perform a purchase action on eBay, one should provide his credentials.

Non-repudiation An important requirement in the communication between two entities is to be able to prove that a message was indeed sent by its sender; for example, once a user submits a purchase order, she cannot claim that has not done so.

Authorization This requirement is also known as *access control*. It is, essentially, a mechanism that determines the sort of access that a user can have on systems resources; for example, as an ordinary user, I cannot modify the systems database by deleting items that I am not authorized to delete.

2.2.7.2 Cryptography

Authorization and authentication are two requirements that can be achieved fairly easier that the rest. To support authorization, a system can simply apply one of the proposed models for access control (e.g. MAC[33], DAG[34], RBAC[35], ABAC[36] etc.). Similarly, authentication can be achieved by leveraging the current web infrastructure and the HTTP protocol capabilities (e.g. content negotiation). To meet the rest requirements, a system can use cryptography technics. More specifically, confidentiality can be ensured with encryption[37] whereas non-repudiation with digital signatures[38]. Finally, integrity can be guaranteed with either of those two technics. We review those technologies in the following sub-sections.

Symmetric Encryption This type of encryption requires that the entities that participate in a message exchange communication use the same key for the encryption and decryption phase. For example, figure2.10 illustrates a message exchange scenario.

In this scenario Alice wants to send a message to Bob. As shown in the figure, she encrypts the original data (plaintext) with a private key, and sends it to Bob who decrypts the encrypted message (cipher text) using the same private key. Some of the algorithms that can used to encrypt data are: 3DES, DES, AES, RC4

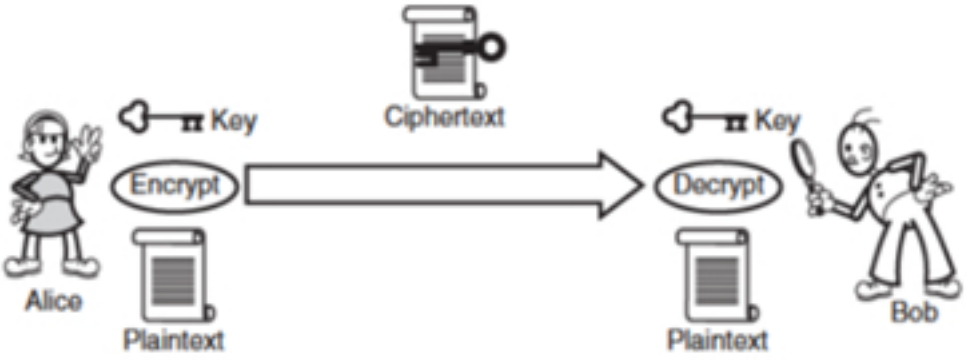

FIGURE 2.10: Symmetric encryption[39].

Asymmetric Encryption Asymmetric encryption is more flexible, in that two different keys are used: a private and a public key; for example, figure2.11 demonstrates the same scenario but now an asymmetric encryption is used. Alice uses Bob's public key to encrypt the plaintext and sends it to Bob, who decrypts the cipher text using his private key.

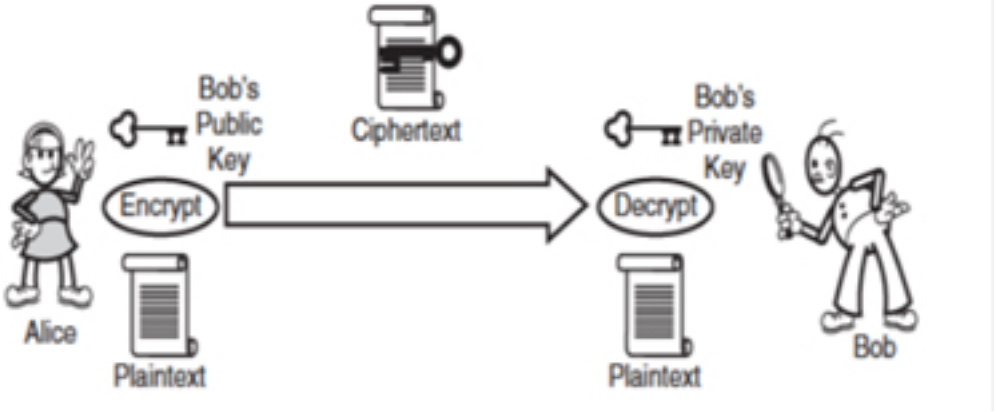

FIGURE 2.11: Asymmetric encryption[39].

Digital Signature Integrity and non-repudiation can be guaranteed with the use of digital signatures. To better describe this technology look at figure2.12. Alice has a pair of private and public keys.

She creates a signature value with her private key and sends it to Bob along with the original plaintext. On the other side, Bob uses Alice's public key to generate a signature value based on the content of the message he received. He then can compare the two signatures to verify the integrity if the incoming message.

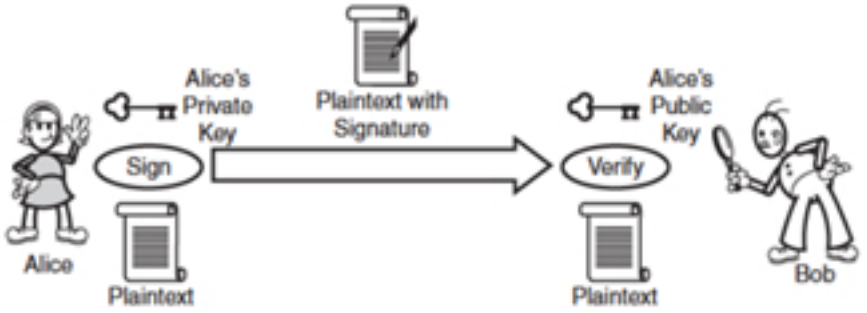

FIGURE 2.12: Digital Signature[39].

Asymmetric versus Symmetric Encryption A noticeable caveat of the symmetric encryption is the key distribution problem. Both participants need the same key to encrypt and decrypt a message; therefore they need to somehow exchange it. This has to be done carefully to avoid malicious attackers from stealing the transmitted keys. On the other hand, the most evident downside of the asymmetric encryption is that it exhibits very poor performance. Asymmetric encryption is much slower that symmetric.

2.2.7.3 Security protocols

SSL is probably the most prevalent security protocol for distributing keys in a safe mode. It, essentially, defines the requirements for a secure exchange of keys between two parties. The main steps described by the protocol are the followings:

1. The client communicates with a server
2. The server responds back by returning its certificate
3. The client computes a random number which represents a seed for generating private keys. He then encrypts this number using the public key included in the server's certificate and sends it to the server.
4. The server obtains the random number by decrypting the received message
5. Ultimately, both parties have the same seed; they therefore can create the same private key.

2.2.7.4 Public key infrastructure

As described earlier, in the asymmetric encryption we do not need to bother about how to distribute private keys, since we can share a public key. However, it is critical that we associate

the public key with a particular party. In other words, the owner of the public key has to be identified. To that end, the **Public Key Infrastructure** (PKI)[40] has been designed. The main constituent of PKI is a certificate, which is a digital resource that associates a participant to a public key. This certificate is digitally signed and issued by a trustworthy third-party authority, which is called **Certificate Authority (CA)**. There is a relatively small number of CAs in PKI and they can issue certificates to other certificate authorities.

To better describe this infrastructure, consider the following example. Figure2.13 illustrates a typical PKI scenario.

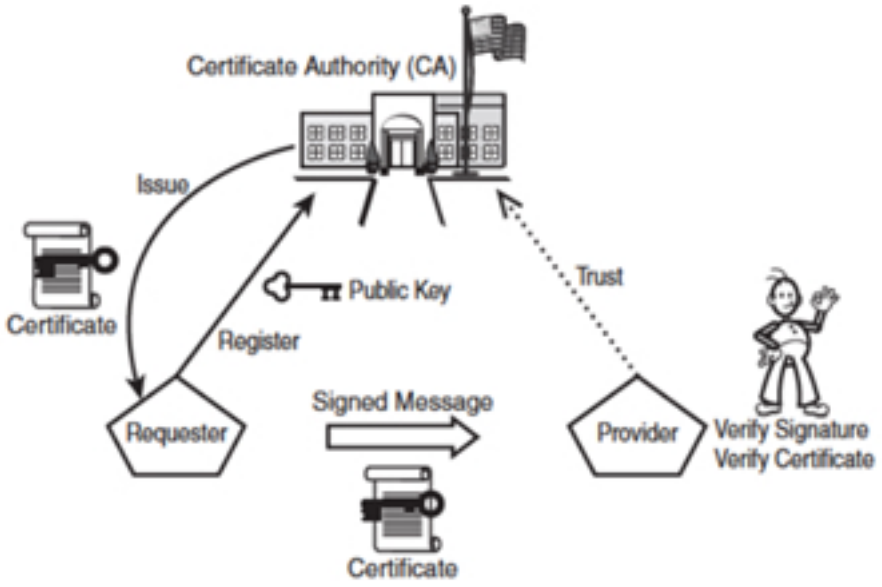

FIGURE 2.13: Using certificate for a digital signature[39].

Alice signs a message with a private key and sends it to Bob along with a certificate. Bob creates a signature value using the public key contained in the certificate. But before that, he has to verify that the certificate is signed by a trustworthy certificate authority.

2.2.8 Publishing and querying provenance information

An important task for a provenance-aware system is to capture and store provenance information. An equally crucial feature is for someone to be able to locate that information and ultimately query and retrieve it some format[41]. For that reason, an infrastructure should be supported so that external applications, users or the application itself could reuse the captured provenance information. Such infrastructure should resolve the problem of locating, retrieving and querying that information. The most appropriate solution is to leverage the current web infrastructure, as well as, some semantic web technologies. In more detail, *uniform resource*

identifiers (URIs)[4] can be assigned to provenance information (provenance-URIs), as well, as, its constituents (i.e. entities, activities and agents). Any application can then perform HTTP GET[5] requests to access the information that is associated with the Uri in question[30]. This is similar to the way URIs are dereferenced when accessing web resources. Nevertheless, it not a strict requirement to assign unique URIs, due to the difficulties that such process has. Thus, alternative methods can be provided to facilitate the same functionality (we discuss them further in this section).

Locating provenance information is crucial and indispensable in scenarios where a provenance-URI is unknown. In such occasions, an application or a user has to provide some additional information to a provenance provider, which can be a third-party application. According to the submitted information, a provider can locate the provenance-URI and/or the provenance information itself.

Provenance information, as already mentioned, consists of causal relationships between entities, activities, and agents. As such, it is a requirement that one can retrieve descriptions about those resources. The concept is similar, URIs (entity-URIs) can be assigned to those resources and one of the following mechanisms can be provided:

- In the case where the requester knows the entity-URI, a simple HTTP mechanism can be used to access the dereferenced content. More specifically, if a resource is accessible via an HTTP GET request, an additional link header denoting the URI where provenance for that resource is located can be injected into the header of the response.

  ```
  Link: provenance-URI; rel="provenance"; anchor="entity-uri"
  ```

  The *entity-URI* indicates the location where additional description for that resource can be obtained, whereas the provenance-URI identifies the location of the provenance information which the entity is part of. If the provider though, does not know anything about provenance locations, it can point to a third-party provider or a provenance service. In that case, the link header is slightly different.

  ```
  Link: provenance-service-URI; rel="provenance-service";
  anchor="entity-uri"
  ```

  The *provenance-service-URI* will, essentially, return a service description that will aid a client to locate a provenance-URI or fetch provenance information for the entity in question.

[4] http://www.w3.org/TR/uri-clarification/
[5] http://www.w3.org/Protocols/rfc2616/rfc2616.html

- For resources that are presented in HTML, provenance-URI can be included within the document. The simplest way to do that would be to use the <Link>html element. Consider the following example.

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <link rel="provenance" href="provenance-URI">
        <link rel="anchor" href="target-URI">
        <title>Welcome to example.com</title>
    </head>
    <body>
       ...
    </body>
</html>
```

  Just as is the case with the previous approach, the document provider can point to a provenance service where the information is stored. In that case the HTML snippet will resemble the following one:

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <link rel="provenance-service" href="service-URI">
        <link rel="anchor" href="target-URI">
        <title>Welcome to example.com</title>
    </head>
    <body>
        ...
    </body>
</html>
```

- Finally, if a resource is represented in one of the several RDF[6] serialization formats (i.e. RDFa, XML/RDF, N-Triples, Turtle etc.) then additional RDF triples can be inserted to describe the provenance of the resource. For this reason, the property *prov:hasProvenance* is defined. The object for that predicate is the resource that contains provenance information about a resource that is subject of this triple.

```
@prefix prov: <http://www.w3.org/ns/prov#>
<> dcterms:title      "Welcome to example.com" ;
```

[6] http://www.w3.org/RDF/

```
prov:hasAnchor     <http://example.com/data/resource.rdf> ;
prov:hasProvenance <http://example.com/provenance/resource.rdf> ;
prov:hasProvenanceService <http://example.com/provenance-service/> .
  :
(RDF data)
```

2.2.8.1 Provenance Services

There might be occasions where the provenance-URI is unknown. In such cases, a provenance service can provide a means for discovering and/or retrieving provenance information. A provenance service might be supported by a third-party provider, and is uniquely identified via a service-URI. Dereferencing, this URI will bring a service description, which provides guidelines to clients about how to locate or/and fetch provenance information. Two mechanisms are supported to accomplish that task: the provenance discovery service mechanism gets the URI for some resource and returns a set (one or more) of URIs, pointing to the location where the provenance information can be acquired. Conversely, the provenance retrieval service is used when there is no URI associated with the provenance information. As such, the service is responsible for locating and retrieving that information, on client's behalf.

Details about these services are described in the service description, which should be available in RDF (see example below).

```
<service-URI> a prov:ProvenanceService ;
    prov:provenanceUriTemplate "service-URI?target={+uri}" .
```

The object of the *prov:provenanceUriTemplate* is a literal value that contains a URI template[7]. Users can replace the variable uri with actual uri of the entity for which provenance is required.

An alternative option is to use a query engine. For interoperability's sake, it is preferred that a SPARQL endpoint is used. Requesters can use a simple SPARQL query including an entity-uri, to get the corresponding provenance-uri.

```
@prefix prov: http://www.w3c.org/ns/prov#

SELECT ?provenance_uri WHERE
{
  <http://example.org/resource> prov:hasProvenance ?provenance_uri
```

[7] J. Gregorio; R. Fielding; M. Hadley; M. Nottingham; D. Orchard. URI Template. March 2012, Internet RFC 6570. URL: http://tools.ietf.org/html/rfc6570

```
}
```

If the requester does not hold any URI, but knows some details about the resource, then those details can be included in the SPARQL[8] query; for example if the DOI identifier for a document is known, then the following query can be submitted to the SPARQL endpoint:

```
@prefix prov: http://www.w3c.org/ns/prov#

@prefix prism: <http://prismstandard.org/namespaces/basic/2.0/>
SELECT ?provenance_uri WHERE
{
  [ prism:doi "1234.5678" ] prov:hasProvenance ?provenance_uri
}
```

Finally, if specific elements of provenance information are required, then the SPARQL query may look like the following:

```
@prefix prov: <http://www.w3c.org/ns/prov#>
SELECT ?generationStartTime WHERE {
    <http://example.org/resource> prov:wasGeneratedBy ?activity .
    ?activity prov:startedAtTime ?generationStartTime .
}
```

In this example, the client wants to retrieve the start time of an activity in the provenance information in question.

2.3 Carbon Footprints

During the last century, human activities brought severe damages to the environment. This has been more evident during the recent years, where anthropogenic environmental impacts have led to severe climate changes[42]. To that end, several mechanisms to tackle with this problem need to be devised. In this section we introduce the fundamental terms and concepts that underpin the process of quantifying human's impact on the environment.

[8] http://www.w3.org/TR/rdf-sparql-query/

2.3.1 Fundamental Notions

Carbon footprinting is probably the most prevalent method for quantifying human's impact on the environment. More specifically, the term "footprint" can be associated with quantities of *Greenhouse Gas (GHG)* emissions[43] caused by activities, individuals, systems or populations. In cases where a more accurate evaluation of climate risk is needed, a measure called *climate footprint*[44] can be used. Figure 2.14 illustrates the distinction of those measured. We can notice a third layer which corresponds to a yet broader set of GHGs emissions and is called *GHG inventory.*

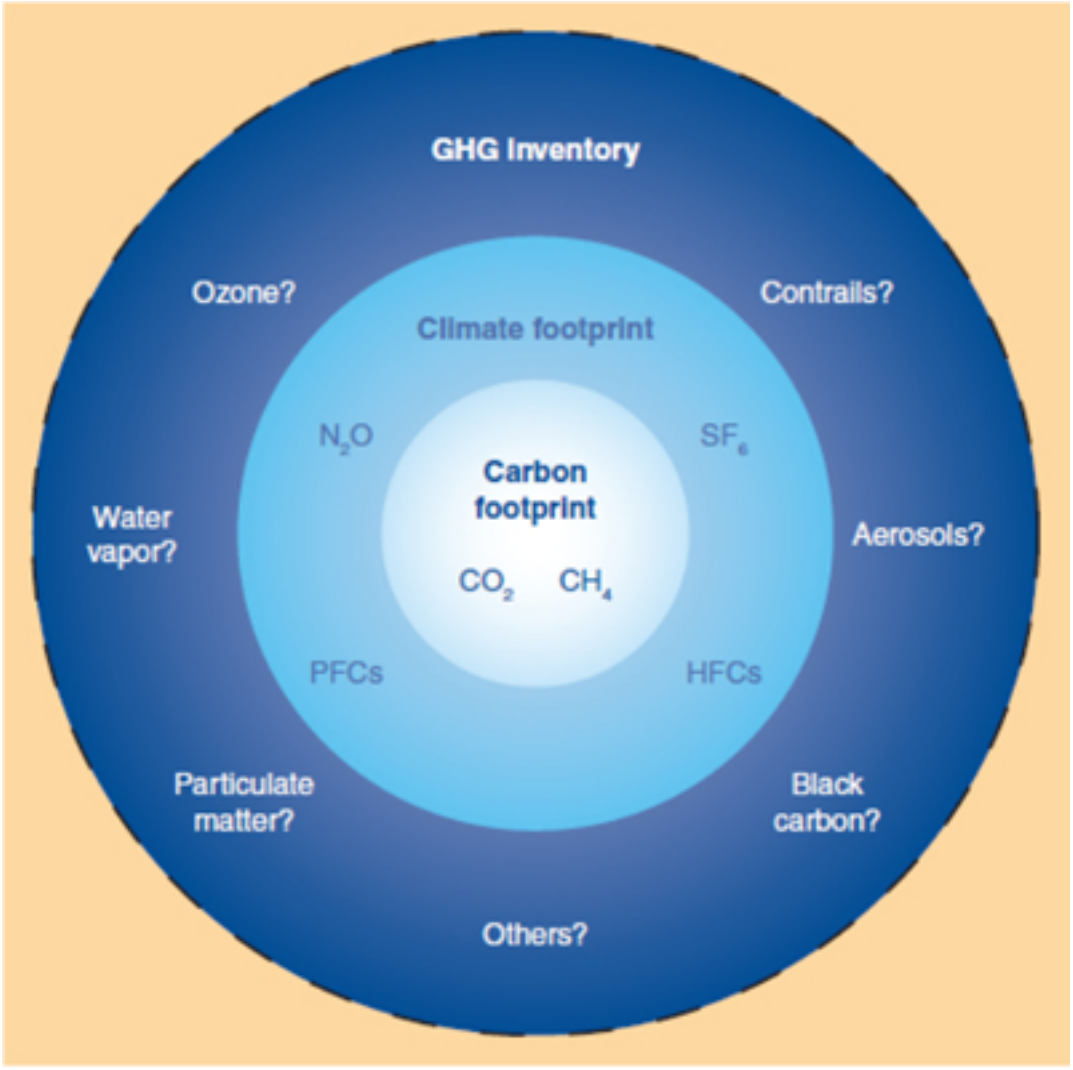

FIGURE 2.14: Three measures for carbon emissions[45]

2.3.2 Calculating carbon emissions

The process of calculating carbon footprints is relatively straightforward and consists of the following steps.

1. Initially, all emission sources for the subject need to be compiled and classified; for example, an emission source for an individual might be his daily travels to work with bus.
2. A method for quantifying the emissions of each source has to be selected. The unit for that quantity is expressed in CO2e, which combines CO2 and CO4 emissions.
3. All the data that are needed by the method have to be gathered; for example, a method might need some activity data[46] and an appropriate conversion factor (or emission factor); we will come back to these terms in a bit.
4. A documentation of the method is vital, so that computed values can be validated by re-executing the same method, in the future. Provenance, which we discussed in previous sections, can be a good solution.

2.3.3 Gathering Emissions Sources

The carbon footprint calculation process starts by gathering the emissions sources for the subject. To better identify possible emissions sources we can distinguish two types of emissions: direct and indirect emissions.

Direct are the emissions which the subject has full control of. These are the emissions that are primarily produced during the combustion of fossil fuels. On the other hand, indirect emissions are not directly associated with a specific activity, but are due to the demand for products brought by the subject in question; for example, each product that we buy consists of a number of activities that result in the product being on super-markets shelves. Apparently, all these activities cause CO2e emissions. This implies that our demand for products indirectly produced a quantity of CO2e that was emitted to the environment.

2.3.4 Classifying Emissions

The prime reason that lies behind the decision for categorizing emissions is the problem of double counting. In essence, it refers to the problem of associating the same quantity of carbon emissions to multiple subjects[47]. To avoid this issue, emissions are classified into three distinct scopes.

Scope 1 In scope 1, the direct carbon emissions of individuals, activities or systems can be found; for example, carbon emissions produced by a car owned by a company.

Scope 2 Scope 2 consists of emissions that are produced from energy generation. It basically, refers to energy purchased by the subject in question, for own consumption.

Scope 3 Emissions which are the result of the subject's actions that take place outside the organization or geographic boundaries are scope 3 emissions. In general, this sort of emissions are said to be shared by several subjects rather solely belong to one.

2.3.5 Calculation Methods

The next step in calculating emissions is to select an appropriate calculation method, based on the data that are available. The simplest formula suggests that the number of units of the activity that occurred be multiplied by an appropriate conversion factor (emission factor[46]. To better illustrate this concept, imagine that the carbon emissions of a trip made by car needs to be calculated. In that case, we can multiply the distance traveled (or fuel consumed) by a value that defines the CO2e emissions of that car per kilometer units.

According to the accuracy and quality of the data that are present, calculation methods can be classified into three categories or tiers; if we choose to use the fuel consumed by a car then several factors that determine the quality of this figure should be considered. The type of the journey, the vehicles age and condition are some of the factors that affect the quantity of fuel consumption. A tier Three method will consider all those factors, as it will use the actual quantity of the fuel used. Further, it will multiply this number by an emission factor published by a government body and which is associated with the particular fuel type. A tier Two method will be less accurate. It will take the distance traveled as the main activity data and multiply it by the emissions factor for the specific car model. Finally, a tier One method will yield the least accurate figure, because it will use the least specific data. In particular, an average distance (published by some government's authority) for that sort of cars will be multiplied by the emission factor associated with that fuel type.

2.3.6 HEI Scope 3 Carbon Emissions

In this section we will briefly describe how higher educational institutions (HEI) should calculate scope 3 carbon emissions caused by travels made by the institution's members[48]. First we will remind that scope 3 emissions are those that are caused by sources that are not owned by the HEI. A representative example would be, emissions that are produces by commuting travels by a transport mean that is not owned by the HEI e.g. bus, train etc.

There are two categories of travels: business and commuter travels. The main factor that determines the scope of carbon emissions is the mode of transport that was used; for instance, emissions caused by transport means owned by the HEI are scope 1 emissions. However, travels are often made by modes of transport owned by third-party bodies; hence the corresponding emissions are regarded as scope 3 emissions.

