Thesis Title

Politecnico di Milano

Carlos Blasco Andrés

April 20, 2020

Abstract

Abstract goes here

Dedication

To mum and dad

Declaration

I declare that..

Acknowledgements

I want to thank...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This is a test.

1.1 Section 1

This a test for the first section.

1.2 Section 2

And this is a test for the second section.

Chapter 2

State of the Art

2.1 Data-Intensive Applications

Data-Intensive Applications (DIAs) are a class of parallel computing applications which handle high amount of datasets and that devote the most of their processing time to I/O and manipulate such datasets. These kind of applications are able to manage datasets of several terabytes and petabytes which are available in a wide variety of formats and that are distributed among several locations. This capability to process large volumes of data is possible due to the fact that DIAs use multi-step analytical pipelines with transformations and fusion stages in order to parallelize the execution of the data.

Intensive applications can be classified according to the parallel processing approaches commonly known as compute-intensive and data-intensive. On the one hand, compute-intensive is used to describe application programs that are compute bound. Such applications devote most of their execution time to computational requirements and they process small volumes of data. In this kind of applications, processes must be parallelized by means of individual algorithms and then they must be decomposed into separate tasks. Later, such tasks are executed in a pipeline reaching higher performance that in serial processing. Finally, compute-intensive applications are able to perform multiple operations simultaneously as they allow task parallelism.

On the other hand, data-intensive is used to describe application programs that are I/O bound. Such applications devote most of their execution time to I/O, move and manipulate the datasets. This second parallel processing approach, unlike compute-intensive, processes large volumes of data. Data-intensive applications require the partition or subdivision of the initial dataset in some smaller datasets due to the fact that the large volume of data that they are composed. These partitions must be processed independently by the application allowing the parallelization of the initial dataset. Once the partitions are executed, they must be reassembled in an output dataset.

As any type of application which takes advantage of data-parallelism, DIAs need to select an algorithm not only to distribute data among the processing nodes of the cluster but also to execute the data in such processing nodes. Moreover, they need to adopt a strategy to decompose the datasets, find a trade-off in terms of working loads in the processing nodes by means of load balancing systems and to communicate the nodes of the cluster.

2.1.1 Characteristics

Data-intensive applications are modeled in a different way than any other computing application due to the property of DIAs to be processed in a cluster infrastructure. Moreover, as any other system, DIAs need to reach a high performance in order to be make our application efficient; this is possible by means of the minimization of the number of movements of the data. Other important properties required by DIAs are that they must be available when they are called, annd that they must be reliable to the number of failures; both properties are accomplished due to data-intensive applications are fault resilient. Last but not least, data-intensive applications must be able to change the number of required nodes taking into account the workload. Then, four characteristics can be defined in order to distinguish data-intensive applications from any other kind of computing system:

Dataflow Model

Data-intensive applications are modeled by means of nodes where some transformations take place and data or programs flow between such nodes. Moreover, such model allows to control how to schedule, to execute, to balance the load and to communicate programs and data during runtime using a cluster computing infrastructure.

Data Movement Minimization

Minimize the number of movements required by each data along the different computations in order to achieve high performance is crucial in data-intensive applications. Then, processing algorithms are executed on the nodes of the dataflow model where the data is located, reducing the number of movements and therefore increasing performance.

Fault Resilient

Due to the fact that DIAs work with high amounts of data, the probability for the hardware to fail is greater than in other computing systems but the same happens with the probability of communications errors and software bugs. This is why, DIAs are designed to be fault resilient. In order to make these applications available and reliable, on the disk of the storage of such systems redundant copies of all data files or intermediate processing results can be found. But also they are provided with systems capable to detect the failure of any node or processing failures and, once such failures are detected, then re-compute the results.

Node Variability

Depending on the hardware and software architecture of the data-intensive application, the required number of nodes and processing tasks and its variability along the runtime can be fix or variable. This is due to the fact that it is very important to achieve a comfortable processing independently of the amount of data and, also, to reduce the critical time of computation by adding some nodes.

2.1.2 Frameworks

2.1.2.1 Apache Hadoop

Apache Hadoop is an open source software framework that supports distributed applications. As any other DIA, Hadoop allows to work with thousands of nodes and petabytes of data. This framework is composed by a processing layer called MapReduce which was inspired by Google documentation.