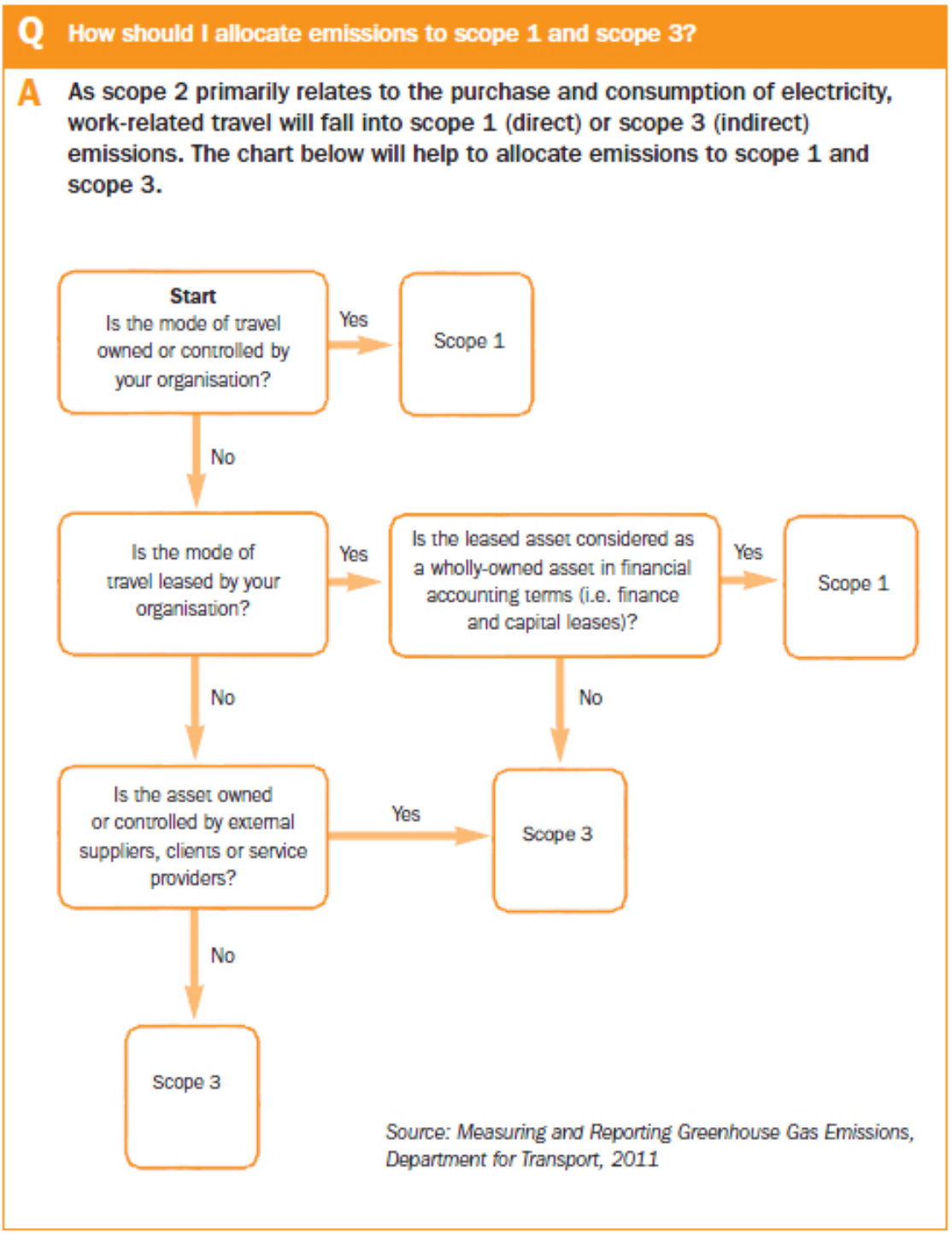

FIGURE 2.15: How to distinguish scope 1 and scope 3 emissions[48]

2.3.7 Travel emissions calculation

The process of calculating emissions caused by travels is the same with what we have presented earlier. As far as travels are concerned, the type of transport mean determines the emission factor that will be used in the calculation formula.

2.4 Carbon Footprint Applications

In this section we will present some of apps that are similar to the one we have developed, in that they calculate individual carbon footprints. The pros and cons of each are highlighted.

2.4.1 CarbonFootPrint.com

This is an online carbon emissions calculator. The tool is hosted on the www.carbonfootprint.com web site[9]. Users can insert some data concerning their household activities (e.g. kWh of electricity consumed), as well as, the distance they have traveled by various mode of transport; for instance, figure 2.16 shows the form for adding the distance traveled by a car.

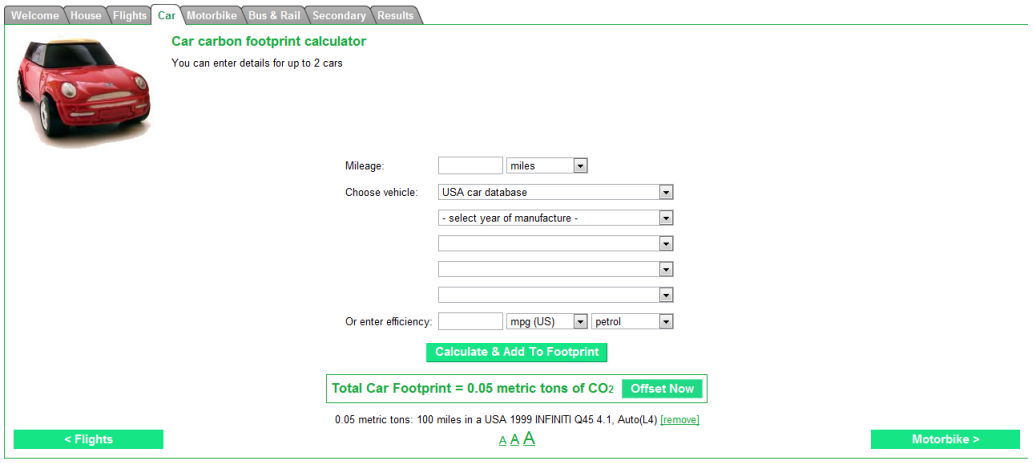

FIGURE 2.16: Form for adding travels made by a car

The tool summarizes all the carbon emissions via a web UI, where carbon footprints of individuals are presented.

A factor that might hinder users from tracking their carbon footprints is the knowledge of traveled distance. It is obvious that such information is difficult to be possessed or can be erroneously inserted.

[9] http://www.carbonfootprint.com/calculator.aspx

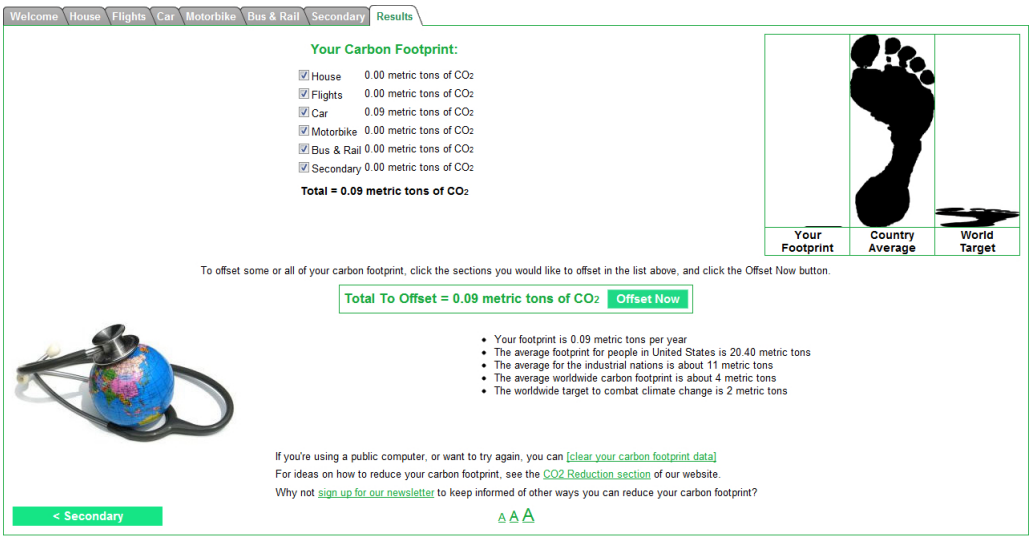

FIGURE 2.17: Carbon footprint report

2.4.2 AMEE Location Footprinter

A slightly more usable tool is provided by AMEE[10] and is called AMEE Location Footprinter[11]. It is a web based tool that operates in conjunction with Foursquare[12]. The biggest advantage of the app is that it does not require users to insert any data. It, essentially, reads all the check-ins the user make on Foursquare, finds the distance between those check-ins and tries to identify the transport that was used. Ultimately, it computes the carbon emissions caused by those journeys and sends an email enclosing user's carbon footprint report, on a weekly basis.

The main downside is that it calculates the carbon emissions caused by journeys from one check-in to another. In other words, if users have to constantly make check-ins on Foursquare such that an accurate carbon footprint can be computed. Furthermore, as mentioned earlier, the application makes a guess about the transport mean used between check-ins. That means that the guess might be wrong in which case the computed carbon emissions value is not accurate and misleading.

2.4.3 Ecobot

Ecobot[13] is a free open-source application that calculates the amount of power, fuel and paper you consume during the course of a day. Users are free from adding nay data to the app. It makes a guess about how frequently you travel by tracking wireless network you access.

[10] http://www.amee.com/
[11] http://alf.ameeapps.com/
[12] https://foursquare.com/
[13] http://ecobot.taxi.ca/

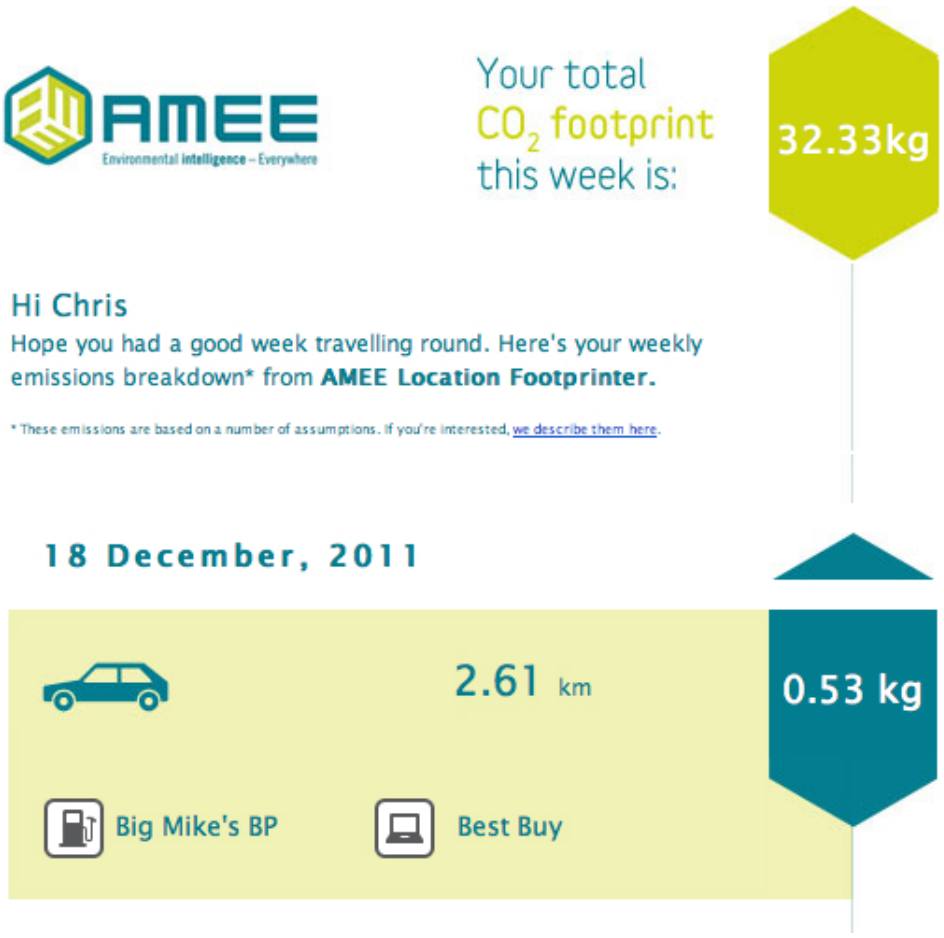

FIGURE 2.18: Carbon footprint report send to user's email

Additionally, it records the amount of papers you use for printing and the energy consumed by several household appliances like computer, video games and televisions.

The application makes several speculations to figure out the activities undertaken by the user. Thus, the total carbon emissions it calculates might be far from the actual amount that user emits. Another disadvantage is that it is currently compatible only with Mac OS.

2.4.4 The Guardian's Quick Carbon Calculator

The official Guardian's web site offer a carbon footprint calculator[14], which computes carbon emissions caused by home activities, travels and shopping habits.

You can add the money you spend for various products and the application calculates a rough estimation of carbon emissions of the corresponding activities. An advantage of that app is that the input data come in the form of money spent; therefore it is easy for users to insert

[14] http://www.guardian.co.uk/environment/interactive/2009/oct/20/guardian-quick-carbon-calculator

FIGURE 2.19: Ecobot carbon footprint calculator

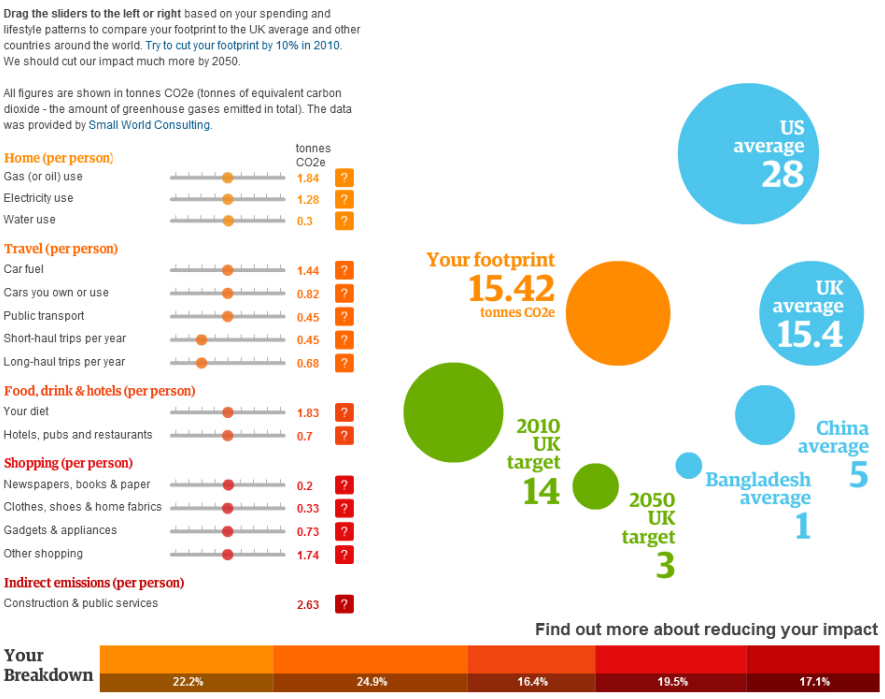

FIGURE 2.20: The Guardian's quick carbon calculator

Name	Developer	Compatibility	Available
CarbonTrack	Logo Design and Marketing Ltd	IOS	YES
Zero Carbon	Mobitelio LTDA	IOS	YES
Clear Standards' Carbon Tracker	Don Frehulfer	IOS	YES
Carbon Diem	Andreas Zachariah	BlackBerry	NO
Ecorio	Ecorio, Inc.	Android	YES
Carbon Footprint Calculator	1WebApps.com	IOS	YES
ecoFootprint	Max Gontar	Android	YES
MyPlanet	Blue Chip Marketing Worldwide	Android	YES

TABLE 2.1: List of applications

accurate values. On the other hand, the computed carbon emissions might not be accurate, due to abstract description of activities added by users; for instance, users specify the cars they own, but they do not clarify the exact model of those cars.

2.4.5 Other Applications

Table 2.1 summarizes several other tools that have been developed for calculating carbon footprints.

2.5 Tools and Libraries

In this section we will present the tools that were user throughout each step of this project. A brief description of each tool will be given accompanied by the rationale for using it.

2.5.1 Tools Used for Literature Research

Table 2.2 summarizes the tools that were used for the literature research phase of the project.

2.5.2 Tools Used During the Application's Design Phase

Table 2.3 summarizes the tools that were used for during the application's design phase of the project.

Tool	Description	Usage	License
Mendeley Desktop	Mendeley is desktop application for managing references and organizing your research.	Mendeley was primarily used for searching and storing scientific papers.	EULA
Evernote	A software for taking notes in different format, such as text notes, web pages, images etc.	Mendeley Evernote was used in two different ways. It was the main tool for keeping notes concerning the project. Additionally, a web clipper plugin was used to capture useful information from various web sites.	Freemium
LaTeX	A markup language for composing different kinds of documents.	Latex was used in conjunction with the WinEdt editor for writing the project report.	LaTeX Project Public License (LPPL)
Adobe Fireworks CS6	Adobe software for processing vector graphics. It is extensively used by web developers for creating web site prototypes.	The software was used for creating and processing the images added to the report and the web application.	Shareware

Table 2.2: List of tools used for literature research

2.5.3 Tools Used During Application's Implementation Phase

Table 2.4 summarizes the tools that were used for during the application's implementation phase of the project.

2.5.4 Frameworks and Libraries used during the application's implementation phase

Table 2.5 summarizes various frameworks and libraries that were used during the application's implementation phase.

Tool	Description	Usage	License

Django	An open-source web application framework for developing web 2.0 applications. The framework is written in Python and follows the Model View Controller architectural pattern.	Django was the framework that used to develop our web application.	BSD License
Ember.js	Ember.js (ex SproutCore) is a relatively new JavaScript framework for applying Model View Controller pattern. Some of the powerful features offered are UI binding (MVVM pattern), computed properties, observable properties, auto-updating templates and many more.	The framework was used to organize the client side code, and apply separation of concerns. Features like UI bindings were utilized for enriching the application's user interactivity.	MIT License
Handelbars.js	A JavaScript template engine for composing updatable and easily manageable html templates.	Handlelbars.js was used in conjunction with ember.js to leverage all the capabilities offered by the framework.	MIT License
Kendo UI (web)	A powerful JavaScript library with lots of capabilities pertaining to web application UI design and rich user interactions.	We used several HTML widgets like calendars, tab menus etc. to enrich the appearance and user interaction of the application.	GPL v3

jQuery	Probably the most popular JavaScript library used today. It extends the core capabilities of the language and simplifies several tasks to minimize the boilerplate code.	jQuery was extensively used in the application to accomplish tasks like, AJAX request, JSON request etc.	MIT license or GNU GPL
Bing Maps API	Bing is widely known web search engine provided by Microsoft. Among others it includes a mapping service called Bing Maps, which can be leveraged by application via an API.	Bing maps API was used to simplify the task for users of inserting addresses.	Various Licensing options
Highcharts	A JavaScript charting library for drawing different kinds of charts (e.g. line, alpine, area charts etc.).	The library was used for drawing the charts that are presented to users on the carbon footprint report page	Free for non-commercial
JavaScript Infovis Toolkit	A powerful visualization library written in JavaScript.	The provenance graphs that are displayed on the carbon emission report page were implemented with the aid of this library.	New BSD License

jQuery.validate	An open-source JavaScript library for manipulating date through JavaScript code.	The library was used for different tasks concerning date manipulation (e.g. converting datetime formats, locating the amount of days passed etc.).	MIT License
jQuery.vTicker	A jQuery plugin for vertical news scrolling.	The library was used to support the news RSS widget on the home page.	Free (details not specified)
jQuery.zrssfeed	A jQuery plugin that reed RSS feeds from any web resource using the Google Feeds API.	The library was used to read RSS feeds from the ECS department of the University of Southampton.	Free
Date.js	A plugin library for jQuery for html form validation	The library was used for validating user inputs (via html forms)	MIT License
Provpy	A python library developed at the university of Southampton for creating and storing provenance information.	All the provenance tasks were achieved via this library.	Free
Matplotlib	A python plotting library capable of creating a big diversity of plot figures	Static provenance graphs in PNG format were produced with the aid of this library.	Matplotlib License

Googlemaps	A python library which supports features like, geo decoding, driving directions, local search and reverse geo decoding via Google maps API.	The library was used to calculate the driving distance between two addresses.	Affero GNU Public License , GNU Library or "Lesser" General Public License version 3.0 (LGPLv3)
Twitter bootstrap	A widely used framework for designing layout for HTML pages and applying pretty CSS styles.	The application's layout and styling was configured with the help of this framework.	Apache License 2.0

TABLE 2.5: List of framework and libraries used during the application's implementation phase

2.6 Summary

This chapter was an intro to the theory of provenance and carbon footprinting. Some core principles of provenance of electronic data were initially presented along with a description of different ways for representing, storing and fetching provenance information. We then introduced some fundamental notions of carbon footprinting. Finally, we discussed about a set of applications which are similar to the one we developed; pros and cons of each applications were given.

Tool	Description	Usage	License
Visual Paradigm for UML 10	Visual paradigm offers an integrated environment for capturing requirements for software development, UML modeling, data modeling and many other features that simplify the design of software.	The tool was used for UML and data modeling.	Proprietary with Free Community Edition
Microsoft Visio	An application for drawing different sort of diagrams. Part of the Microsoft office suite.	Visio was used for drawing the system architecture as well as provenance graphs.	Proprietary commercial software
Balsamiq Mockups	Balsamiq mockups is a tool for quickly creating mock-ups for the application's UI.	The tool was used to draw low-fi sketch wireframes.	Proprietary commercial software
Axure RP Pro 6.5	A prototyping and wireframing tool for designing prototypes for web applications. A feature for generating interactive HTML pages is provided.	Axure was used during the design phase to create a rapid prototype of the applications UI.	Proprietary commercial software
Microsoft Project 6.5	A project management tool provided by Microsoft. The tool is useful for scheduling and managing tasks of large and small projects.	The application was used for scheduling and managing the tasks needed for the project completion, with the aid of Gantt charts.	EULA

Table 2.3: List of tools used during the application's design phase

Tool	Description	Usage	License
PostgreSQL	An object relational database management system (ORDBMS)	PostgreSQL was the used to persist data produced by the web application.	PostgreSQL license
Aptana Studio 3	Aptana studio is an integrated development environment (IDE) used for creating software applications. It is designed for web application development	Aptana studio was the IDE used for developing the web application.	Dual License Aptana Public License, v1.0 GNU General Public License
Fiddler2	Fiddler is a tool used for debugging http requests/responses. It, essentially, acts as a proxy server capturing HTTP traffic.	The tool was used primarily for debugging http GET and Post request made via AJAX calls in the web application.	Freeware (LPPL)
Github	A well-known web-based application using the Git revision control system for tracking changes in the source code of an application.	Github was the revision control system that was used during the application development.	Freemium

Table 2.4: List of tools used during the application's implementation phase

Chapter 3

Application Design

3.1 introduction

In chapter 2 we investigated the theory and technology underpinning the design and developments of our application. In this chapter we investigate the design phase. We will primarily focus on some low level aspects of the design phase by presenting various UML diagrams.

3.2 Application's Design Phase

3.2.1 System Requirements

The application design started by investigating the system requirements. In this section we summarize the initial requirements of the system and provide the use cases derived from those requirements.

Figure 3.1 illustrated an abstract view of the requirements.

Expanding on the diagram depicted in figure 3.1 we can view the following requirements.

Trip Creation : Requirement The system shall allow a student or a member of the staff to add a new trip, by providing the appropriate information, such as, source and destination of the trip, as well as, the transport mean that was used. A user must be authenticated before adding a new trip.

Trip Correction : Requirement The system shall allow a student or a member of the university staff to edit a trip that he has already created. This action demands users to be authenticated.

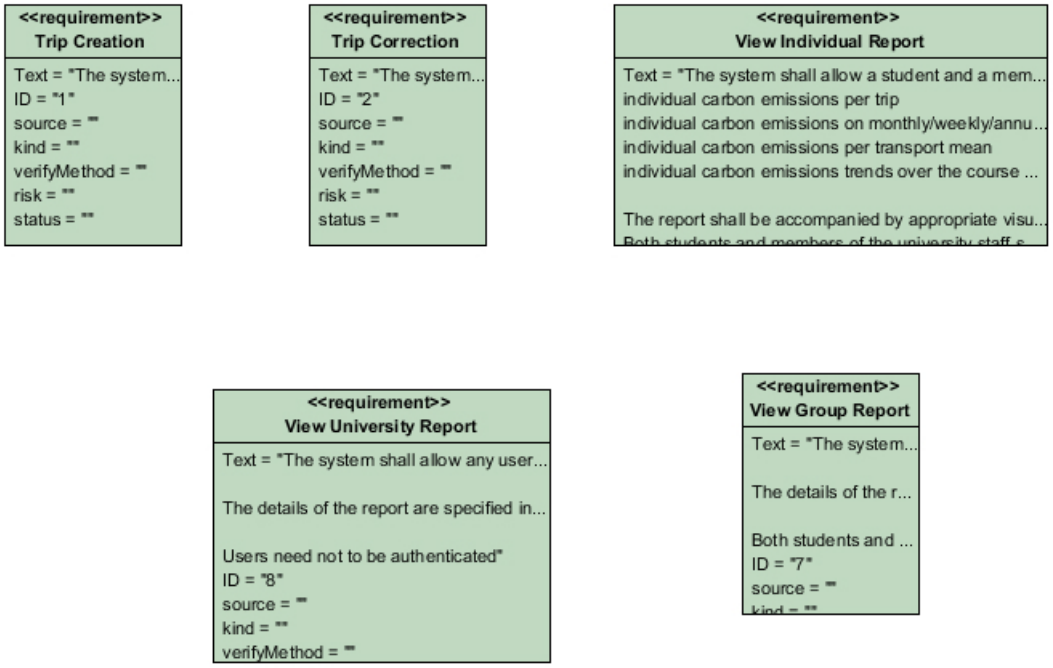

FIGURE 3.1: System requirements diagram

View Individual Report : Requirement The system shall allow a student and a member of the university staff to view his individual report presenting his carbon footprints. The report shall present:

1. individual carbon emissions per trip
2. individual carbon emissions on monthly/weekly/annual basis
3. individual carbon emissions per transport mean
4. individual carbon emissions trends over the course of time

The report shall be accompanied by appropriate visualization elements (e.g. charts etc.).

View Group Report : Requirement The system should allow a student and a member of the university staff to view carbon emission report, concerning the groups that they are member of (e.g. research groups etc.). Both students and members of the university have to be authenticated.

View University Report : Requirement The system shall allow any user to view a summary report about the carbon emissions of the entire university. Users need not to be authenticated.

3.2.2 Use Cases

Finding the requirements was the first step. The next step is to fing use cases of the application. Figure 3.2 illustrates a diagram with the system's use cases. A description of each use case is summarized in table **??** further below along with the corresponding flow of events.

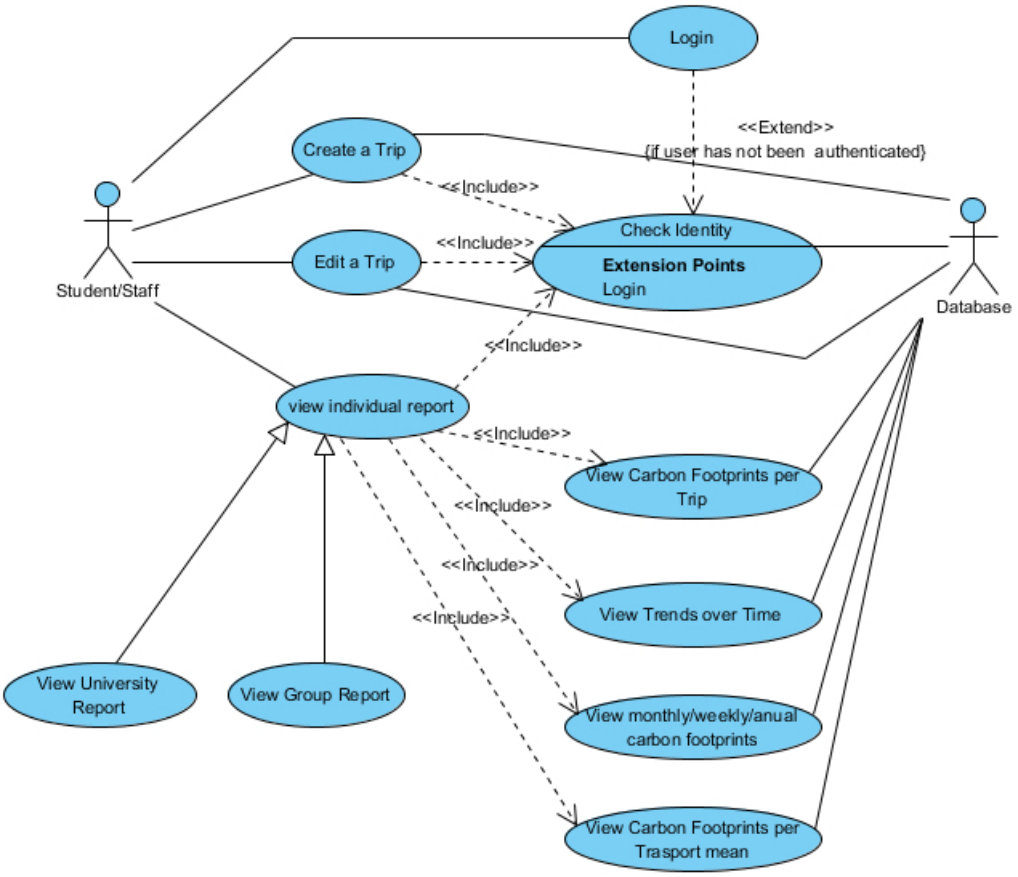

FIGURE 3.2: Use cases diagram

Student/Staff : Actor Students or members of university staff.

Database : Actor The persistent storage where all the data are stored.

Create a Trip : Use Case A use case where authenticated users can add trips they made, by providing the appropriate details.

Check Identity : Use Case A process that assures that specific operations can only be executed by authenticated users.

Edit a Trip : Use Case A use case indicating that an authenticated user (i.e. a student or a member of the staff) can modify the details of a trip that she has added to the system.

Login : Use Case A use case where users provide their credentials in order to enter the system and are able to execute operations that require them to be authenticated (e.g. trip creation).

view individual report : Use Case A use case where authenticated users (i.e. students or members of the university staff) can view their individual carbon emissions report trough a rich web UI.

View Carbon Footprints per Trip : Use Case A use case where authenticated users (e.g. students and members of university staff) can view the carbon emitted by each of the trips they have made.

View monthly/weekly/anual carbon footprints : Use Case A use case where authenticated users (e.g. students and members of university staff) can view the carbon emissions caused by their travel activities, on weekly, daily or annual basis.

View Trends over Time : Use Case A use case where authenticated users (e.g. students and members of university staff) can view their carbon emissions over the course of time, through appropriate web UI widgets (e.g charts).

View Carbon Footprints per Trasport mean : Use Case A use case where authenticated users (e.g. students and members of university staff) can view the carbon emissions categorized by the transport means that were used.

View Group Report : Use Case A use case where authenticated users can view the carbon emissions report for the groups that they are members of.

View University Report : Use Case A use case where any user can view carbon emissions report about a university.

3.2.2.1 Flow of Events

Each use case comprise a number of actions that needs to be executed. We therefore, strived to locate those steps way before the implementation part was started. We provide the flow of events for our use cases, in the following lines.

Create a Trip : Use Case

Steps

1. `The user asks the system to create a new trip`

2. Check Identity
3. The system asks the user to add the source and destination of the trip, as well as, the transport mean that was used. For each of those fields, the system might let the user to choose from some pre-defined values (e.g. common addresses and transport means) that had been configured by the user.
4. The user provides all the required information and submits the form.
    4.1. The user adds the source and destination address of the trip
        4.1.1. if The address exist in the system
            4.1.1.1. if the user wants to select a private address
                    (i.e. addresses had bee added by the user in the past)
                4.1.1.1.1. The user selects an address from the list of private addresses
                4.1.1.1.2. The system shows the address on a map
                4.1.1.1.3. The user confirms the address or picks a different
            4.1.1.2. else
                4.1.1.2.1. The user selects an address from the list of public addresses
                4.1.1.2.2. The system shows the address on a map
                4.1.1.2.3. The user confirms the address or picks a different
                 end if
        4.1.2. else
            4.1.2.1. The users chooses to add a new address
            4.1.2.2. The system shows the address creation form
            4.1.2.3. The user enters the address
            4.1.2.4. The user specifies the visibility of the address
                    (i.e. private or public)
            4.1.2.5. The user click the add address button
            4.1.2.6. The system stores the address into the persistent storage
           end if
    4.2. The user adds the transport mean
        4.2.1. The user selects the type of the transport mean
              (e.g. car, bus, taxi etc.)
        4.2.2. if the user selected the car option
            4.2.2.1. if the user wants to select one of the cars he stored into the system
                4.2.2.1.1. the system show all the cars that the user stored in the system
                4.2.2.1.2. the user selects a car

```
            4.2.2.2. else
                4.2.2.2.1. The system show a car creation form
                4.2.2.2.2. The user adds some the information about the car
                           (e.g. engine capacity,name etc. )
                4.2.2.2.3. The system add the car to the list of cars that have
                           been used by the user
                     end if
        4.2.3. else
            4.2.3.1. The user selects the specific type of transport mean
               end if
5. The user add the date and time of the trip
6. A new trip is created and inserted into the persistent storage.
```

Edit a Trip : Use Case

Steps

```
1. The user asks the system to edit a trip
2. Check Identity
3. The system retrieves from the persistent storage, the the trips that
   has been added by the user. Trips should be easily searchable, so that
   users can easily locate the one they want to alter.
4. The user selects the trip she wants to modify
5. The system presents a form populated with the details about the
   selected trip.
6. The user modifies the trip details and submits the form
7. The modified trip record is stored back to the persistent storage
```

Login : Use Case

Steps

```
1. while User is not authenticated
    1.1. The systems asks the user to provide with her credentials
    1.2. The user enters her credential
    1.3. The system checks if the given data exist in the database
    1.4. The system authenticates the user
   end while
```

Extension

```
1.3.a. Authentication Failure
    1. The system cannot find user's credential in the database
    2. An error message is displayed
    3. jump to 1.1. The systems asks the user to provide with her credentials
```

Check Identity : Use Case

Steps

```
1. The system checks if the given user has already been authenticated
2. The system allow the user to execute the operations that need
   authentication
```

Extension

```
1.a. Login
```

View individual report : Use Case

Steps

```
1. The user asks the system to view her individual carbon emissions report
2. Check Identity
3. The system presents the individual report for the user
    3.1. View Carbon Footprints per Trip View monthly/weekly/anual carbon
    footprints View Trends over Time View Carbon Footprints per Trasport mean
```

3.2.2.2 Activity Diagrams

The flow of events for each use case that we oulined earlier, can be better illustrated with the aid of UML Activity Diagrams that follows.

3.2.3 Class Diagrams

The code of the application both on back-end and front-end was written in object oriented languages (though javascript is not pure object oriented language); therefore, what follows are class diagrams for the back and front-end.

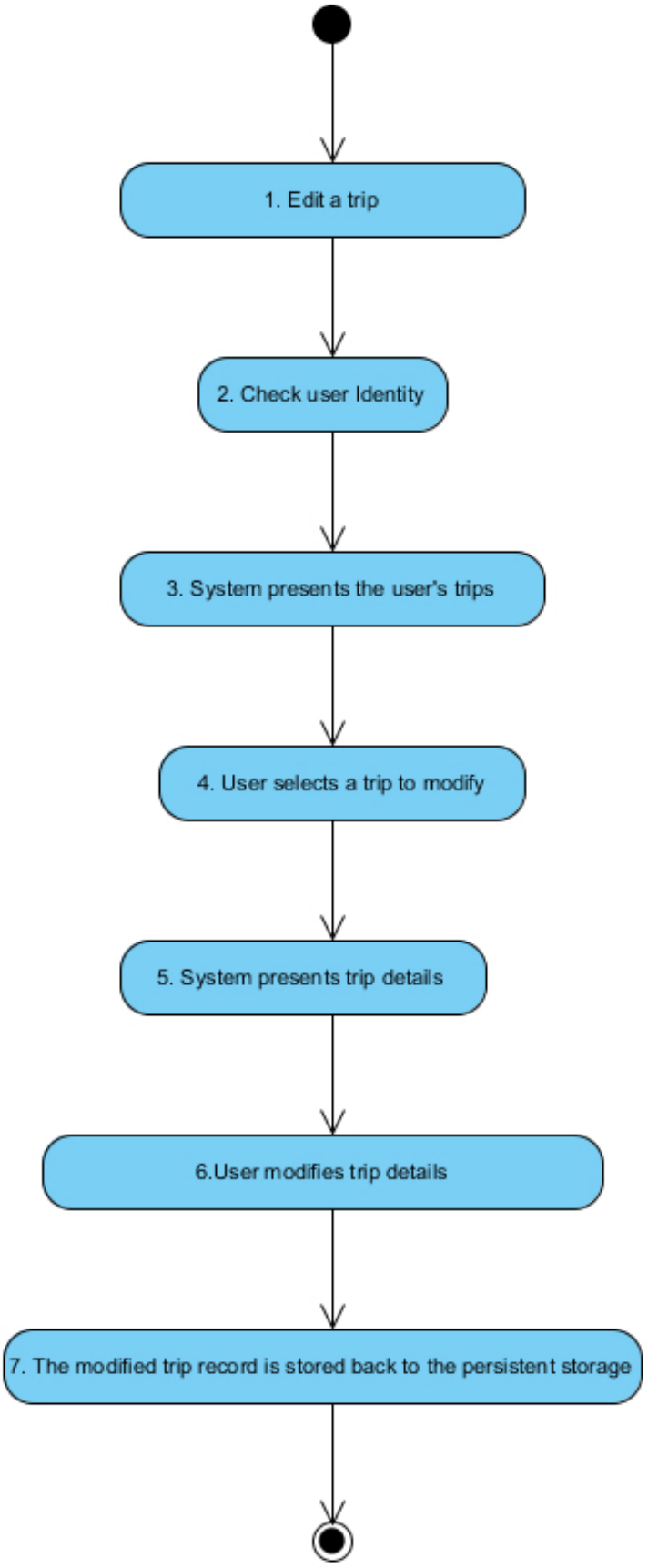

FIGURE 3.3: Edit trip activity diagram

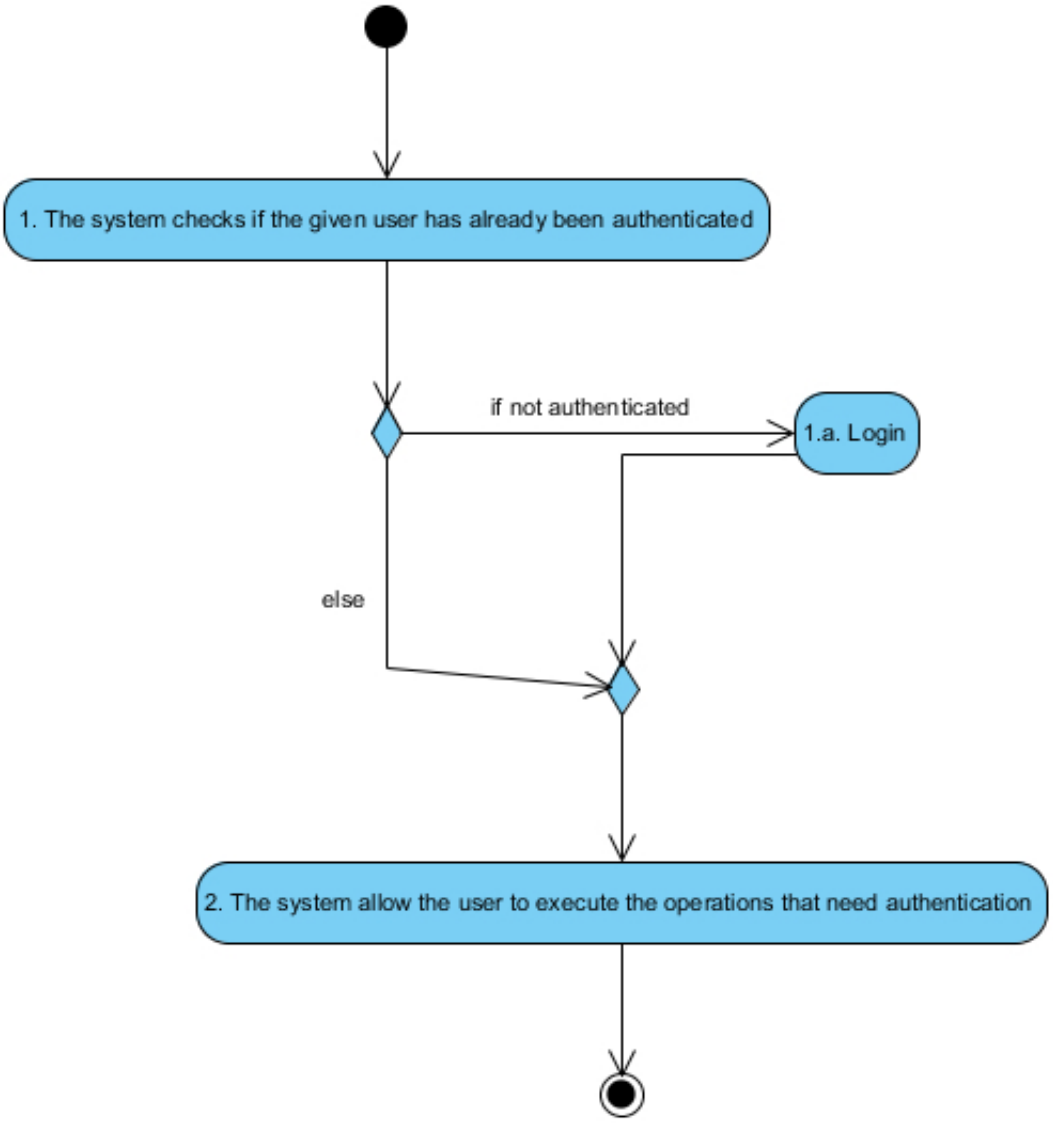

FIGURE 3.4: Check identity activity diagram

3.2.3.1 Back End

Figure 3.6 shows the back end class diagram and table 3.1 summarizes brief descriptions for each class.

3.2.3.2 Front-End

Our application has several tasks that are executed on client side. More specifically, the application employs the MVC using the Ember.js framework. Thus, the client-side code consists of the core constituents of the MVC pattern that is, models, views and controllers. Views are the classes which users interact with via the corresponding html template, whereas models are the

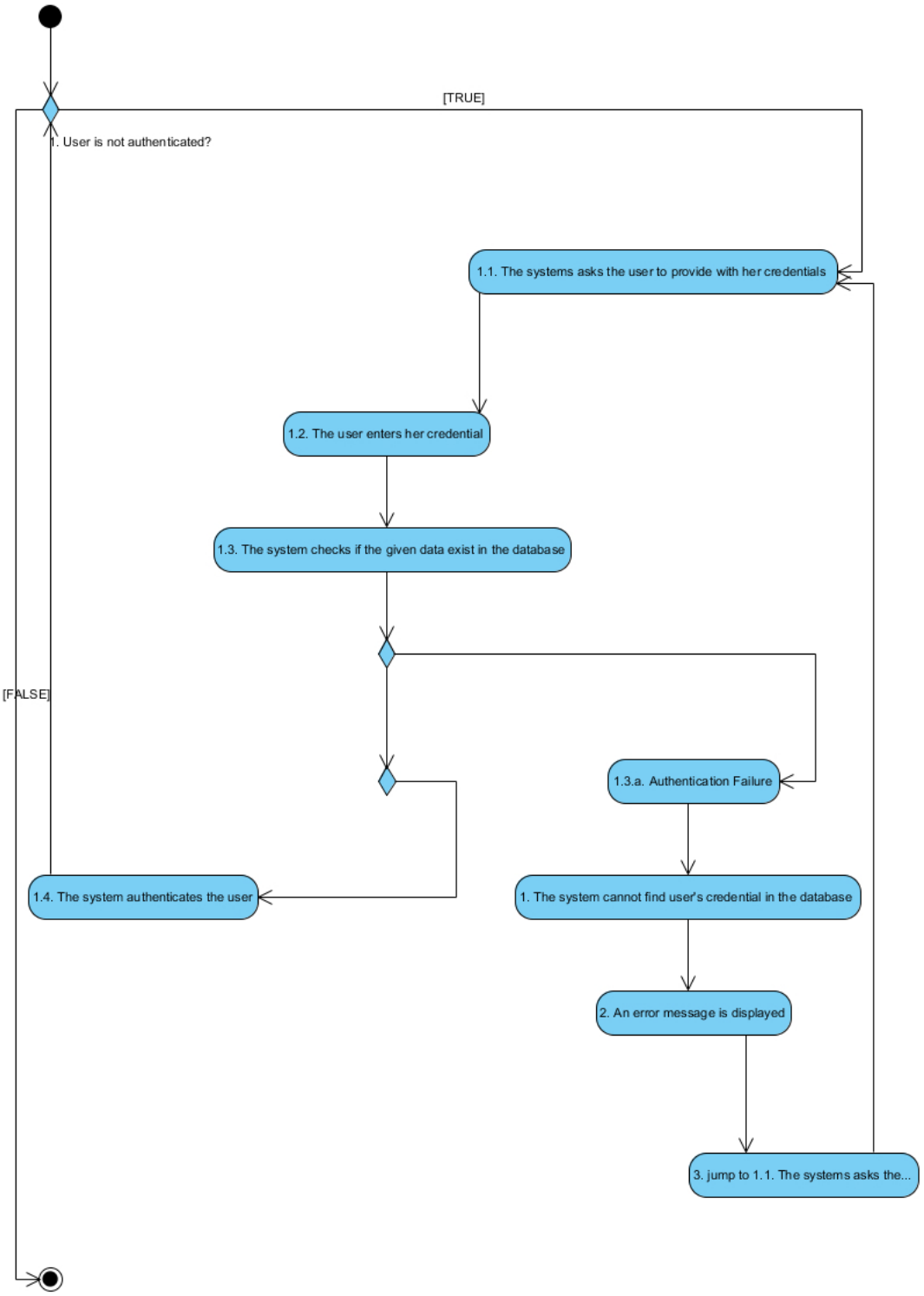

FIGURE 3.5: Login activity diagram

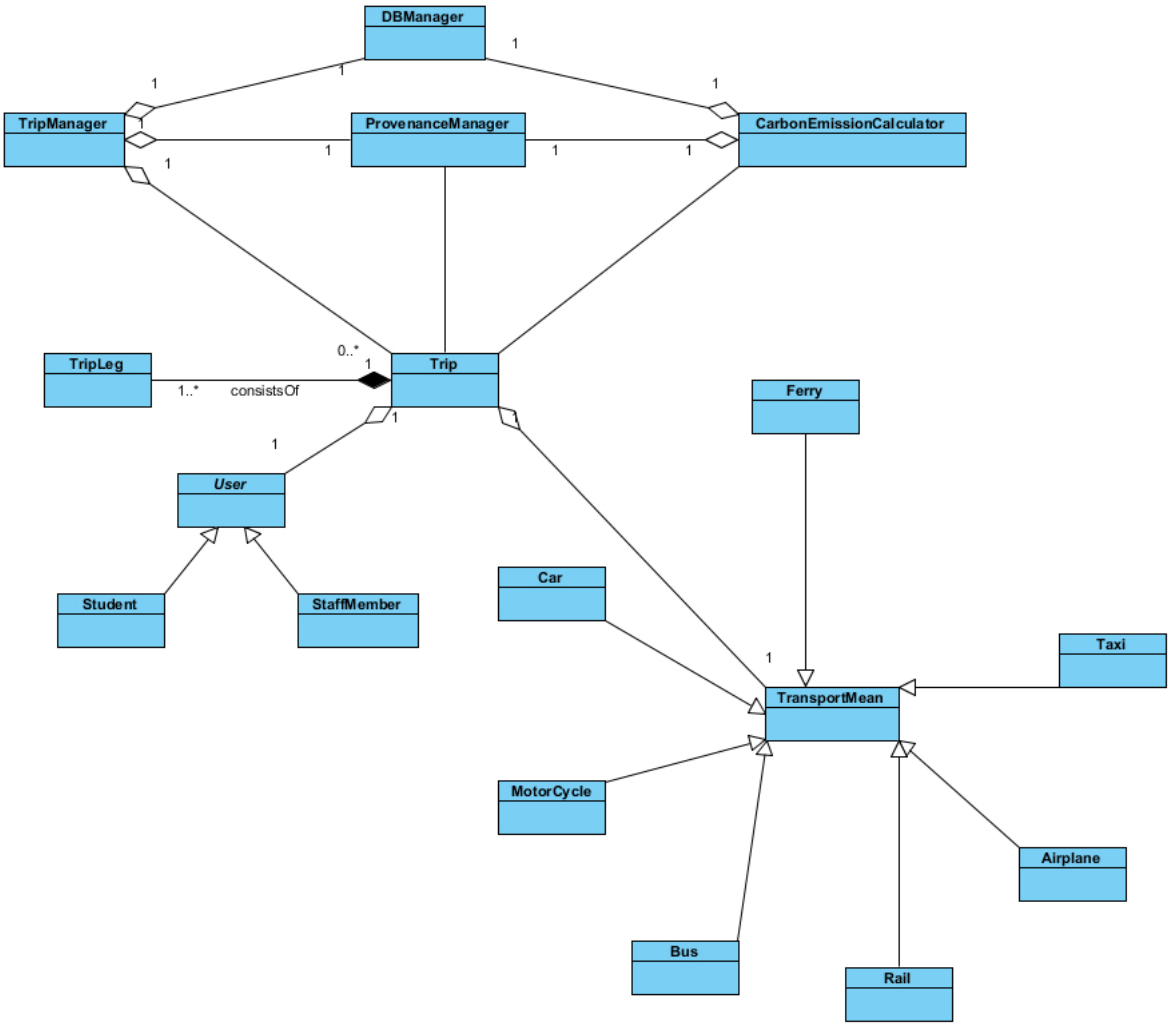

FIGURE 3.6: Back-end class diagram

actual data that the application holds. Finally, controllers are the connections between views and models.