MapReduce works as a DIA processing large volumes of data. First of all, this programming model divides the incoming works into a set of independent tasks and, after that, such tasks are executed. Then, the user sends the complete job to a master which divides it into independent tasks and submits them to the slaves in order to process the works in parallel. This process allows Hadoop to reach speed and reliability of cluster.

In summary, Hadoop MapReduce works breaking the incoming datasets into independent sets that are executed in parallel by means of two phases: map phase and reduce phase. Firstly, in the map phase, we specify the business logic putting the custom code in the way MapReduce works. And, secondly, in the reduce phase, we specify processing like summations and aggregations.

2.1.2.2 Spark

Spark is an open source platform that allows general-purpose and fast cluster computing. It is an engine that allows to process large volumes of data.

Spark enables users to access repeatedly to datasets in order to perform streaming, machine learning or SQL workloads by means of high-level APIs in Java, Scala, Python and R. It also allows users to accomplish batch processing or stream processing.

Moreover, Spark can be integrated with any other data-intensive tool due to its design. Spark is compatible with Hadoop as it is able to access Hadoop data sources and run on Hadoop clusters. This platform extends MapReduce, the engine of Hadoop, by including iterative queries and stream processing.

Some features make Spark more efficient than Hadoop. The following features make Spark being the 3G of Big Data:

- Fault tolerance.
- Dynamic in nature.
- Reusability.
- Advanced analytics.
- Lazy evaluation.
- Real-time stream processing.
- In-memory computing.

The in memory computing feature reduces the number of read-write to disk operations what achieves a data processing performance 100 times faster in memory and 10 times faster on the disk. Due to the high amount of operators that Spark

provides, to achieve parallel applications is easier with Spark. Regarding to fault tolerance, Sparks handles the failure of the nodes in the cluster by means of Resilient Distributed Datasets (RDDs) which its specific design makes to reduce the number of failures to zero.

In conclusion, Spark can be differentiated from Hadoop in its cluster management system. On the other hand, regarding to the similarities between Spark and Hadoop, Spark takes advantage of Hadoop for storage.

2.1.2.3 Apache Flink

Apache Flink is an open source platform which allows data streaming computation because of its data flow engine. Moreover, as happens with Spark, Apache Flink is able to execute stream processing and batch processing and it is also compatible with Hadoop.

Apache Flink can undertake efficiently some different types of processing, becoming in the most potent open source platform in the market to handle them. Such processing types are:

- Batch Processing.
- Interactive processing.
- Real-time stream processing.
- Graph Processing.
- Iterative Processing.
- In-memory processing.

Apache Flink reduces the complexity that other platforms as Spark faced by means of some improvements. Among such improvements highlight the integration with MapReduce of query optimization, some concepts from database systems and efficient parallel in-memory and out-of-core algorithms. These optimizations are given due to the fact that Flink architecture is designed taking into account the streaming model. Such design improves the micro-batch approach taken in Spark for data streaming processing making Flink faster and with lower latency for such kind of processing.

Following the example of Spark, Apache Flink is provided with three APIs (DataStream, DataSet and Table APIs) in order to use its engine. DataStream and DataSet APIs are two regular programs used by Flink in order to perform transformations on data streams (filtering, aggregating, update state...) and data sets (joining, grouping, mapping...) respectively. On the other hand Table API is used for relational operations and it can be integrated into DataStream API for relational stream processing and into DataSet API for relational batch processing.

Finally, it is important to remark the five features that Apacha Flink presents:

1. Low latency and high performance.
2. Fault tolerance.
3. Memory management.

4. Iterations.

5. Integration.

All these characteristics are what make Flink the 4G of big data and the most powerful tool when dealing with data stream processing.

2.2 Privacy Policies in IT

Computer security, also known as cybersecurity or information technology security (IT security), is the protection of computer systems and networks from people who is interesting in stealing or damaging their hardware, software or the data with which they are working.

Every computer security system accomplish the well-known CIA paradigm. "C" stands for Confidentiality, "I" for Integrity and "A" for Availability and they are the three basic requirements that compose such paradigm. Any computer security system has to grant that any piece of data has to be accessed only by those who are authorized, accomplishing such requirement, information is confidential. Moreover, such systems have to be able to allow data modifications only to those how are authorized for it and only in the way that they are entitled to modify them, this requirement makes information to be integrated. Finally, data must be available to everyone who has a right over them within some time constraints, this makes information to be available. These three requirements raise an engineering problem and it is that availability conflicts with confidentiality and integrity due to the fact that if a data is available then it could not be confidential as anybody may modify it and it could not be integrated as anybody could modify then. This is the main problem that has to be faced when designing computer security problems. In order to handle this complex engineering problem, designers can use legal instruments but also technical tools.