We will separate the front-end class diagram into three figures. The first diagram (figure 3.7) presents the classes that participate in trip management tasks, such as trip creation. The second diagram (figure 3.8) comprises classes that participate in data visualization-related tasks, such as chart and graph presentation. Finally, the third diagram (figure 3.9) illustrates the classes that take part in the user trip presentation task.

3.2.4 Database Schema

The data model for our ampliation was probably the most important part, after capturing the requirements and forming the use cases. The application stores all data in a relational database which consists of 48 relations. More specifically, there are three sets of tables for different

Class	Description
Trip	The main enity of the application. It represents the trip that users make.
TripLeg	Each trip can have several intermediary step. Trip legs represents those steps
TransportMean	The class represents transport means that are used during user's trips. It is an abstract class which has concrete subclasses (i.e Car, Motorcycle etc.)
User	instances of this class refer to the user of our application.
TripManager	A class responsible for all the tasks concerning user trips management, such as trip addition, trip modification etc.
ProvenanceManager	A class responsible for provenance-related tasks, such as creation, storing etc. of provenance graphs.
CarbonEmissionCalculator	A class responsible for calculating the carbon emissions of trips, trip legs etc.
DBManager	A class responsible for database-related tasks. To be accurate, this is an ORM tool provided by the Django framework for supporting object relation mapping.

TABLE 3.1: Back-end classes

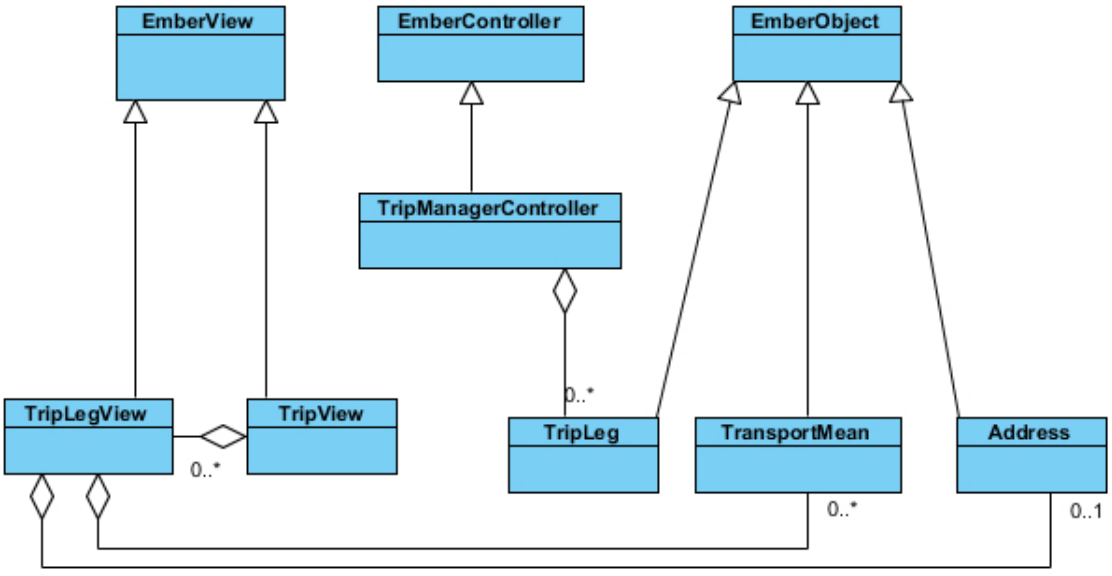

FIGURE 3.7: Front-end class diagram comprising classes used in trip management tasks

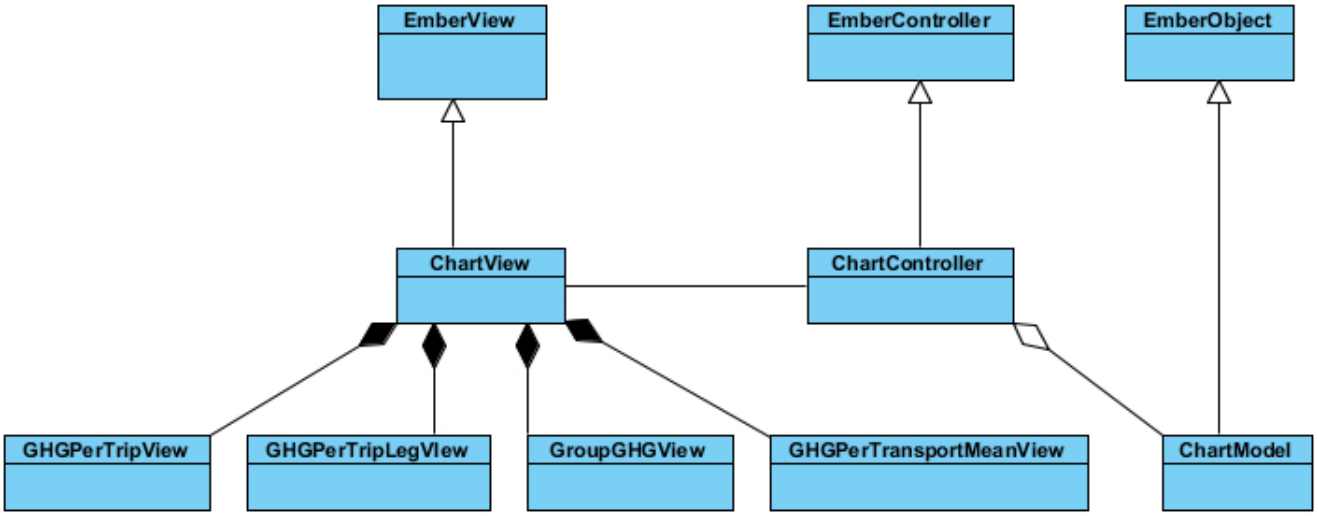

FIGURE 3.8: Front-end class diagram comprising classes used data visualization tasks

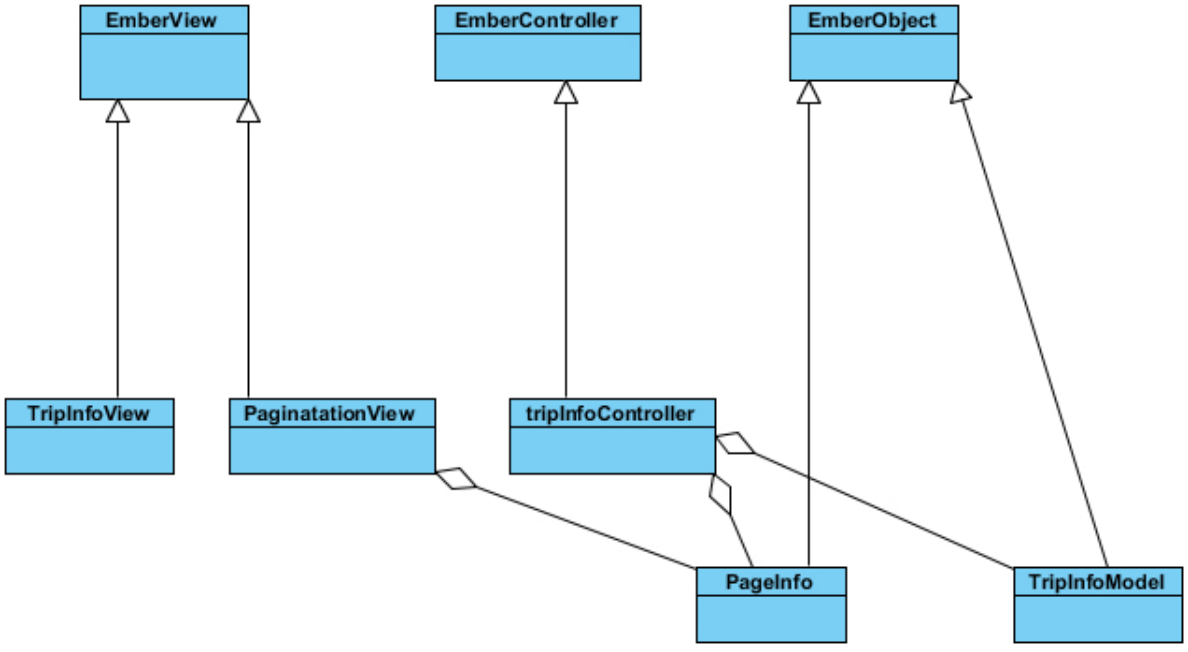

FIGURE 3.9: Front-end class diagram comprising classes used in user trips presentation task

Class	Description
EmberView	A class which represents views as defined by the Ember[1] library (Ember.View).
EmberController	A class which represents controllers as defined by the Ember library (Ember.Controller).
EmberObject	A class which represents models as defined by the Ember library (Ember.Object).
TripManagerController	A controller responsible for gathering trip informations and sending them to the back end via AJAX Post call.
TripView	A view that has an html template (i.e. html form elements) for adding information about trips.
TripLegView	A view that has an html template (i.e. html form elements) for adding information about trip legs (e.g. source address, transport mena etc.). It is a sub view of the TripVew view.
TripLeg	A model holding information about trip legs.
TransportMean	A model that holds information about transport means.
Address	A model that holds information about geographical addresse.
ChartView	A view that has an html template for displaying several charts. Those charts are hosted within GHGPerTripView, GHGPerTripLegView, GroupGHGView, GHGPerTransportMeanView views.
ChartModel	A model containing appropriate data that are passed as input to a JavaScript charting library.
TripInfoView	A view that has an html template for displaying the trips that a used has done.
PaginationView	A view containing a paginator[2], which helps to navigate through a list of trips.
tripInfoController	A controller that fetched information about trip from the back-end via AJAX GET requests.
TripInfoModel	A model containing information about trips.
PageInfo	A model containing information that are required by the paginator.

Table 3.2: Front-end classes

reasons. There is a set of tables supporting the authentication and authorization mechanism; another set used for storing provenance information; and finally a third set of tables storing the rest data produced by the application (e.g. user trips, carbon emissions, transport means etc.). Figure 3.10 presents the database schema including tables of the third set. A brief description for each table is summarizes in table 3.3. Note that the database schema is missing several other tables which share similar features, such as all the other modes of transports apart from cars and their corresponding carbon emissions.

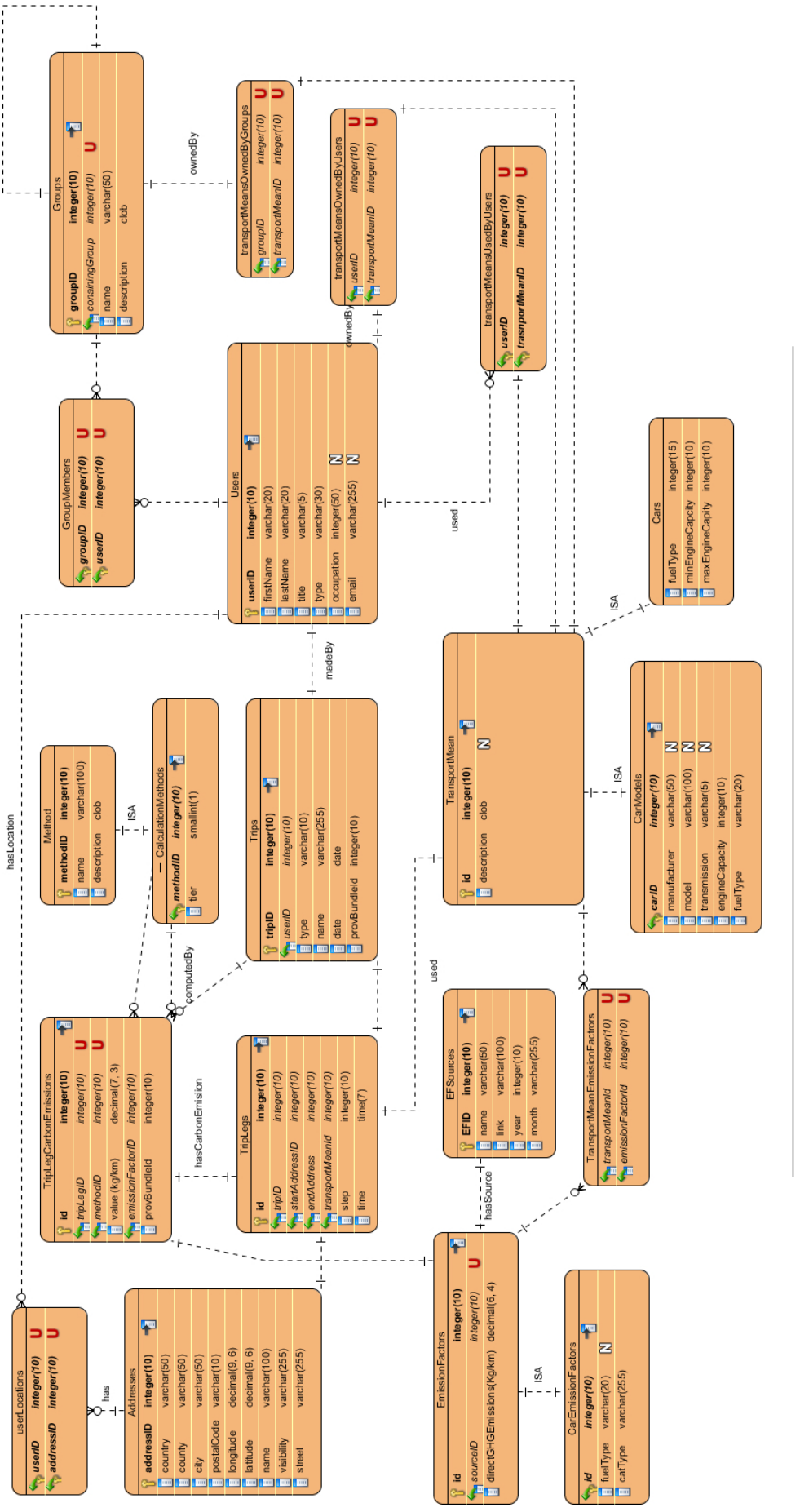

FIGURE 3.10: Database schema

3.3 Summary

In this chapter we went through the application's design phase. To make the description more obvious, various types of UML diagrams were given illustrating the path we followed towards implementing the application.

Relation	Description
Trips	Table containing the trips made by students and the members of the university staff.
TripLegs	Table storing the intermediary steps of a trip.
TripLegCarbonEmissions	Table containing the calculated value of carbon emissions caused by trip legs.
Method	Table containing description about different sort of methods employed by the application. For example, an extrapolation method for calculating the annual carbon emissions for an individual, based on partial set of trips that were added.
CalculationMethos	Special case of the Method relation. It contains various methods for calculating carbon emissions.
UserLocations	A table containing the locations that user has visited.
Addresses	Table containing details about geographical addresses.
TransportMean	A abstract relation that has concrete relations storing different modes of transports. CarsModels and Cars and is an example of such concrete relation.
EmissionFactors	A abstract relation that has concrete relations storing emission factors for different modes of transports. CarEmissionFactors is an example of such concrete relation
TransportMeanEmissionFactors	A many-to-many relation between emission factors and transport means. Each transport mean can have different emission factors based in the source that provides them.
EFSource	Table containing the sources of emission factors (e.g. IPCC etc.)
Users	Table containing the users of our application.
Groups	Any group that contains a number of people, such as a research group, an academic unit or the entire university.
GroupMembers	Table containing the members (users) of each group.
TransportMeansUsedByUsers	Table contains the transport that users have used.
TransportMeansOwnedByUsers	Table contains transport means that users own.
TransportMeansOwnedByGroups	Table contains transport means owned by groups.

Table 3.3: Database tables

Chapter 4

Provenance in the Application's Context

4.1 introduction

In the previous section we have extensively covered the concept of provenance of electronic data. We went through the benefits of capturing provenance information. We then presented different data models for representing, as well as, locating and retrieving that information. The purpose of this chapter is to describe all those concepts, in the scope of the web application that we developed. We will start by reviewing the benefits that provenance brought to our application. Then the data for which provenance is tracked, will be presented. Finally, we briefly describe how the application stores and exposes that information to external applications.

4.2 Benefits of Provenance

Our web application is a provenance-aware crowd sourced system for monitoring GHG emissions caused by commuter travels. In essence, users are the main source of information; they insert some data pertaining to the travels they make and the application calculates the GHG emissions they produce. It is critical for users to be able to verify the correctness of such calculations. To make that possible, the application needs to track all the steps in the carbon emissions calculation process: the module responsible for calculating carbon emissions has to compile all emissions sources for the subject (in the context of this application, emission sources refer to the trip made by users). Then, an appropriate method for quantifying the emissions of each source has to be selected. Finally, the data required by the method should be gathered, so that carbon emissions for the trip in question can be computed. In order to verify and validate the

computed figure, one should be able to view details for each of those steps; for instance, consider that the total amount of carbon emissions of an individual needs to be verified. In that case, the emission sources that were used have to be presented. Further, the calculation methods, as well as, the data used (i.e. activity data and emission factors), need to be examined. It is obvious that all the aforementioned data are already stored in the applications' persistence storage (i.e. database). However, what is missing are the connections (dependencies) between those data. This gap is filled by implementing a software component responsible for provenance related tasks. What follows is a descriptions of those tasks.

4.3 Tracking Provenance

In this section we will explain how our application captures, stores and exposes provenance information. We start by presenting the provenance graphs that represent that information.

4.3.1 Trip Creation Process

One of the chief functionalities provided by the application is the insertion of trips. Users, add with the aid of an interactive web UI (refer to chapter 5 for more details) the trips that they make. The provenance manager, which is the system's component responsible for provenance-related tasks, creates a provenance graph during the process of creating new trips (the "trip creation" process). Figure 4.1, illustrates a visualized form of that graph, which reveals the provenance of a trip made by the user.

The application adopts the PROV-DM model to represent provenance information. For this reason the graph in figure 4.1 consists of entities, activities and agents, as well as, the relationships among them.

The trip entity refers to the trip that was created during the "trip creation" process. Each trip might consist of several intermediary steps, which we call "trip legs". This is the reason why the *wasDerivedFrom* property connects the trip entity with the corresponding trip legs entities.

Each trip and trip leg is created (*wasGeneratedBy* edge) via the "trip creation" process, which corresponds to the "TripCreation" activity. In some cases it is required to know the computational time of that activity. For this reason the provenance graph includes two additional nodes, storing the start and end time of that activity. Finally users are responsible for creating new trip. Hence, an agent node is added denoting that users has a degree of responsibility for the "Trip Creation" activity.

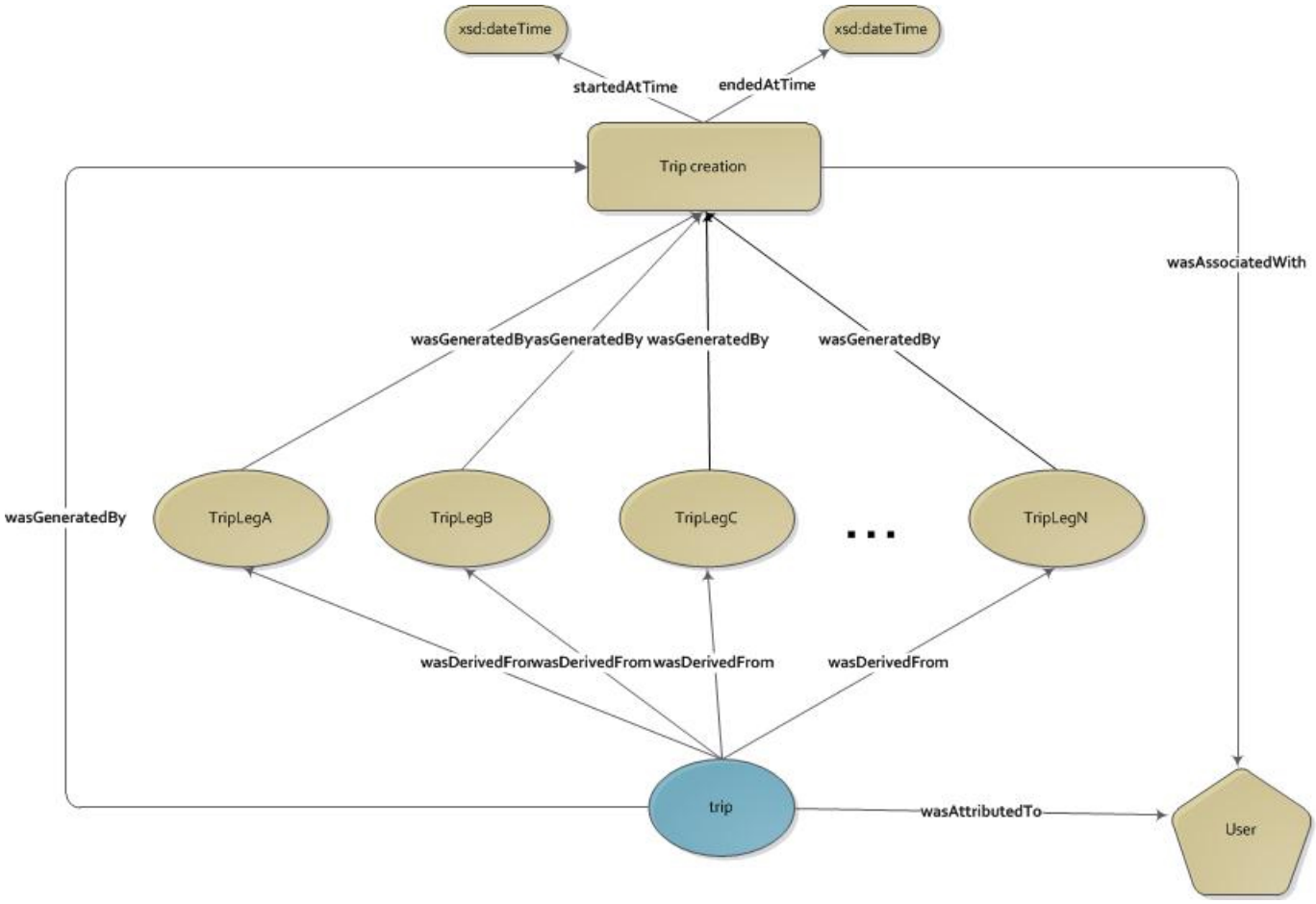

FIGURE 4.1: Trip creation provenance graph.

4.3.2 Trip Leg Calculation Process

During the "Trip creation" process, the system initiates the "Trip Leg Calculation" process. This process, calculates the carbon emissions for each trip leg, based on the data provided by the user. It is the responsibility of this process to select the activity data and an appropriate emission factor (based on the mode of transport used) that will be used in the calculation formula.

The provenance graph that is created during this phase is illustrated in figure 4.2.

The "TripLegCarbonEmissionsCalculation" activity, which is initiated by the "CarbonEMissionCalculation" agent, corresponds to the process of computing the carbon emissions for a specific trip leg. The calculation process needs some sort of activity data and appropriate emissions factor, based on the mode of transport used. Ultimately, those data are combined via a calculation method to produce the carbon emissions for that trip leg. The relationship is expressed via the used edges from the "TripLegCarbonEmissionsCalculation".

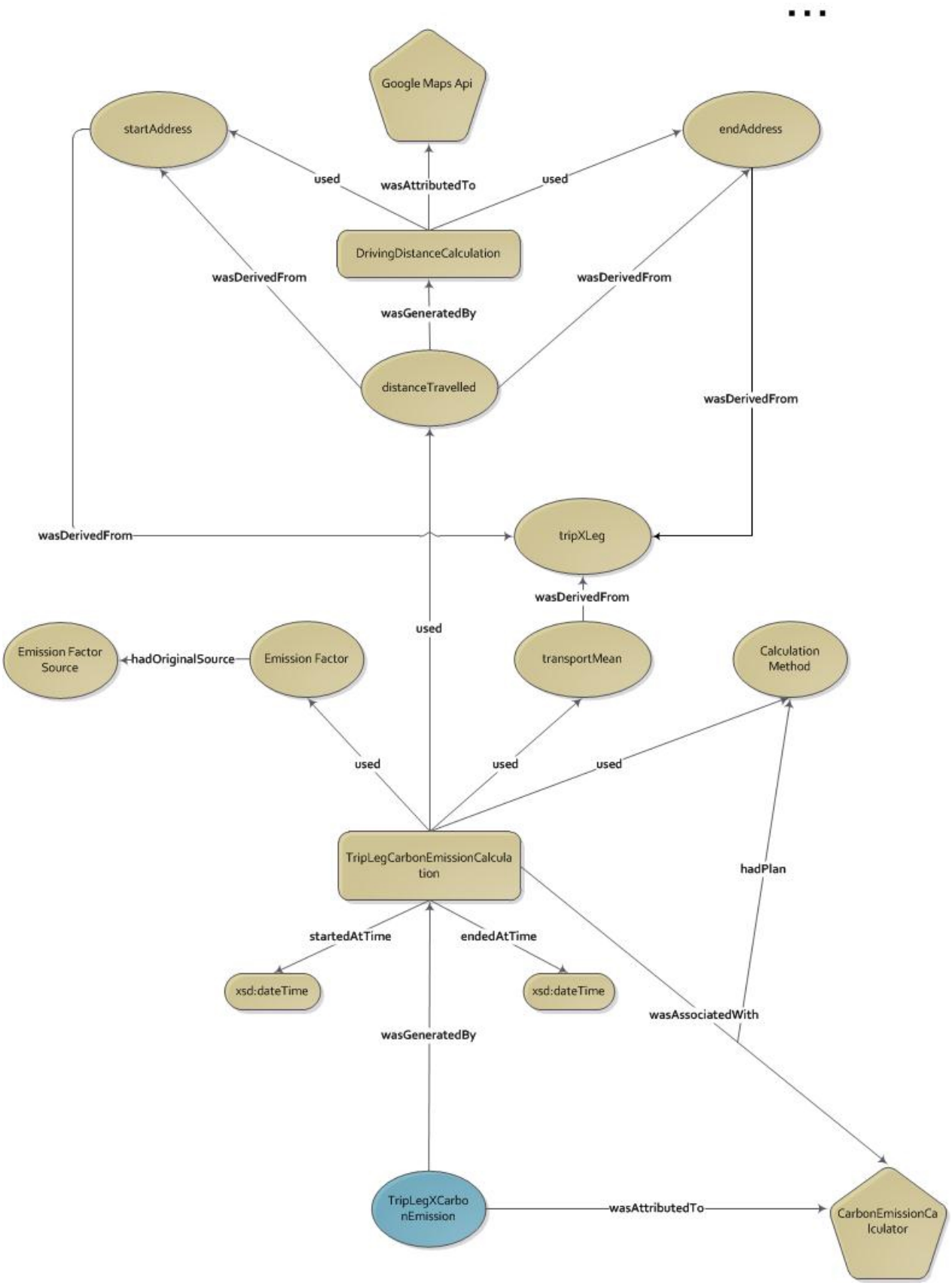

FIGURE 4.2: Graph illustrating the provenance of the TripXLegCarbonEmission entity.

The "distanceTravelled" entity represents the driving distance travelled by the transport mean. This figure is computed dynamically through a service (DrivingDistanceCalculation) provided by a third party application (GoogleMaps API). The input data for this service are the source and destination addresses of the trip. Finally, there can be several organizations that publish emission factors; we need to explicitly specify the source of the emission factor. For this reason, the *hadPrimarySource* edge from "emissionsFactor" to "emissionFactor Source" is added.

4.3.3 Trip Carbon Emissions Calculation Process

The process of calculating the total carbon emissions caused by a trip is relatively straightforward. It simply sums together the carbon emissions caused by all the intermediary trip legs.

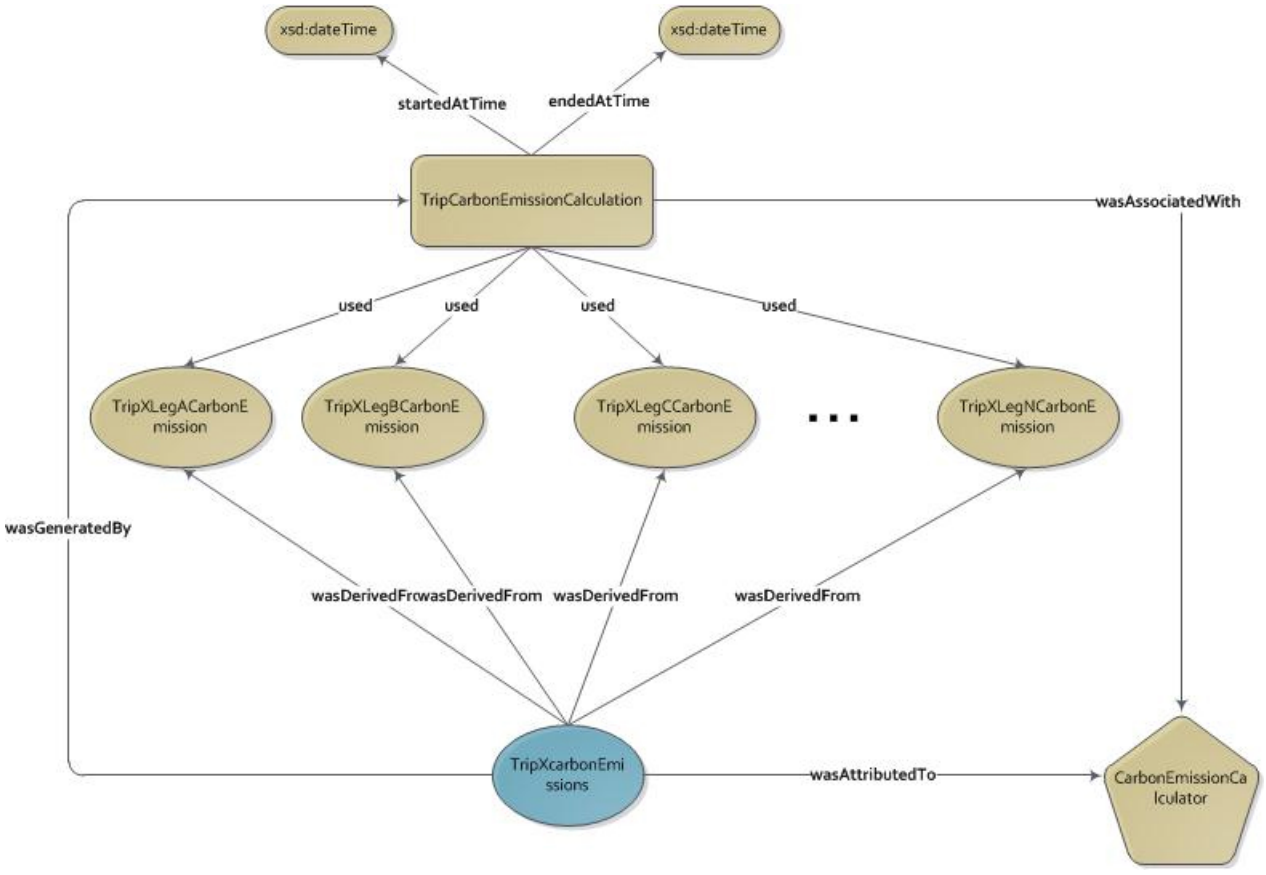

FIGURE 4.3: Graph illustrating the provenance of the carbon emissions of a trip

Figure 4.3 demonstrates a graph that describes the provenance the carbon emissions caused by a trip, that is, how that value was derived. The calculation of carbon emissions is processed by the "TripCarbonEmissionCalculation" activity, which sums the the carbon emissions of all the intermediary trip legs to compute that total emissions for the trip (TripXCarbonEMissions). The

dependency is represented via *wasDerivedFrom* edges. Finally, the "CarbonEmisisonCalculator" agent is responsible for undertaking the whole process.

This is good point to demonstrate an example describing how a provenance graph can help users validate their carbon emissions. Consider a scenario where the user modifies the data for one trip leg of a hypothetical trip (TripX). Assume that she changed the transport mean used during that trip leg. At some point, though, she realizes that there is a significant fluctuation between the old and the new value. More specifically, the new carbon emissions value for TripX is much higher than the previous one. In order to verify that there was no error during the calculation process; user can navigate through the provenance of that value (see figure 4.4).

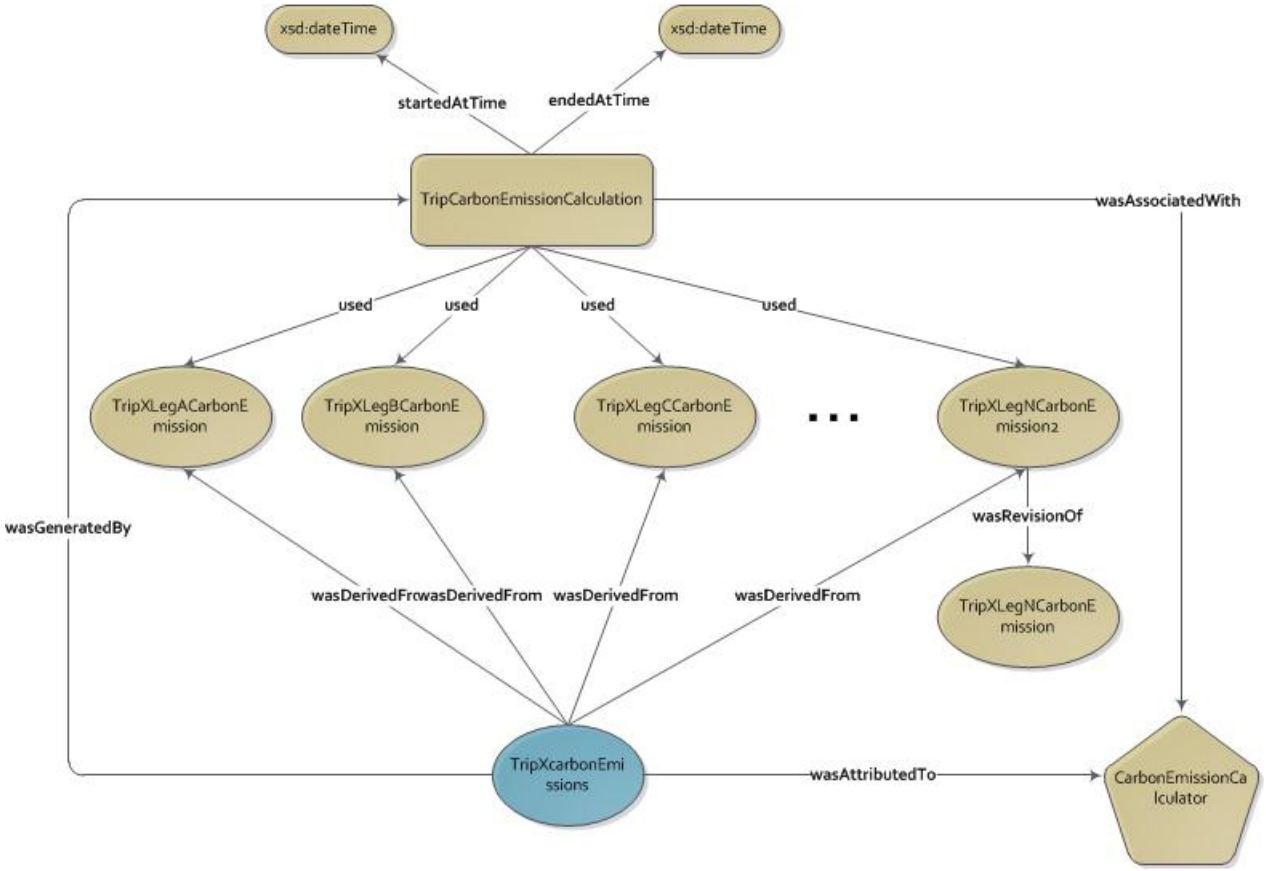

FIGURE 4.4: Graph illustrating the provenance of the carbon emissions of a trip after a trip leg was modified

The graph reveals that the "TripXLegNCarbonEmission" entity has been modified and replaced by the "TripXLegNCarbonEmission2" entity. Ultimately, by investigating the provenance of the new entity (the provenance graph is similar to the one illustrated in figure 4.2), she realizes that she has accidentally inserted the wrong transport mean.

The same approach can be followed when the validation for values for which provenance information is stored, is needed.

4.3.4 Calculation process for individual and group carbon emissions

The application gives a chance for users to view a report where details about individual carbon emissions are presented. One of the information included in that report is the total carbon emissions caused by all the trips made by the user. To achieve that, the carbon emissions for all trips are summed together.

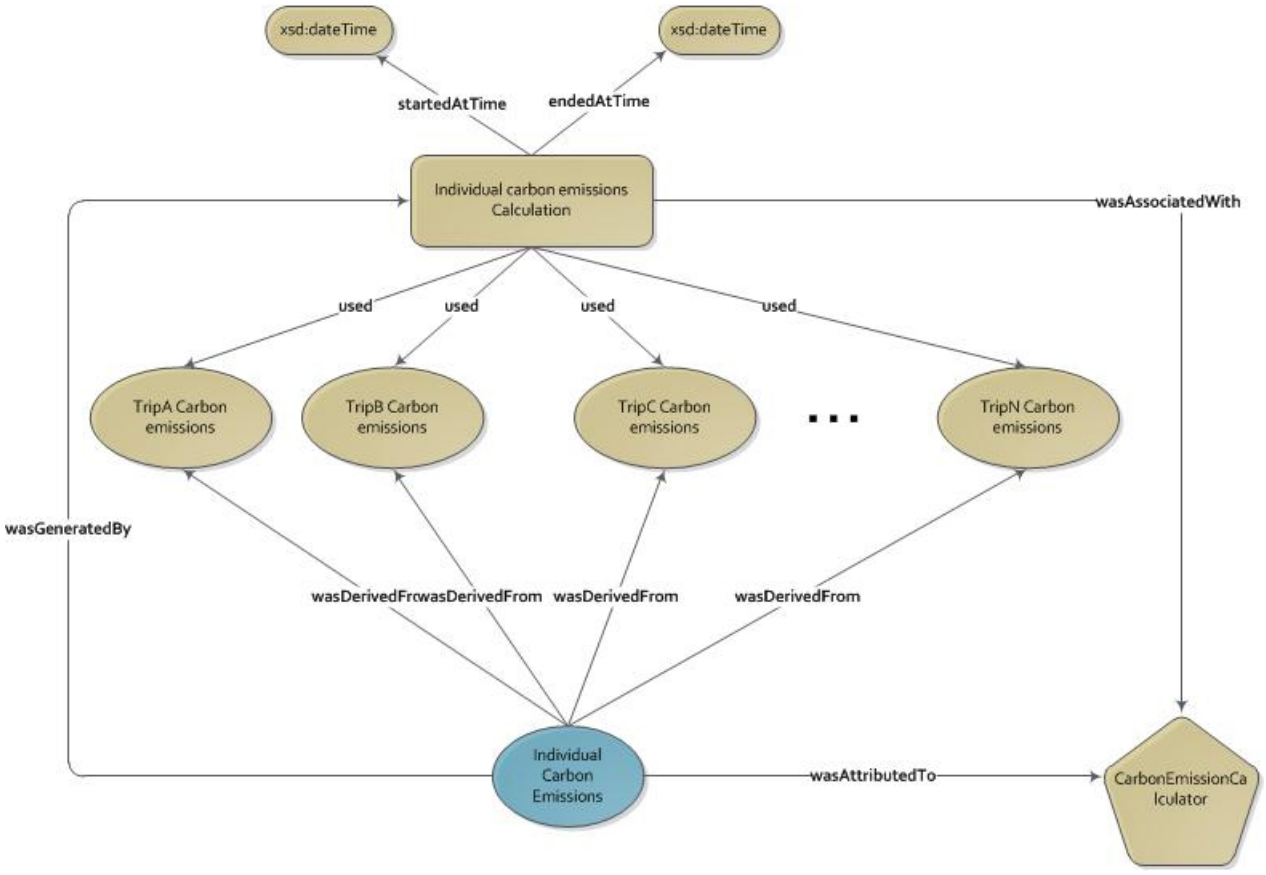

FIGURE 4.5: Graph illustrating the provenance for individual carbon emissions

The same approach is adopted by the "Group Calculation" process, whereby the carbon emissions caused by all the trips made by all the members of the group in question are summed together. Figure 4.6 illustrates the provenance graph for group carbon emissions.

4.4 Storing Provenance

The provenance graphs that we discussed earlier are represented in some application-specific format and stored in a relational database. More details about the process of creating and storing provenance graphs are presented in the next chapter.

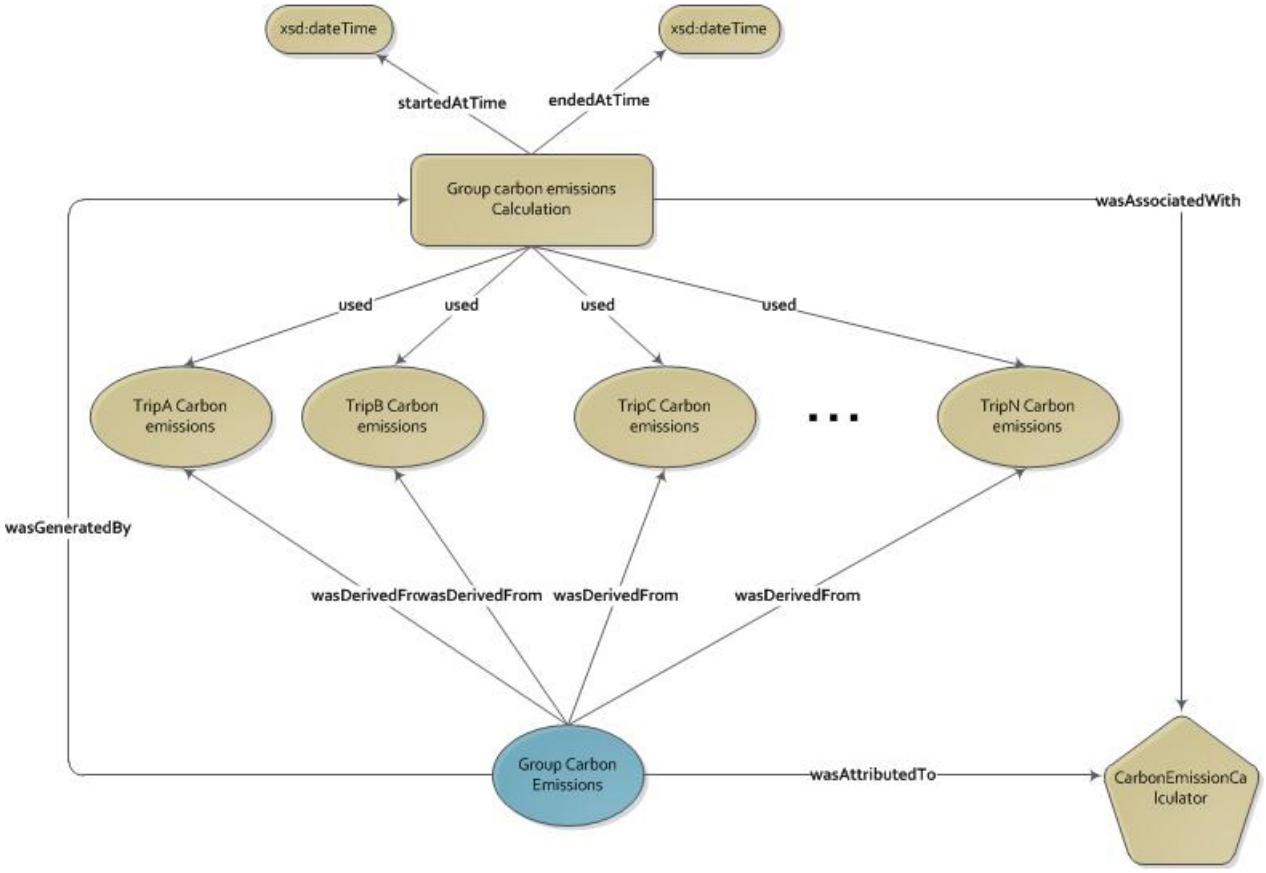

FIGURE 4.6: Graph illustrating the provenance for group carbon emissions

4.5 Fetching Provenance Information

Our application has two options for viewing the provenance information for a specific piece of data; for instance, if users what to view provenance information in the form of graphs (like the above graphs), then a specific user interface can be used (next chapter describes this interface). Additionally, the application provides a REST web service that gets the identifier of a provenance graph and returns the graph in JSON format.

The JSON data below was returned after accessing the following service: http://127.0.0.1:8000/get-prov-graph/?provBundleId=1


```
{
   "wasDerivedFrom":{
      "_:id4":{
         "prov:usedEntity":"cf:tripLeg-2",
         "prov:generatedEntity":"cf:trip-1"
      },
```

```
      "_:id2":{
         "prov:usedEntity":"cf:tripLeg-1",
         "prov:generatedEntity":"cf:trip-1"
      }
   },
   "wasAssociatedWith":{
      "_:id6":{
         "prov:agent":"cf:ppoliani",
         "prov:activity":"cf:tripCreation"
      }
   },
   "wasAttributedTo":{
      "_:id7":{
         "prov:entity":"cf:trip-1",
         "prov:agent":"cf:ppoliani"
      }
   },
   "agent":{
      "cf:ppoliani":{
         "prov:type":"prov:Person",
         "foaf:mbox":"<mailto:ppoliani@gmail.com>"
      }
   },
   "entity":{
      "cf:tripLeg-1":{

      },
      "cf:trip-1":{

      },
      "cf:tripLeg-2":{

      }
   },
   "prefix":{
      "foaf":"http://xmlns.com/foaf/0.1/",
      "cf":"http://users.ecs.soton.ac.uk/pp6g11/ontology/carbonFooprints/"
   },
   "activity":{
```

```
      "cf:tripCreation":{

      }
   },
   "wasGeneratedBy":{
      "_:id5":{
         "prov:entity":"cf:trip-1",
         "prov:activity":"cf:tripCreation"
      },
      "_:id1":{
         "prov:entity":"cf:tripLeg-1",
         "prov:activity":"cf:tripCreation"
      },
      "_:id3":{
         "prov:entity":"cf:tripLeg-2",
         "prov:activity":"cf:tripCreation"
      }
   }
}
```


Apparently, the above representation of provenance graphs is not for human consumption. It can rather be processed by a tool that is capable of reading and understanding this sort of serializations.