Due to the fact that the information of many entities (persons, companies, etc.) are involve in such complex problem, there are some legal instruments to handle the design of security systems. This is the case of privacy policies. Privacy policies are statements or legal documents that reveal how an entity can collect, use or handle data from another entity. Privacy policies can differ from one country to another this is why there are some agreements in order to handle them. In the European Union (EU) the General Data Protection Regulation (GDPR) is in charge of harmonizing data privacy laws across all member states of the EU. In spite of these regulations, there are many ways in order to apply privacy policies.

On the other hand, when a computer security system is designed, the designer has to put himself in the place of the cyberthief, or threat agent, in order to achieve a secure system. Such cyberthiefs try to find vulnerabilities and, then, exploit them in order to steal some information. Usually, vulnerabilities are a bug that allows to violate the CIA paradigm but the exploit can be a wide variety of vulnerability uses. Despite this design technique, unfortunately, a great amount of vulnerabilities are found when users accidentally deal with the applications once the applications are in the market.

Another remarkable point is the importance of threats, that are potential violations of the CIA paradigm, in computer security systems. This means that an

information system can be seen its security system broken due to the fact that a circumstance or an event could impact on it adversely by means of the break of any of the three basic requirements of a computer security system. A threat is composed by three layers. The attacker or threat agent that is the person or the information system that performs the attack action, the countermeasure layer that is all the systems that try to identify an attack and stop it and, finally, the target of the attack that is on which resides the vulnerability and on the threat consequences take place when the exploit arrives to the vulnerability.

In conclusion, the concept of computer security involves many fields but when dealing with the design of DIAs, it is very important to take into account not only the CIA paradigm but also the design techniques from the threat agent perspective in order to reach secure DIAs, always under the legal framework of privacy policies. Furthermore, what makes security on DIAs reliable is the balance on its conflicting requirements what is a complex engineering problem. Finally, systems with a high amount of vulnerabilities but with no threats, are secure systems. However, systems with many threats and only a few vulnerabilities become in the most attacked systems and, then, the ones that need more security measures.

2.2.1 Privacy Policies in DIAs

Due to the fact that DIAs are constantly managing data from different entities, it is very important the way in which privacy policies are applied on them. The final goal of applying them is that, in case of threats, threat agents cannot obtain such data. This reduces the probability of threats as there is no sensitive data that can be obtained and, then, the CIA paradigm cannot be broken.

When dealing with DIAs, sensitive data are sent in data streams. Such data streams are composed of tuples which can contain a high variety of data. Each tuple of a data stream contains a finite set of data that are always ordered in the same way. Due to the fact that all the tuples contained in a data stream have the same structure, when speaking about data streams, each of the data of the tuple is considered as a field of the data stream. Due to this distinction on the fields of the data streams, data streams can be classified into three different sets:

- Subject-specific streams.
- Subject-generic streams.
- Non-personal streams.

A subject-specific stream is a stream composed of a field referring to the data subject. This means that each tuple of the stream contains a data which is the owner of the tuple. A subject-generic stream is a stream without data subject. However, subject-generic streams are produced by an operator whose input is a subject-specific stream. Finally, a non-personal stream is a stream without data subject and that it has not been produced by an operator whose input is a subject-specific stream.

The context in which a tuple is conveyed is also important. Depending of the context in which data are transferred, an entity could want that the data of the tuple are private or public. This context is specified by means of three variables:

- Observer.

- Role.
- Purpose.

The first variable is the observer of the data, the entity who demands the usability of the data. The second variable is the role that the requesting entity is playing. The requesting entity can be a person but also a company. In the case of the company, many persons working for such company can be demanding the data of the tuple and each person has a role inside the company, this is why the role is an important variable in order to define the context in which a privacy policy can be applied. Finally, the third variable that defines the context is the purpose. The purpose for which the requesting entity demands the data of the tuple. Then, the context in which a privacy policy has to be applied is defined by means of three variables that are called static context variables (SCV) as, once they are defined, they are not going to change and because of this they are static.