4.6 Summary

In this chapter we have described the benefits of adapting the provenance of electronic data technology in our application. We presented the processes during which the provenance information is captured and illustrated that information in the form of graphs. Finally, a very brief description about the storage and exposure of the captured provenance information was presented. We will come back to those issues in the following chapter.

Chapter 5

Application Implementation

5.1 introduction

In chapter 3 we summarized the sequence of actions that occurred during the application's design part. In this chapter we will go through the implementation phase, outlining the core functionalities of the application. In addition, screen shots illustrating the application in action will be provided. Furthermore, a description about how the application was tested will be presented, along with the appropriate unit tests that were run. The chapter ends with a discussion and criticism about the final outcome of the implementation phase along with future work.

5.2 Application's Implementation Phase

In this section we will present how the application was implemented based on the decisions that were taken during the design phase. More specifically, we will introduce a set of sequence diagrams, illustrating the flow of function calls. The diagrams will be accompanied by a brief description, as well as, several screenshots showing the application in action.

5.2.1 Add Trip

As explained earlier, this use case requires users to insert the trips that they make during the course of time. The sequence diagram in figure 5.1 illustrates the process of adding information about the trips, whereas the sequence diagram in figure 5.3 demonstrates the flow of function calls that take place so that the information users insert is stored in a persistent database.

Figure 5.2 illustrates a web user interface for the trip creation process. The red rectangle indicates the *addTripview* Ember view which is the main form for adding new trips. The

tripLegContainer is shadowed with the blue rectangle and is, essentially, an Ember container view[1] capable of dynamically adding new sub-views, such as *tripLegView* for each trip. Finally, the yellow rectangle shows the carView view.

The sequence diagram 5.1 shows a scenario where the user initially inserts information concerning the trip's name, type, date and number of trip legs. When choosing the number of trip legs, the *tripLegContainer* is created which in turn creates a number of *tripLegView* views based on the number of trip legs. Then user again needs to indicate some information about the trip leg's source and destination address, and the transport mean that was used. Selecting a transport mean, results in the creation of the *tripLegTransportContainer* view, which is a dynamic view that we have used to support rich interactivity. In this scenario, user has chosen a car; therefore a *carView* view (yellow rectangle) is added to the *tripLegTransportContainer*. The form element in that view are populated with data that are fetched from the back-end via AJAX GET calls. Finally, after filling all the required fields, the user submits the form.

[1] $http://docs.emberjs.com/doc = Ember.ContainerView$

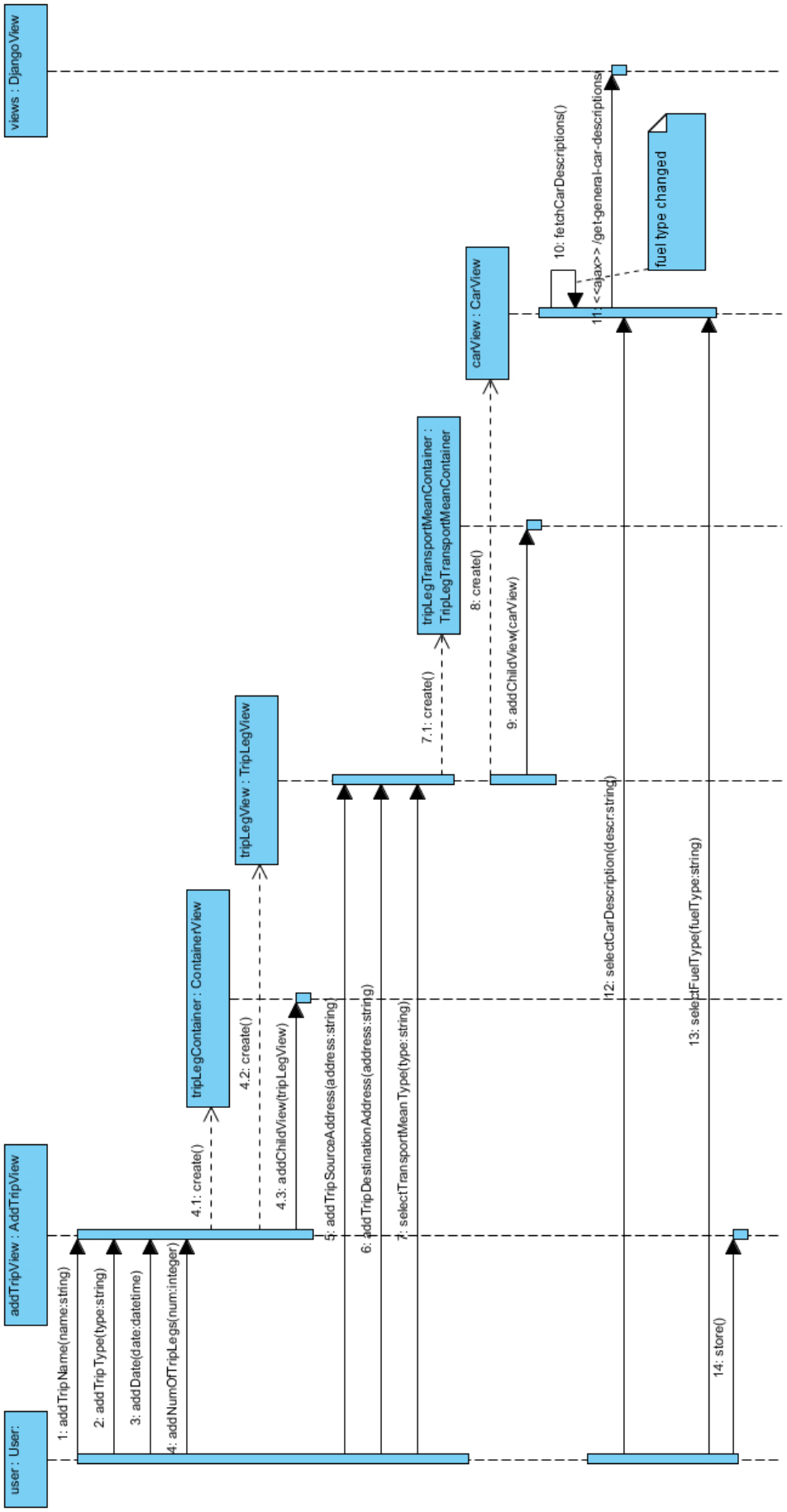

FIGURE 5.1: Add trip information sequence diagram

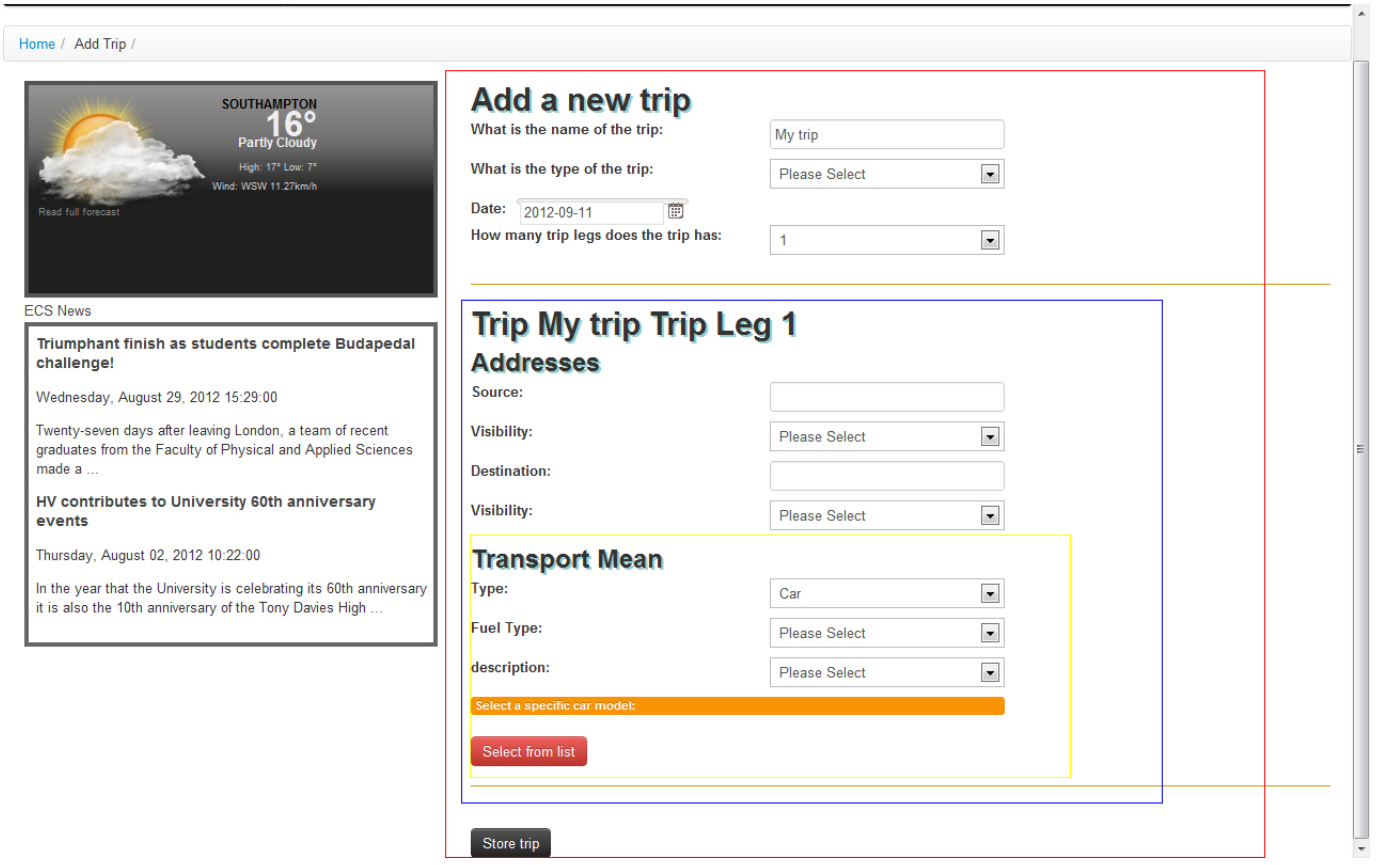

FIGURE 5.2: Add trip form

The process of storing a trip starts after the user submits the trip creation form, that we described earlier. Initially, the *tripManagerController* compiles the data inserted by the user and fills the corresponding models, namely tripLegModel and generalCarModel. It then makes an AJAX Post call to the back end, sending information about the trip. On the back-end, the corresponding view creates a trip model and stores it in the database. On the next step, the controller makes consequent AJAX Post request, asking for the back end to store each trip leg into the database. Finally, two more AJAX calls occur; the first asks the back-end to create a provenance graph for each trip leg, whereas the second asks for the carbon emissions of each trip leg to be calculated.

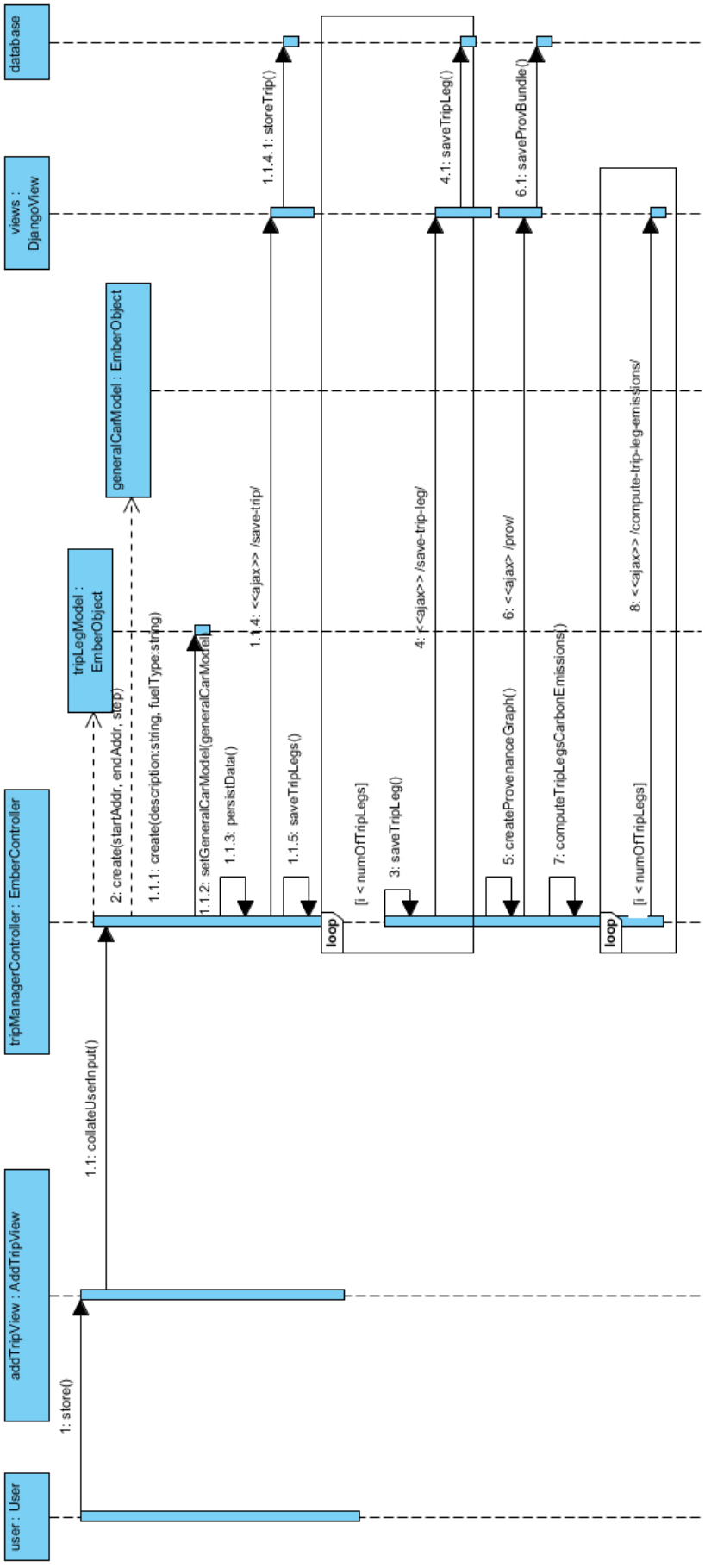

FIGURE 5.3: Trip storing sequence diagram

5.2.2 Show User's Trips

The user have the chance to manage the trip that he has added to the system. More specifically, the actions that are offered are: view, edit and delete tips. Figure 5.4 illustrates the web user interface that displays all user's trips. Notice that trips are categorized according to the transport means that were used.

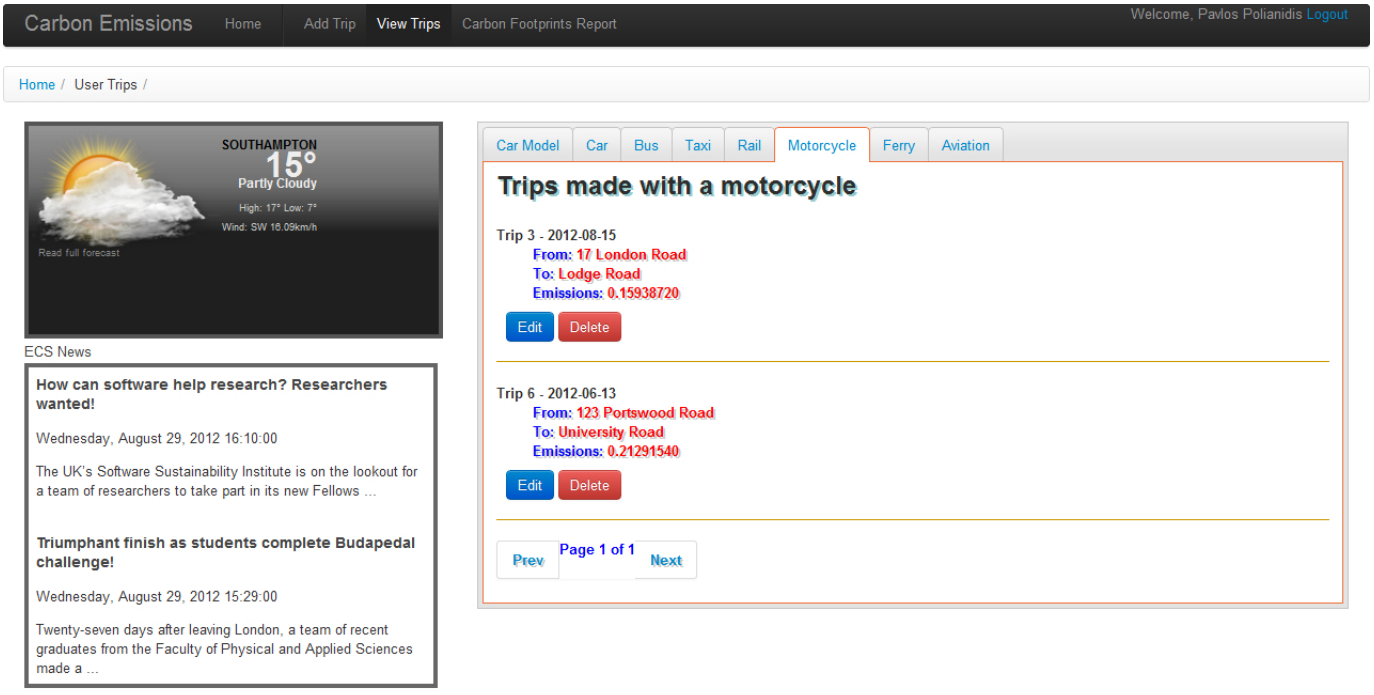

FIGURE 5.4: UI showing user's trips

As shown in the sequence diagram 5.5 the process is fairly simple. The *tripInfoController* fetches the trips made by a car from the back-end via an AJAX GET request. The back-end, in order, retrieves those data from the database and returns the in JSON format.

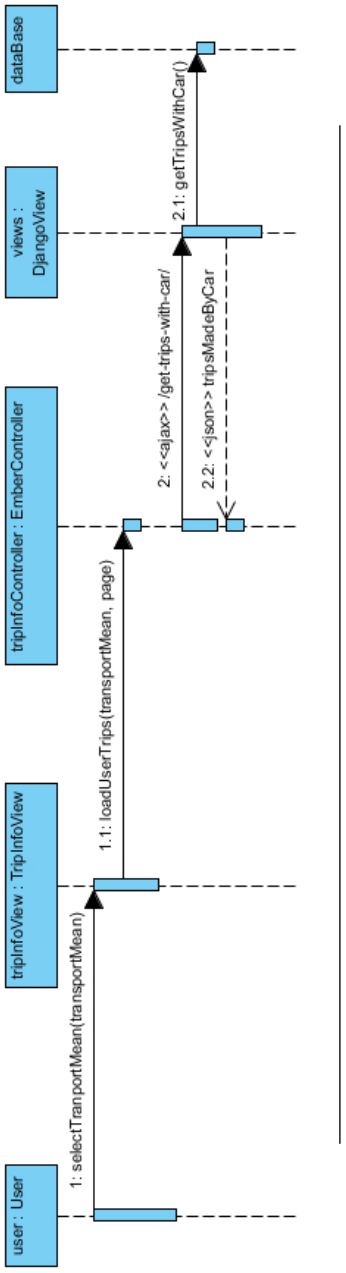

FIGURE 5.5: Displaying user's trips sequence diagram

5.2.3 Carbon Footprint Report

The prime incentive for users adding the trips they make is to present a report describing their individual, as well as, group carbon footprints. In its current state, the application supports visualization of carbon emissions in the form of line and bar charts. Furthermore, the provenance for trip legs' carbon emissions are illustrated with two types of graphs; dynamic and static provenance graphs. We will expand on that in a bit. Figure 5.6 illustrates the a small part of report html page (/report).

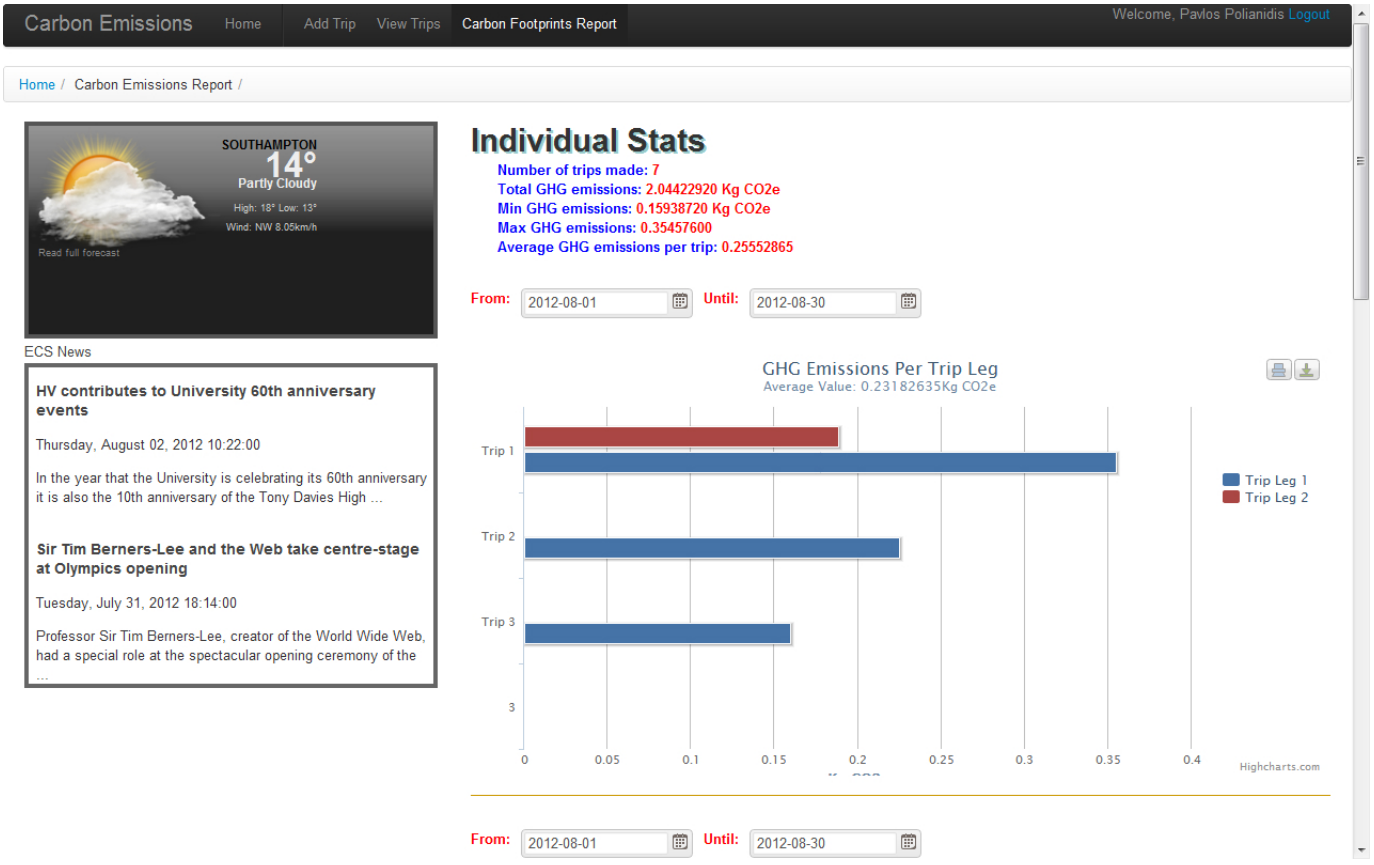

FIGURE 5.6: Carbon footprint report page

5.2.3.1 Individual Carbon Emissions

Users can view a small summary of their carbon footprints at the top of the report page. In its current state, the application presents the following statistics (view figure 5.7).

- The number of trips that a user has made overall.
- The total weight of greenhouse gas emissions (ghg) caused by user's travels.
- The minimum value of greenhouse gas emissions.

- The maximum value of greenhouse gas emissions.
- The average weight of greenhouse gas emissions (ghg) caused by user's travels.