Another important distinction is the one given by the data that compose a tuple. Each of these data can be strings or numbers. Because of this, a classification on the type of each data of the tuple, and then on the type of the fields of the streams, has to be defined. Moreover, following the approach given by the SCVs, each of these values are considered as a variable as the values of a field of a given stream can change depending on the tuple that is referred. Then, the variables that compose a field of a stream are:

- Categorical variables.
- Numerical variables.

Categorical variables are those that correspond to a string in any programming language. They can be words but also links or any set of letters with special characters and numbers. The second type of variables correspond to numbers but also to a range which corresponds to a pair of numbers. For example, a data subject can be a categorical variable when it is the name of the user who uses the DIA but a data subject can be also a numerical variable if it is an ID which is compose only by numbers. Furthermore, the set of categorical and numerical variables used by a DIA are called contextual variables.

Taking into account these three classifications, privacy policies can be defined by means of some rules. These rules have to be defined making a relationship between the contextual variable used by the DIA and some values that the user selects in order to encrypt the original fields of the stream. These relationships are made by means of logical operators. In the case of categorical variables, the operators that can be used are: equal to ($=$) and not equal to ($\neq$). On the other hand, in the case of numerical variables, the relation operators that can be used are: equal to ($=$), not equal to ($\neq$), is greater than ($>$), is less than ($<$), is greater than or equal to ($\geq$) and is less than or equal to ($\leq$)

Finally, due to the fact that privacy policies encourage the anonymity of the entity who owns the data of the tuple, depending on the data stream classification explained above and the rules defined by each user, privacy policies can differ. Then, another classification can be made, in this case about the privacy policies in DIAs:

- View Creation Policies (VCPs).
- Data Subject Eviction Policies (DSEPs).

2.2.1.1 View Creation Policies (VCPs)

View Creation Policies (VCPs) are a type of privacy policies for DIAs which are applied to subject-specific streams. VCPs are applied when the context is precisely specified by means of the SCVs and the information contained in all the fields of the stream is observable, which means that it has not been modified before by another privacy policy.

Furthermore, VCPs are defined by means of a set of rules which are relationships between the contextual variables used by the DIA and some values that the user selects in order to encrypt the original fields of the stream. In this kind of privacy policies, the encrypted final values of the stream are defined by means of generalization vectors. A generalization vector defines the value to be used in order to generalize each of the fields that compose a subject-specific stream.

In conclusion, given a tuple belonging to a subject-specific stream, if at any instant such tuple satisfies all the rules defined by means of logical operators relating some contextual variables with a value defined by the user of the DIA, a generalization vector is applied on the tuple in order to generalize each of the fields of the streams and obtaining an encrypted tuple. Then, a VCP is defined by means of four items:

1. Subject-specific stream.
2. Data subject.
3. Set of relational rules.
4. Generalization vector.

2.2.1.2 Data Subject Eviction Policies (DSEPs)

Data Subject Eviction Policies (DSEPs) are applied when given an operator, which input is a subject-specific stream, produces a subject-generic stream. Sometimes, the entity who owns the data which are flowing along the DIA does not want that an operator takes some information from the input stream in spite of the output stream is not showing any information directly pointing to such owner. This is the case in which DSEPs are applied.

Unlike what happens with VCPs, DSEPs do not encrypt the fields of a given data stream. In this case, DSEPs prevent the tuples flowing in the subject-specific stream that inputs into a given operator from being processed by such operator. In this case, the output from the operator is computed without taking into account the data of the data subject who defines the DSEP, preserving the anonymity of the owner of the data for the computation of a certain statistic.

In conclusion, in order to define a DSEP, first of all, a set of rules have to be specified by the user relating some contextual variables with some values that such user has to define. Then, given an instant when all the defined rules are satisfied on a subject-generic stream which outputs from a given operator, all the tuples referring to the data subject who defined such rules have to be evicted from the subject-specific stream that inputs into the given operator. Then, a DSEP is defined by means of three items:

1. Subject-generic stream.

2. Data subject.

3. Set of relational rules.

2.3 Modeling & Metamodeling

A model is a representation of a reality in order to explore, to redesign or to transform it. This representation can be externalized by means of a diagram in a paper or in a computer software. Modeling is the method that has to be developed in order to reach a model. A modeling method consists of a modeling technique and a mechanism or algorithm working on the model. Moreover, a modeling techniques consists of a modeling language and a modeling procedure. A modeling procedure is a sequential process composed of several steps but also an iterative process since the procedure can be repeated many times until the final model is reached. In spite of several modeling procedures can be accomplish in order to create a model, all these approaches share some common steps:

- Purpose definition.
- Boundaries definition.
- Model elements definition.
- Relationships definition.

The first step that has to be accomplished in order to create a model is to define its purpose. With such purpose, the questions that the model will have to answer, once it is implemented, should be specified. Later, the boundaries of the model must be specified. Such boundaries must focus on the reality that has to be modeled discarding any information that is not able to reach the predefined purpose. Once these boundaries are defined, everything that is going to take part of the model is called the model domain. Due to the fact that usually this domain is still very big, each concept belonging to the model domain must be filtered taking into account the relevance of each of them and grouping those concepts that are identical in order to reach the purpose. After that, each of the groups with identical concepts can be transformed into a model element, which is an abstraction of all the concepts that are part of the group. Finally, the relevant relationships among the model elements have to be identified and represented in the model.

Furthermore, a model must be complete in order to be able to answer the question specified in the purpose but also, it must be consistent which entails the lack of contradictions in its representation. In order to reach such characteristics, modeling methods take advantage of modeling techniques in addition to a mechanism, an algorithm, working on the model.

The first component of a modeling technique that is a modeling language is a set of well-known elements and relationships by means of which a model can be represented. A modeling languages is described by a syntax, semantics and a formal notation. A syntax is the set of elements, relationships and rules that can be written in order to represent a model by means of a grammar. Moreover, a syntax can be described by two different types of grammars: graph grammars and metamodels. A

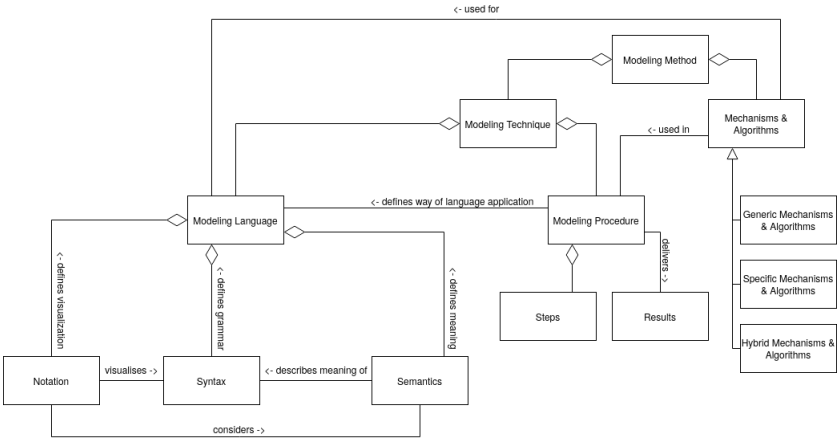

Figure 2.1: Modeling Method Components

metamodel is a model of a modeling language. A metamodel is written by means of a modeling language that is known as the metamodeling language. The metamodeling language has a model which defines it. This is why exists a modeling hierarchy which is not limited and that ends given a useful abstraction level of the model. In the table 2.1 there is an example of a modeling hierarchy with five levels.

Language Level	Models	Language Name
Level 1	Model	Modeling Language
Level 2	Metamodel	Metamodeling Language
Level 3	$Meta^2$-Model	$Meta^2$-Modeling Language
Level 4	$Meta^3$-Model	$Meta^3$-Modeling Language
Level 5	$Meta^4$-Model	$Meta^4$-Modeling Language

Table 2.1: Modeling Hierarchy

The semantics of a modeling language describe the meaning of the language by means of a semantic domain and a semantic mapping.

Finally, the formal notation describes how the modeling language is visualized. Some symbols are used in order to represent the represent the syntax of the language in a diagram.

On the other hand, the finality of the mechanism is to check that the model written with the modeling language by means of the modeling method is correct and satisfies the purpose of the model. There are three main types of mechanisms and algorithms: generic, specific and hybrid.

In the figure 2.1 o model of a modeling method is shown.

2.3.1 Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized modeling language whose objective is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.