Apart from that, the report includes four charts presenting different information each. The first chart (figure 5.7) summarizes the carbon emissions caused by trip legs, during a specified period of time. By default, the application shows the emissions during the current month. Nevertheless, users can define the time period with the aid of two calendar widgets that reside on the top of each chart. Additionally, all the charts can be printed and downloaded in several formats, namely in PNG, PDF, JPEG and SVG.

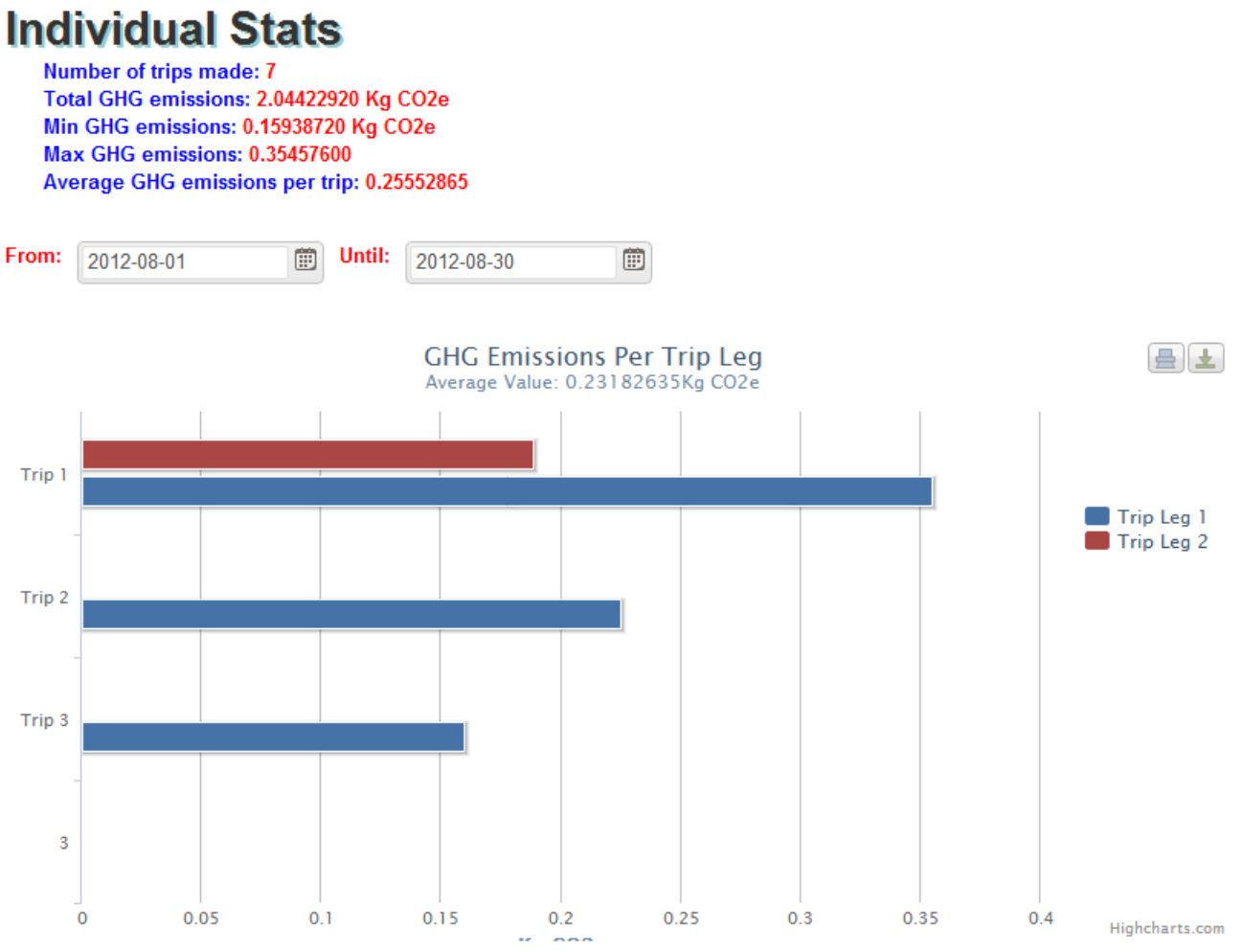

FIGURE 5.7: Greenhouse gas emissions per trip leg

As a complement to that chart, users can view the provenance graph of each trip leg carbon emission value. This is done by clicking on any bar; for instance, clicking on the first bar, the provenance information describing the derivation of that value, will appear (view figure 5.8).

This is a interactive graph, in that, users can relocate the nodes, zoom in and zoom out, and click on each node to view a brief description.

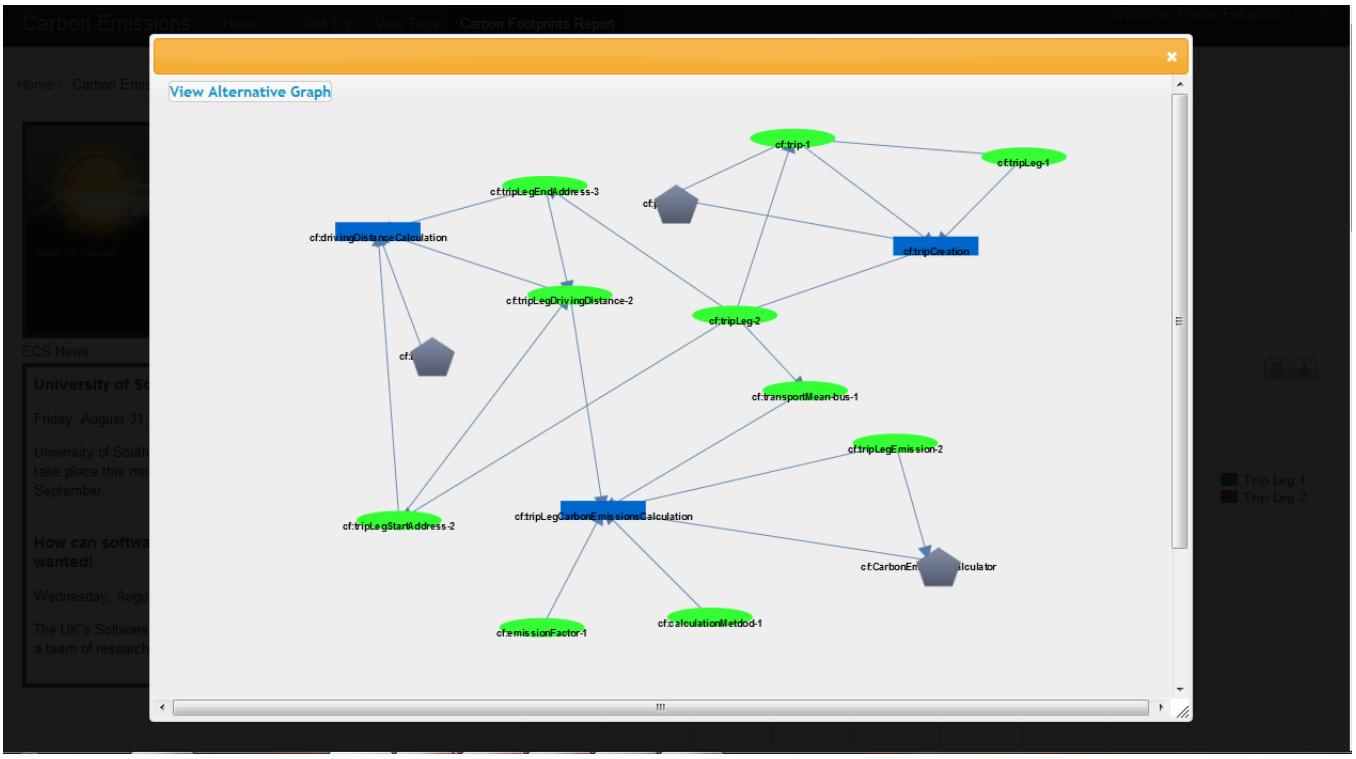

FIGURE 5.8: Interactive provenance graph

It is obvious that the interactivity comes at the cost of a non-intuitive visualization. That is to say, the graph is not very readable, unless user relocate manually all the nodes. As a result, a second static graph is provided (figure 5.9).

Greenhouse gas emissions can be summarized at the level of trips, as well. *The GHG Emissions Per trip* line chart (figure 5.10), visualizes the weight of ghg emissions for each trip user made, during a specified period of time.

The last chart (figure 5.11) pertaining to individual carbon footprints, illustrates the weight of carbon emissions caused by the different modes of transports, which were used during the trips made within a specified period of time.

5.2.3.2 Group Carbon Emissions

It was a initial requirement that users can be members of different groups and that carbon emissions of those groups are trackable. Consequently, aside from individual carbon footprint report, the application presents a report for users' groups. The information is similar to that enclosed in the individual report.

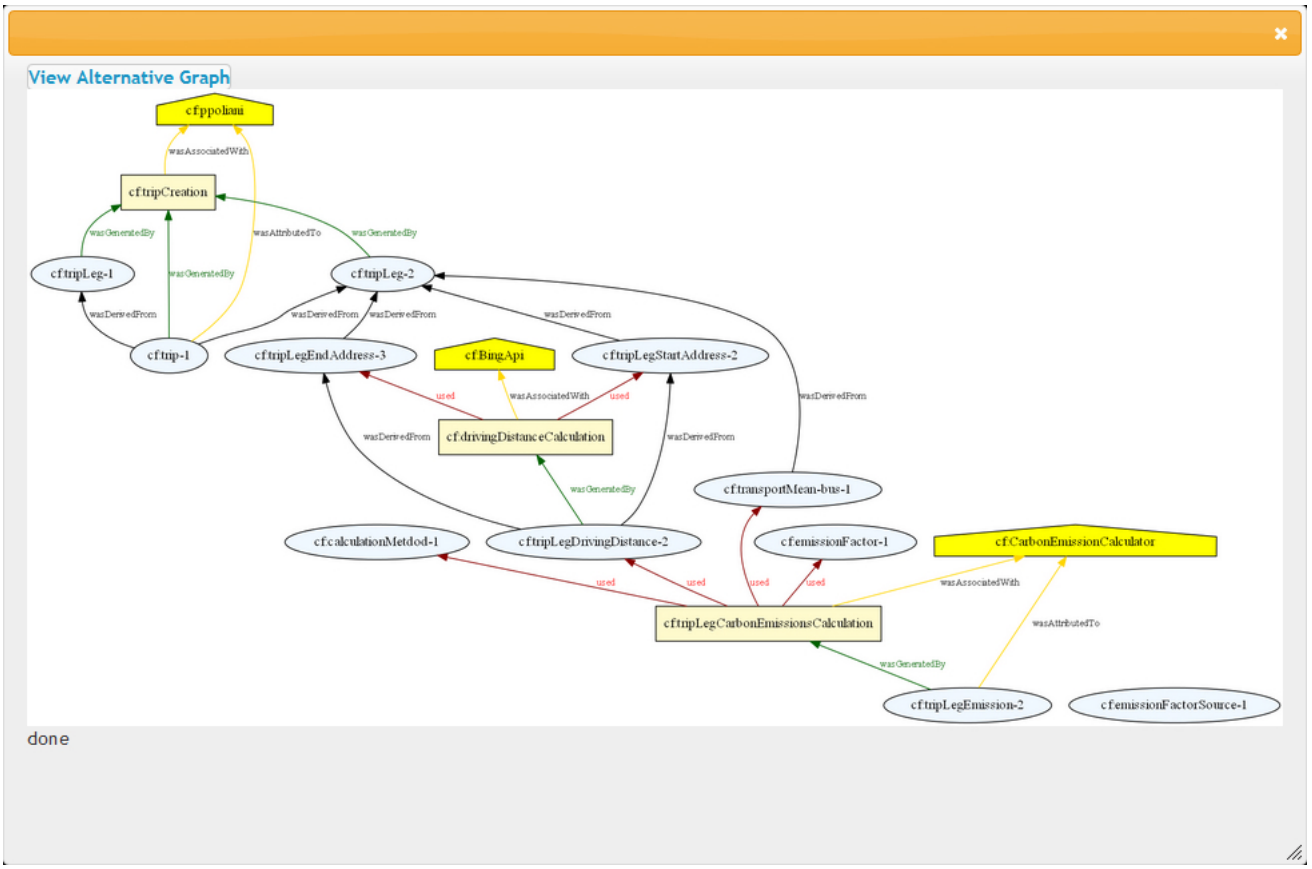

FIGURE 5.9: Static provenance graph

- The number of trips that were made by the members of all user's groups.
- The total weight of greenhouse gas emissions (ghg) caused by all trips made by the members of all user's groups
- The minimum value of greenhouse gas emissions.
- The maximum value of greenhouse gas emissions.
- The average weight of greenhouse gas emissions (ghg) caused by all trips made by the members of all user's groups

5.3 Evaluation via Testing

An initial requirement after finishing implementing the application was to get users try it and express their experience. However, this requirement changed and we tried to evaluate the system solely via tests. That is to say, each and every implemented functionality was accompanied by a unit test to verify that there are no bugs in the code. Furthermore, a series of accessibility

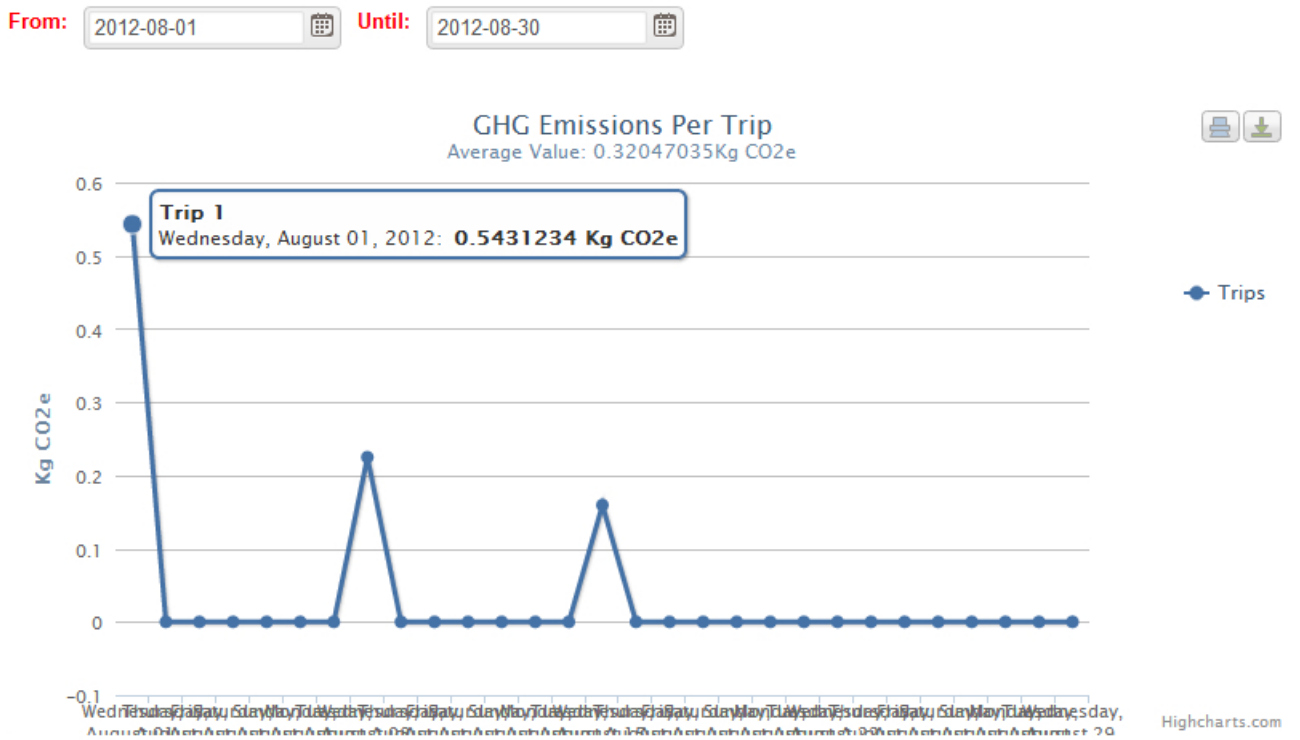

FIGURE 5.10: Greenhouse gas emissions per trip leg

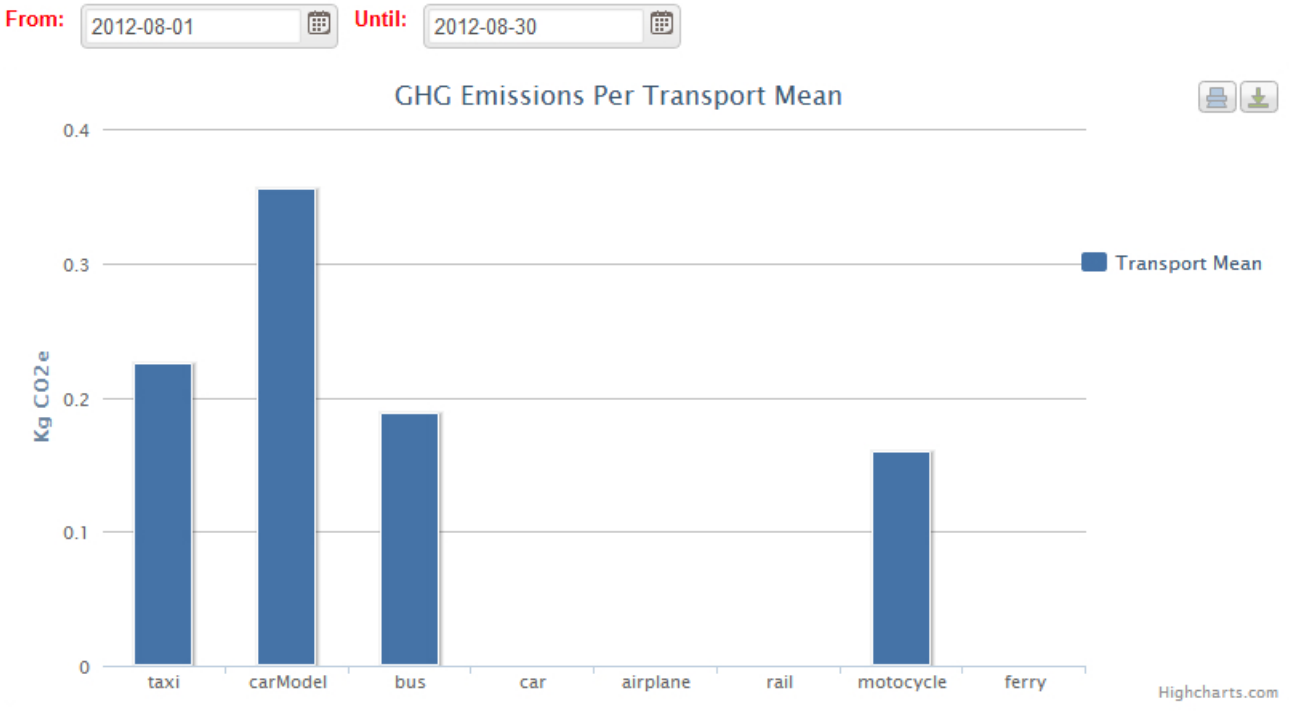

FIGURE 5.11: Greenhouse gas emissions per transport mean

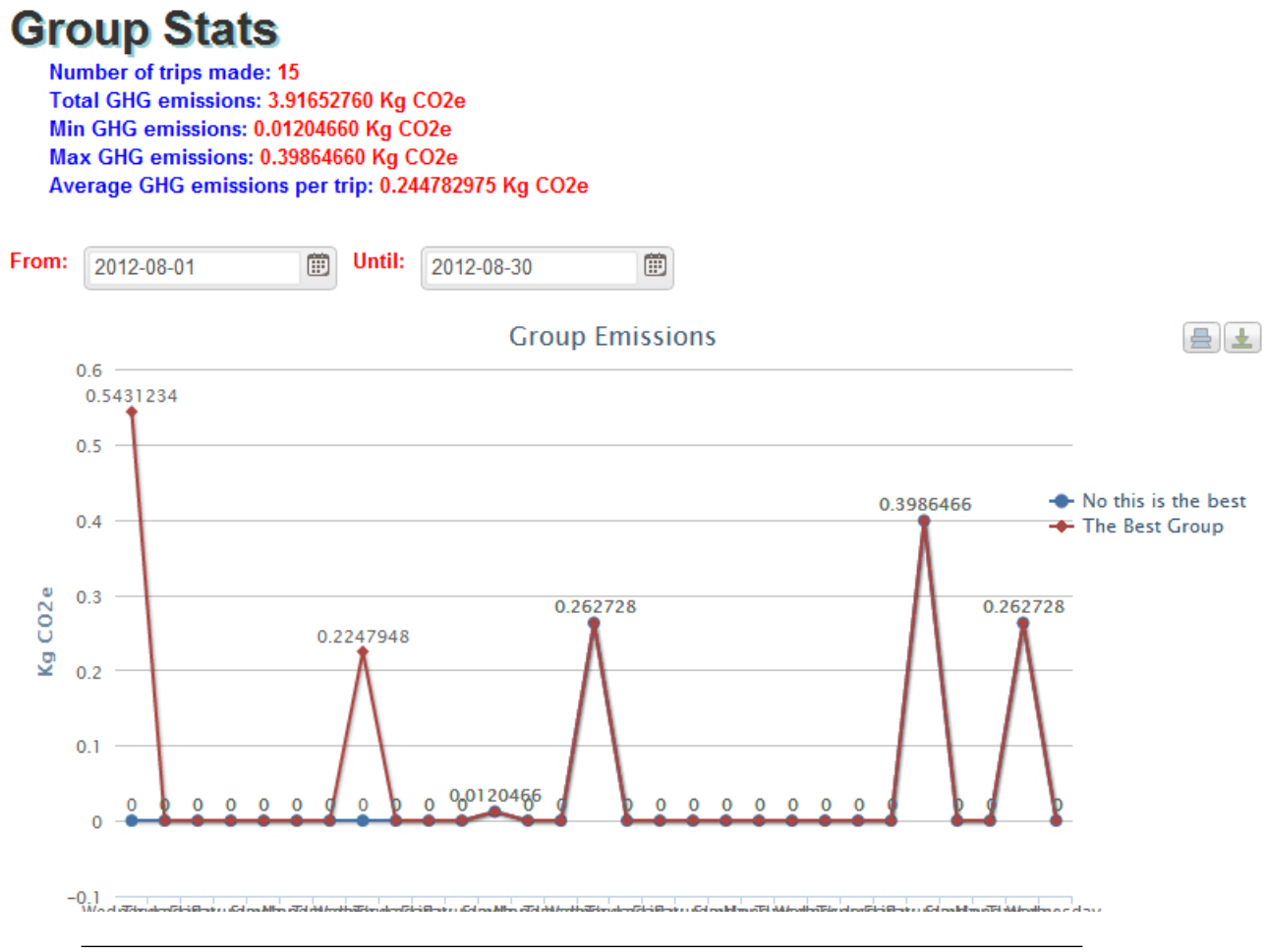

FIGURE 5.12: Group greenhouse gas emissions

and usability checks were performed. This section outlines the a subset of unit tests that were performed, as well as, presents the results of the accessibility and usability test.

5.3.1 Unit Testing

The prime reason for testing the application's code, is to verify that every piece of functionality is in accordance with the system's requirements. The majority of the unit tests we present here, refer to verifying the correct data insertion into the database. To that end, we utilized the Django's built-in unit test framework. The advantage of this decision is that there is no need to pollute the database with testing data, since the framework is capable of creating a temporary test database which is destroyed at the end of the testing process.