UML is able to achieve such goal by means of a wide variety of options to build modeling diagrams. Among the fourteen possibilities that UML provides to

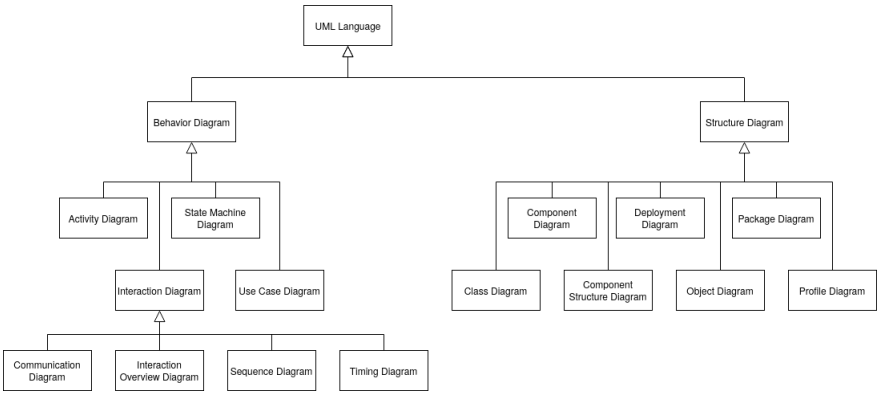

Figure 2.2: UML Metamodel

construct a diagram, a distinction between two main classes can be made: behavior diagrams and structural diagrams.

Behavior diagrams represent the dynamics of a system, this means the representation of all the possible changes that a system can experience over time. This class is composed of seven diagrams: activity diagram, communication diagram, interaction overview diagram, sequence diagram, state machine diagram, timing diagram and use case diagram.

On the other hand, structural diagrams represent the different static parts of the system, that ones which do not change over time. This representation can be done from different abstraction and implementation levels and making the corresponding relationships between the parts of the system. This second class is composed of other seven possibilities for diagram construction: class diagram, component diagram, component structure diagram, deployment diagram, object diagram, package diagram and profile diagram.

A UML diagram making a description of all the UML diagrams can be seen in the figure 2.2.

2.3.1.1 UML Diagrams

In spite of UML allows developers to use fourteen diagrams for modeling, some of such diagrams are the most commonly used. This is the case of the following ones.

Class Diagram

This kind of diagram represents the model elements and the relationships that exist between such elements. The model elements, which are represented by means of nodes, can be: class, component, data type, interface, model or package among others. Regarding to the relationships between the nodes, they can be: association, association class, dependency, generalization or information flow among others.

Class diagrams are commonly used to represent the metamodel of the syntax of a modeling languages. Due to the fact that this kind of diagrams is not able to fully express syntactical rules, usually they take advantage of constraint languages as Object Constraint Language (OCL).

Profile Diagram

A profile diagram allows to create the model domain by adding model elements called stereotypes and making the corresponding relationships between them. The most commonly used relationships in this kind of diagrams are the composition and the generalization.

2.3.1.2 UML Frameworks

Due to the UML popularity among software developers, there are several frameworks working with it in order to make software development easier. Different platforms can be used for UML modeling but also platforms for converting a model to a source code can be found.

Papyrus

UML can be found in a wide amalgam of platforms due to its utility for modeling. Moreever, there are some platforms that are the most commonly used for software development. This is the case of Eclipse. Eclipse is the most popular Java Integrated Development Environment which allows a large amount of plugins in order to increase its functionalities.

One of the plugins provided by Eclipse is Papyrus, an open-source UML tool. In spite of Papyrus was developed by the French Alternative Energies and Atomic Energy Commission (CEA-List), currently it is one of the most used UML tools for software development as it can be expanded with UML profile diagrams giving a wide functionality for modeling software applications.

Acceleo

Regarding to platforms for converting an UML model into source code, Acceleo is an open-source generator from Eclipse Foundation. Acceleo is one of the Eclipse plugins that allows to transform UML models into source code files. It is written in Java and it is available for Linux, Windows and Mac OS.

Acceleo uses its own language in order to generate the source codes. Such language is based on MOF Model to Text Transformation Language (MOFM2T) and on template focusing. A template contains some text which describes the data that should be extracted from the elements of the model and such data are extracted iteratively from each component of the model. Moreover, the descriptions of the information that should be taken from the model elements are written in OCL language.

Finally, it is import to remark that Acceleo can extract the information from different types of models, such as EMF, UML or DSL and it is able to generate different source languages such as C, Java or Python.

2.3.2 Object Constraint Language (OCL)

Object Constraint Language (OCL) is a language to describe formal expressions in UML models. These expressions represent invariants, preconditions, postconditions, initializations, guards, derivation rules, as well as queries to objects in order to determine its state conditions.

Initially, OCL was developed by IBM. But, then, in 2003, OCL was adopted as part of UML 2.0 by the group OMG.

This language does not cause side effects, so the verification of a condition, which presupposes an instantaneous operation, never alters the objects of the model. Its main role is to complete the different artifacts of the UML notation with formally expressed requirements.

In conclusion, OCL associates boolean expressions with the elements of the model and these expressions are evaluated as true every time the program is executed. In addition, they can also be associated with methods. In this case, the Boolean expressions are evaluated as true at the moment before or after the execution of the program.

Chapter 3

Requirements and Design

3.1 Requirements

The first requirements that are needed in order to develop the big data application is to install the corresponding versions of Eclipse, Papyrus and Acceleo. In spite of the application is able to work with other versions, some problems could appear because of the changes that they present. The models implemented in this document have been developed with the following versions:

- Eclipse 2019-03 (4.11.0).
- Papyrus SysML 1.4 Feature 1.3.0.
- Acceleo 3.7.8.201902261618.
- m2e-Maven Integration for Eclipse (includes Incubating components) 1.11.0.20190220-2119.
- Apache Flink 1.4.0.

3.2 Design

3.2.1 Great Seller DIA

Along this document a DIA for an e-commerce company called Great Seller is implemented. This DIA is able to compute real time statistics about the transactions generated by the consumers. Such statistics can be observed by some other companies that pay in order to have access to the information making that Great Seller acts as a data broker. In order to simplify the implementation, Great Seller is going to sell three different types of statistics:

- Statistic 1: the total amount of money that each consumer of Great Seller spend in the last 10 minutes.
- Statistic 2: the number of transactions issued by each user in the last 10 minutes.
- Statistic 3: the number of users who spent more than €1000 in the last hour.

As any DIA, Great Seller generates such statistics by some transformations which are fed from a source and the generated information is stored into a sink which is accessible by the observer companies. Moreover, as Great Seller is producing three different types of statistics, its DIA requires three different sinks where the information should be stored. Due to the fact that Great Seller DIA is computing real time statistics about the transactions that are generated by the consumers of the company, only one source is required for the DIA model. Finally, one transformation is necessary for the computation of each statistic. In summary, Great Seller DIA requires one source, three transformations and three sinks for the design of its model.

3.2.1.1 Great Seller Source

In the implementation described along this document, Great Seller DIA is fed from a text file where all the transactions generated by the consumers of Great Seller are stored. All the transactions are going to have the same predefined tuple structure 'transactionId,dataSubject,spentAmount,purchasedProduct'. This predefined structure is represented in the Great Seller DIA model by means of a Data Type called InputTransaction.

Regarding to the fields of such tuple, the transactionId field is an integer which varies from 1 to the number of generated transactions, such number can be 10, 100 or 1000. The dataSubject is the user who generates such tuple and it is one of the following: Bob, Carlos, Elisabetta or Michele. The spentAmount is the price paid for the product bought with the transaction and it is an integer with a low boundary of €1 and an upper boundary of €200. Finally, the purchasedProduct is the product bought with such transaction.

The Great Seller stock is composed by means of 25 products. In order to simplify the implementation, each product is named by the word 'product' immediately followed by a number between 1 and 25 in order to specify the product referred in the stock. In the table 3.1 can be seen such stock.

3.2.1.2 Great Seller Transformations

In order to compute each of the three statistics that Great Seller sells as data broker, the Great Seller DIA needs three transformations. Each of this transformations is the operator of each statistic. Thus, the operator one (OP1) computes the total amount of money spent by each user in the last 10 minutes. The second operator (OP2) computes the number of transactions issued by each user in the last 10 minutes. Finally, the operator three (OP3) computes the number of users who spent more than €1000 in the last hour.

These transformations input a data stream with a set of tuples that all of them have identical structure. Thus, the first stream (S1) is composed of tuples of the kind InputTransaction. This stream is duplicated and it is sent to the operators OP1 and OP2. Moreover, as each operator works taking into account the data subject of each tuple, the stream S1 must be keyed by the data subject field of this first stream. Finally, as the operators have to compute the statistics taking into account only the tuples generated in the last 10 minutes, the stream S1 is windowed by a time window of 10 minutes.