The function that is presented below consists of several actions taken before the execution of unit tests. In particular, those actions refer to setting up the tables of the database with some initial records that are ultimately being tested.

```
def setUp(self):
    self.user = User.objects.create(username='ppoliani')
    self.userProfile = models.UserProfile.objects.create(user=self.user, title='Mr', \
        type='student', occupation='student')

    self.address = models.Address.objects.create(country='UK', county='Hampshire', \
        city='Southampton', street='723 portswood road' ,postalCode='SO17 3ST', \
        longitude = 123.6765, latitude= 123.675, name='some name', visibility=False)
    self.trip = models.Trip.objects.create(userProfile=self.userProfile,
        type='commuter', name='trip name', date=datetime.now())
    self.car = models.Car.objects.create(manufacturer='Audi', model='A5',
        engineCapacity=2000, fuelType='petrol')
    self.tripLeg = models.TripLeg.objects.create(trip=self.trip,\
        startAddress=self.address, endAddress=self.address, transportMean=self.car,\
        step=1, time=datetime.time(datetime.now()))
    self.transportMeanUsedByUser =
        models.TransportMeansUsedByUsers.objects.create(transportMean=self.car,\
        userProfile=self.userProfile)

    self.emissionFactorSource =
        models.EmissionFactorSource.objects.create(name='Defra', year=2012,\
        link='http://link.com')
    self.emissionFactor =
        models.CarEmissionFactor.objects.create(source=self.emissionFactorSource,
        directGHGEmissions=0.1232, fuelType='petrol', carType='small car')

    self.transportMeanEmissionFactor =
        models.TransportMeanEmissionFactor.objects.create(transportMean=self.car, \
        emissionFactor=self.emissionFactor)
    self.calculationMethod =
        models.CO2CalculationMethod.objects.create(name='tier1 method', \
        description='some description', tier='1')
```

The list below summarizes a subset of the unit tests that were performed during the application's implementation phase.

```
def test_insertingUserProfiles(self):
    #passes
```

```
    self.assertEqual(self.user.get_profile().title,'Mr')

    #fails
    self.assertEqual(self.user.get_profile().type,'academic')
```

The *test_insertingUserProfiles* unit test verifies that insertions into table *UserProfiles* are performed without any errors. The first assertion must pass because in the set up function we stored a user who has the title Mr. Whereas, the second assertion fails because the type of the inserted user is student and not academic.

```
def test_insertAddresses(self):
    #passes
    self.assertEqual(self.address.country,'UK')

    #fails
    self.assertEqual(self.address.city,'London')
```

The *test_insertAddresses* unit test verifies that insertions into table *Addresses* are performed without any errors. The first assertion passes because we inserted an address with the value UK for the property country. Accordingly, the second assertion fails because the city of that address is Southampton and not London.

```
def test_insertTrips(self):
    #passes
    self.assertEqual(self.trip.name,'trip name')

    #fails
    self.assertEqual(self.trip.type, 'business')
```

The *test_insertTrips* unit test verifies that insertions into table *Trips* are performed without any errors. The first assertion is correct because the name of the trip that was inserted was indeed 'trip name'. On the other hand, the second assertion is false because a commuter type trip was inserted and not a business type.

```
def test_insertCars(self):
    #passes
    self.assertEqual(self.car.model,'A5')
```

```
    #fails
    self.assertEqual(self.car.engineCapacity, 100)
```

The *test_insertCars* unit test verifies that insertions into table *Cars* are performed without any errors. The car that was inserted has a value 'A5' for the model property and the engine capacity is 2000. As a result, the first assertion is true whereas the second is false.

```
def test_insertTripLegs(self):
    #passes
    self.assertEqual(self.tripLeg.trip.type,'commuter')
    self.assertEqual(self.tripLeg.trip.userProfile.type,'student')
    self.assertEqual(self.tripLeg.startAddress.country,'UK')

    #fails
    self.assertEqual(self.tripLeg.transportMean.engineCapacity, 3000)
```

The *test_insertTripLegs* unit test contains multiple tests. First of all insertions into table *TripLegs* are verified. In addition, connections between the TripLegs table and the *Addresses*, *TransportMeans* and *UserProfiles* tables are checked. The first assertion is true because the the type of the trip of which this trip leg is part of, is commuter. Similarly, the second assertion is true, since the type of the user that made the trip, is student. Finally, the third assertion is true, because the country of the start address matches the value that is checked (i.e. 'UK'). On the other hand, the last assertion fails because the engine capacity of the transport mean that was used, in not 3000 but 2000.

```
def test_insertTransportMeanUsedByUsers(self):
    #passes
    self.assertEqual(self.transportMeanUsedByUser.transportMean.model,'A5')

    #fails
    self.assertEqual(self.transportMeanUsedByUser.userProfile.title, 'Miss')
```

The *test_insertTransportMeanUsedByUsers* unit test checks that transport means used by users are correctly inserted and associated with the appropriate user. The first, assertion will pass because the model of the transport mean matches with the value checked. Similarly, the second assertion will fail because the title of the user that used that mode of transport is 'Mr' and not 'Miss'.

```
def test_insertingTransportMeanEmissionFactors(self):
    #passes
    self.assertEqual(self.transportMeanEmissionFactor.transportMean.model, 'A5')

    #fails
    self.assertEqual(self.transportMeanEmissionFactor.emissionFactor.source.name,
        'WrongName')
```

The *test_insertingTransportMeanEmissionFactors* unit tests checks that emissions factors are correctly connected to the corresponding transport means. Thus, the first assertion is true because we associated the emission factor with the transport mean with the value 'A5' for the model property. The second assertion checks if the conection between the *EmissionsFactors* and *EmissionFactorSources* tables are properly applied. The assertion fails because the name of the emission source is 'Defra' and not 'WrongName'.

```
def test_tripLegCarbonEmissionCalculation(self):
    c = Client()

    # Extra parameters to make this a Ajax style request.
    kwargs = {
        'tripLegId':self.tripLeg.id,
        'drivingDistance': 100,
        'transportMeanType': 'car',
        'calculationMethod': 'tier1'
    }
    response = c.post('/compute-trip-leg-emissions/', kwargs)

    #pass
    emissions = float(models.TripLegCarbonEmission.objects.get(tripLeg=self.tripLeg)
        .emissions)
    self.assertEqual( emissions, 12.32)

    #fail
    self.assertEqual( emissions, 120.32)
```

The *test_tripLegCarbonEmissionCalculation* unit test is slightly different that the previous ones. The reason is that it creates a temporary client that is capable of making HTTP POST request. In essence, it imitates an AJAX Post request to the '/compute-trip-leg-emissions/'

address passing some data for computing the weight of carbon emitted during a trip leg. The first assertion is true because the expected value matches the computed one. This is exactly the reason why the second assertion fails.

5.3.2 Accessibility and Usability Test

Our application is a web based application; we therefore tried to meet several accessibility and usability requirements. Table 5.1 summarizes the results of the accessibility and usability test. The test was conducted according to the guidelines published on web2access.org.uk [2].

5.4 Critical Assessment

Having described the design and implementation of the application, we can now provide a criticism about the final outcome. The application is missing several features that were intended to be implemented, in the first place.

First of all, one of the main challenges was to design an application that would ease the task of adding new data. To that end, the initial intention was to design a user interface that would allow users to re-use the data that they have already inserted. More specifically, the process of adding new trips is relatively dull, and users would definately try to avoid it. Moreover, it is very likely that they are going to add the same trip several times; therefore, a functionality that would allow users to use the same trip again and again without the need of repeating themselves, was planed to be supported by the application. The infrastructure for implementing that feature is already there, which means that it can be easily incorporated into the current application.

Another requirement that was not met, has to do with users' input validation on client-side. Data that users insert into the *trip creation form* needs to be validated, so that only the correct types of data are sent to the back-end. There was an attempt to support this feauture by utilizing a JavaScript library and the native HTML5 validation mechanism. However, the html template engine that we used (Handlebars.js) did not seem to cooperate well with those libraries. Hence, the implementation of that feature was postponed for later, but we did not consider LeBlanc's law: *Later equals never*.

A better navigation through users data was also planned to be included in the application. In the current state, the application displays users' trips categorized according to the mode of transport that was used. An additional feature would be to view trips refering to specific addresses ot trips made at a particular date.

[2] http://www.web2access.org.uk/test

Test	Result
Are login, signup and other forms accessible?	Simple, accessible forms with clear labels e.g. 'username (email)' and 'password'.
Are text alternatives offered for images etc?	Acceptable alternative text throughout.
Is the target for every link clearly defined?	Links fully appropriate, used throughout the site plus alternative navigation element.
Do frames and iframes have appropriate titles and suitable layout?	No frames or iframes used for design.
Is the page fully functional and fully navigable without the stylesheet?	Content and navigation accessible.
Are tables used inappropriately to format the page?	Page layout does not use tables and/or headered tables are used to present data.
Is tab order logical?	Intermittent random ordering with selection jumping from one area to another.
Are the pages beyond login functional and navigable with the keyboard?	Non-critical features on the page require mouse use. Maybe browser dependent i.e. works with one browser but not another.
If a rich-text editor is used, is it accessible?	Fully accessible and usable or Not Applicable.
Is there appropriate feedback after submitting information and adequate time allowances?	Appropriate feedback, user directed to what they should do next, no time restrictions.
Is the content comfortable to read with good colour contrast levels and no colour deficiency issues?	Site colour contrast adequate but some non-critical text in bizarre fonts or not comfortable to read
Does the page maintain its style and usability when the browser zoom feature is used?	Layout and readability unaffected when zooming.
Is text size and style suitably readable? Is there any blinking or flashing	Sans-serif fonts used in all body content (excludes headings). Regular font size (10/12pt) throughout and reasonable layout.

Table 5.1: Accessibility and usability test results

With regards to carbon footprint report, more detailed information has to be incorporated; for instance, the corresponding trip for the minimum ghg emissions value should be indicated. However, because all the required data are already stored in the database, we consider that the implementation of such features would be easy.

Considering the provenance graphs that are presented in the carbon emission report, we need to admit that they do not cover the whole range of graphs that were planed to be implemented. Examples of graphs that are missing are: graphs that illustrate the provenance of the carbon emission values of particular trips, or graphs that display the provenance of individual's or group's carbon emissions.

It is obvious that there are several requirements that the application does not meet. The main reason for that is that a considerable amount of time was spend to design and implement the main infrastructure that would support those features. We therefore acknowledge that meeting those requirements is a matter of a few days' work, trying to bring all pieces of the puzzle together.

5.5 Future Work

There still are various features that could extend the current capabilities of the application. We need to clarify that the following list is not complete and that there might be numerous functionalities that can be added to an online carbon emission calculator.

To start with, we believe that a better solution for keeping track of users' trips would be to utilize the GPS mechanism of mobile devices. In particular, a mobile app can record users' geo locations and make educated estimations about the mode of transports used during the trips. This way, users are not required to add any information to the system. There is a significant caveat with this approach, though. There might emerge privacy issues, since the system would constantly know users' locations. Thus a good understanding of the possible limitations has to be acquired.

Another significant feature that might make users' life better, is to allow them make various customizations; for instance, users can pre-define a set of transport means and addresses that they usually use. Then the application could show those lists every time users add a new trip. This would make much easier and faster the process of adding trips. Furthermore, the results of carbon emission calculations could become more accurate if the users would specify the exact routes of the trips they add. Right now, the application calculates the driving distance of the default route between two addresses. In future editions, users will be able to define precisely those routes.

In terms of calculating carbon emissions, the application could use a more expanded set of calculation methods. Those methods could be added as plug-in methods to the current application. Additionally, calculation methods should be executed when missing data are provided; for instance, when users do not specify the fuel type of a transport mean.

The application can be further extended by adding a recommendation engine, which could help minimizing individual and group carbon footprints. In essence, the engine would suggest modes of transports that have low carbon emissions and are appropriate for the users' daily journeys.

Finally, an ontology for describing relationships in provenance graphs could be designed. This ontology would, essentially, be a special case of the PROV-Ontology.

5.6 Summary

In this chapter we reviewed:

- the implemenation of the application. For that reason, a set of sequence diagrams were given showing the execution flow of different parts of the application. Screenshots demonstrating the application's UI were also presented.
- the evaluation of the application by presenting several test cases, as well as, the results of the accessibility and usability tests.
- some aspects of the implementation, that we could have done considered differently, and
- some additional features which believe that future versions of the application can support.

Chapter 6

Conclusion

The awareness of the damage we bring to the environments is considerably important. Each and every one should have knowledge of the weight of carbon emissions that his activities cause to the environment. The web application that is described in this report provides a means for the users of an organization (e.g. Universities) to monitor the carbon emissions caused by their travelling habits. This is extended with a feature that allows them to verify and make trust judgments about the values that are produced by the application. We started the report with a description of the theory underpinning the provenance of electronic data technology. In addition to that, the bits of the carbon footprint theory that we used in our application were presented, with emphasis given methods for calculating carbon emissions caused by travelling activities.

A separate chapter was dedicated to explaining the benefits that the provenance technology adds to the application. Finally, a substantial amount of work was devoted to designing and implementing the web and mobile applications. We discussed the methodology that was followed to complete each stage in two separate chapters.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] Wang-Chiew Tan. Provenance in Databases: Past, Current, and Future. URL http://users.soe.ucsc.edu/~{}wctan/publications.html.

[2] Luc Moreau. The foundations for provenance on the web. *Found. Trends Web Sci.*, 2 (2–3):99–241, February 2010. ISSN 1555-077X. doi: 10.1561/1800000010. URL http://dx.doi.org/10.1561/1800000010.

[3] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. The provenance of electronic data. *Commun. ACM*, 51(4):52–58, April 2008. ISSN 0001-0782. doi: 10.1145/1330311.1330323. URL http://doi.acm.org/10.1145/1330311.1330323.

[4] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, March 2005. ISSN 0360-0300. doi: 10.1145/1057977.1057978. URL http://doi.acm.org/10.1145/1057977.1057978.

[5] Shawn Bowers, Timothy McPhillips, Martin Wu, and Bertram Ludscher. Project histories: Managing data provenance across collection-oriented scientific workflow runs. In Sarah Cohen-Boulakia and Val Tannen, editors, *Data Integration in the Life Sciences*, volume 4544 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-73254-9. doi: 10.1007/978-3-540-73255-6_12. URL http://dx.doi.org/10.1007/978-3-540-73255-6_12.

[6] Chenyun Dai, Dan Lin, Elisa Bertino, and Murat Kantarcioglu. An approach to evaluate data trustworthiness based on data provenance. In Willem Jonker and Milan Petkovic, editors, *Secure Data Management*, volume 5159 of *Lecture Notes in Computer Science*, pages 82–98. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-85258-2. URL http://dx.doi.org/10.1007/978-3-540-85259-96. 10.1007/978-3-540-85259-96.

[7] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):

3045–3054, 2004. doi: 10.1093/bioinformatics/bth361. URL http://bioinformatics.oxfordjournals.org/content/20/17/3045.abstract.

[8] Shawn Bowers and Bertram Ludscher. Actor-oriented design of scientific workflows. In *In 24st Intl. Conference on Conceptual Modeling*. Springer, 2005.

[9] Juliana Freire, Cláudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing rapidly-evolving scientific workflows. In *Proceedings of the 2006 international conference on Provenance and Annotation of Data*, IPAW'06, pages 10–18, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-46302-X, 978-3-540-46302-3. doi: 10.1007/11890850_2. URL http://dx.doi.org/10.1007/11890850_2.

[10] I. Foster, J. Vockler, M. Wilde, and Yong Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 37 – 46, 2002. doi: 10.1109/SSDM.2002.1029704.

[11] Simon Miles, Ewa Deelman, Paul Groth, Karan Vahi, Gaurang Mehta, and Luc Moreau. Connecting scientific data to scientific experiments with provenance. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, pages 179–186, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3064-8. doi: 10.1109/E-SCIENCE.2007.22. URL http://dx.doi.org/10.1109/E-SCIENCE.2007.22.

[12] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376772. URL http://doi.acm.org/10.1145/1376616.1376772.

[13] Peter Buneman, James Cheney, Wang-Chiew Tan, and Stijn Vansummeren. Curated databases. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '08, pages 1–12, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-152-1. doi: 10.1145/1376916.1376918. URL http://doi.acm.org/10.1145/1376916.1376918.

[14] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 316–330, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-41456-8. URL http://dl.acm.org/citation.cfm?id=645504.656274.

[15] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *In ICDE*, pages 367–378, 1999.

[16] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '07, pages 31–40, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-685-1. doi: 10.1145/1265530.1265535. URL http://doi.acm.org/10.1145/1265530.1265535.

[17] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005. ISSN 0163-5808. doi: 10.1145/1084805.1084812. URL http://doi.acm.org/10.1145/1084805.1084812.

[18] Ewa Deelman, G. Bruce Berriman, Ann L. Chervenak, Óscar Corcho, Paul T. Groth, and Luc Moreau. Metadata and provenance management. *CoRR*, abs/1005.2643, 2010.

[19] Simon Miles, Paul Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. Extracting causal graphs from an open provenance data model. *Concurr. Comput. : Pract. Exper.*, 20(5):577–586, April 2008. ISSN 1532-0626. doi: 10.1002/cpe.v20:5. URL http://dx.doi.org/10.1002/cpe.v20:5.

[20] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM. ISBN 0-89791-133-4. doi: 10.1145/800057.808695. URL http://doi.acm.org/10.1145/800057.808695.

[21] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131428985.

[22] Paul Groth, Simon Miles, and Luc Moreau. A model of process documentation to determine provenance in mash-ups. *ACM Trans. Internet Technol.*, 9(1):3:1–3:31, February 2009. ISSN 1533-5399. doi: 10.1145/1462159.1462162. URL http://doi.acm.org/10.1145/1462159.1462162.

[23] Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An architecture for provenance systems. Technical report, University of Southampton, November 2006. URL http://eprints.ecs.soton.ac.uk/13216/.

[24] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Gener. Comput. Syst.*, 27(6):743–756, June 2011. ISSN 0167-739X. doi: 10.1016/j.future.2010.07.005. URL http://dx.doi.org/10.1016/j.future.2010.07.005.

[25] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. Mcgrath, Jim Myers, and Patrick Paulson. Provenance and annotation of data and processes. chapter The Open Provenance

Model: An Overview, pages 323–326. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-89964-8. doi: 10.1007/978-3-540-89965-5_31. URL http://dx.doi.org/10.1007/978-3-540-89965-5_31.

[26] Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, and Satya Sahoo. Prov-dm: The prov data model. W3c working draft, W3C, July 2012. http://www.w3.org/TR/2012/WD-prov-dm-20120724/.

[27] Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. Prov-o: The prov ontology. W3c working draft, W3C, July 2012. http://www.w3.org/TR/2012/WD-prov-o-20120724/.

[28] James Cheney and Stian Soiland-Reyes. Prov-n: The provenance notation. W3c working draft, W3C, July 2012. http://www.w3.org/TR/2012/WD-prov-n-20120724/.

[29] James Cheney, Paolo Missier, and Luc Moreau. Constraints of the provenance data model. W3c working draft, W3C, May 2012. http://www.w3.org/TR/2012/WD-prov-constraints-20120503/.

[30] Luc Moreau, Olaf Hartig, Yogesh Simmhan, James Myers, Timothy Lebo, Khalid Belhajjame, and Simon Miles. Prov-aq: Provenance access and query. W3c working draft, W3C, June 2012. http://www.w3.org/TR/2012/WD-prov-aq-20120619/.

[31] Khalid Belhajjame, Helena Deus, Daniel Garijo, Graham Klyne, Paolo Missier, Stian Soiland-Reyes, and Stephan Zednik. Prov model primer. W3c working draft, W3C, July 2012. http://www.w3.org/TR/2012/WD-prov-primer-20120724/.

[32] Simon Miles. Electronically querying for the provenance of entities. In Luc Moreau and Ian Foster, editors, *Third International Provenance and Annotation Workshop*, volume 4145, pages 184–192. Springer, 2006. URL http://eprints.soton.ac.uk/262567/. Event Dates: May 2006.

[33] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40 –48, sept. 1994. ISSN 0163-6804. doi: 10.1109/35.312842.

[34] Jonathan Moffett, Morris Sloman, and Kevin Twidle. Specifying discretionary access control policy for distributed systems. *Computer Communications*, 13(9):571 – 580, 1990. ISSN 0140-3664. doi: 10.1016/0140-3664(90)90008-5. URL http://www.sciencedirect.com/science/article/pii/0140366490900085.

[35] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38 –47, feb 1996. ISSN 0018-9162. doi: 10.1109/2.485845.

[36] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM. ISBN 1-59593-518-5. doi: 10.1145/1180405.1180418. URL http://doi.acm.org/10.1145/1180405.1180418.

[37] William Stallings. *Cryptography and Network Security (4th Edition)*. Prentice Hall, November 2005. ISBN 0131873164. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0131873164.

[38] Ralph Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology CRYPTO 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer Berlin / Heidelberg, 1990. ISBN 978-0-387-97317-3. URL http://dx.doi.org/10.1007/0-387-34805-021. 10.1007/0-387-34805-021.

[39] Steve Graham, Doug Davis, Simeon Simeonov, Toufic Boubez, Ryo Neyama, and Yuichi Nakamura. *Building Web Services with Java: Making Sense of Xml, Soap, Wsdl, and Uddi*. Sams, Indianapolis, IN, USA, 1st edition, 2001. ISBN 0672321815.

[40] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. ISBN 0672323915.

[41] Simon Miles. Electronically querying for the provenance of entities. In *Proceedings of the 2006 international conference on Provenance and Annotation of Data*, IPAW'06, pages 184–192, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-46302-X, 978-3-540-46302-3. doi: 10.1007/11890850_19. URL http://dx.doi.org/10.1007/11890850_19.

[42] IPCC. Climate change 2007: Synthesis report: Contribution of working groups i, ii and iii to the fourth assessment report of the intergovernmental panel on climate change. 2007.

[43] Daniel A. Lashof. Relative contributions of greenhouse gas emissions to global warming. *Nature*, 344(6266):529–531, April 1990. doi: 10.1038/344529a0. URL http://dx.doi.org/10.1038/344529a0.

[44] Laurence A. Wright, Simon Kemp, and Ian Williams. 'carbon footprinting': towards a universally accepted definition. *Carbon Management*, 2(1):61–72, 2011. URL http://eprints.soton.ac.uk/210245/.

[45] I.D. Williams, Simon Kemp, Jonathan Coello, David A.. Turner, and Laurence A. Wright. A beginner's guide to carbon footprinting. *Carbon Management*, 3(1):55–67, February 2012. URL http://eprints.soton.ac.uk/210237/.

[46] Laurence A. Wright, Jonathan Coello, Simon Kemp, and Ian Williams. Carbon footprinting for climate change management in cities. *Carbon Management*, 2(1):49–60, February 2011. URL http://eprints.soton.ac.uk/186331/.

[47] Manfred Lenzen, Joy Murray, Fabian Sack, and Thomas Wiedmann. Shared producer and consumer responsibility – theory and practice. *Ecological Economics*, 61(1):27–42, 2007. URL http://EconPapers.repec.org/RePEc:eee:ecolec:v:61:y:2007:i:1:p:27-42.

[48] HEFCE. Measuring scope 3 carbon emissions - transport. 2012.

[49] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 13–18, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-891-6. doi: 10.1145/1314313.1314318. URL http://doi.acm.org/10.1145/1314313.1314318.

[50] Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security issues in a soa-based provenance system. In *Third International Provenance and Annotation Workshop*. Springer, 2006. URL http://eprints.soton.ac.uk/262569/. Event Dates: May 2006.

[51] Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using Semantic Web Technologies for Representing E-science Provenance. In *The Semantic Web ISWC 2004*, pages 92–106. 2004. URL http://www.springerlink.com/content/1xag3695fgn78dme.

[52] Fenglian Xu, Alexis Biller, Liming Chen, Victor Tan, Paul Groth, Simon Miles, John Ibbotson, and Luc Moreau. A proof of concept design for provenance. Technical report, University of Southampton, March 2005. URL http://eprints.soton.ac.uk/260687/.

[53] Geoffrey Barbier and Huan Liu. Information provenance in social media. In John Salerno, Shanchieh Yang, Dana Nau, and Sun-Ki Chai, editors, *Social Computing, Behavioral-Cultural Modeling and Prediction*, volume 6589 of *Lecture Notes in Computer Science*, pages 276–283. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-19655-3. URL http://dx.doi.org/10.1007/978-3-642-19656-039. 10.1007/978-3-642-19656-039.

[54] Divya Pandey, Madhoolika Agrawal, and Jai Pandey. Carbon footprint: current methods of estimation. *Environmental Monitoring and Assessment*, 178:135–160, 2011. ISSN 0167-6369. URL http://dx.doi.org/10.1007/s10661-010-1678-y. 10.1007/s10661-010-1678-y.