The new data generated in the OP1 are represented in a new data type called SpentAmount and that is composed by two fields following the structure: 'dataSub-

Product	Price (€)
product1	196
product2	36
product3	179
product4	17
product5	120
product6	187
product7	139
product8	52
product9	160
product10	110
product11	113
product12	67
product13	100
product14	125
product15	192
product16	115
product17	113
product18	98
product19	113
product20	185
product21	143
product22	18
product23	194
product24	41
product25	26

Table 3.1: Great Seller Stock

ject, totalAmount'. The SpentAmount tuples are collected in the stream S2 and they are keyed by the data subject field of such stream.

Finally, the data generated in the OP2, is collected in a new data type called IssuedTransaction whose structure is 'dataSubject, nTransactions'. This new stream (stream S3) is sent to the OP3. Moreover, this third stream is keyed by the data subject field of the stream and it is also windowed with a time window of one hour. The OP3 generates a randomly partitioned stream with an uniform distribution called stream S4 whose structure is: 'nTopUsers'.

3.2.1.3 Great Seller Sinks

Each of the tuples flowing through the DIA must be stored in a file, in order to be able to access to the tuples that will be sent to the observer companies. This is why, after each generated stream, a file text sink is implemented with different paths in order to store each of the flows obtaining all the results in three different files. The file with the results of the stream S2, the stream S3 and the stream S4. This implementation is given due to the fact that each of the tuples generated by the operators are observable for the observer companies who buy them.

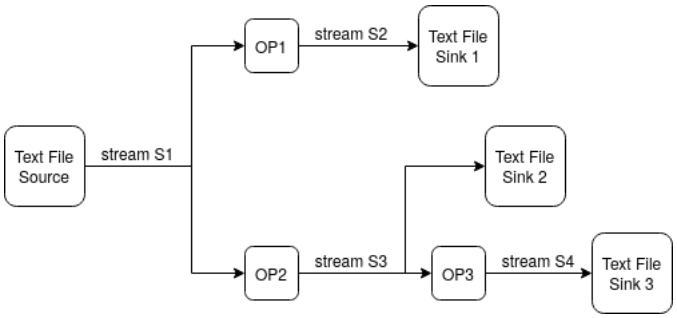

Figure 3.1: Great Seller DIA

In the figure 3.1 the dataflow model of the Great Seller DIA without privacy policies can be seen.

Chapter 4

Solution

In order to implement the application, first of all, a new project has to be created in Eclipse. This project is going to be a Papyrus project (figure 4.1).

The first field that has to be specified is the Architecture Context. Here, the default values are the right ones as an UML model is what is required to implement with Papyrus (figure 4.2).

After that, the name of the project has to be defined. Due to the fact that it is going to implement an application called Great Seller, the name of the Papyrus project is going to be greatSellerApp (figure 4.3).

The next step is to select the representation kind and to choose the StreamGen profile. StreamGen requires Papyrus to make a class diagram representation and, as the StreamGen project is already in the Eclipse workspace, the profile can be found browsing in the workspace (streamgen/profile/StreamUML.profile.uml). As it can be seen in the figure 4.4.

Finally, we can finish with the new Papyrus project initialization (figure 4.5).

The next step is to define the model in the class diagram in order to specify that the application is going to be a Flink application. Then, first of all, we have to create a model node in the greatSellerApp.di file. This node is going to be called greatSeller as it is the object representing the whole application. Once the node is inserted, the properties of such node must be specified. More in detail, the Flink application stereotype has to be applied. In this case, the properties of the model has to be the default ones then nothing else must be done with the model node.

The next step is to add the data types that the application is going to need. In order to do this, a package node has to be inserted inside the Flink application model node. This package is called greatSellerDataTypes and a stereotype has

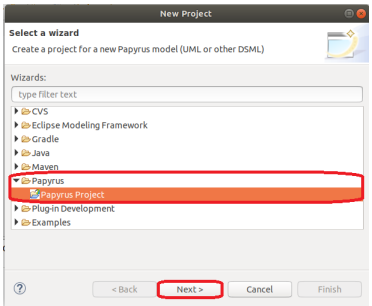

Figure 4.1: New Papyrus Project

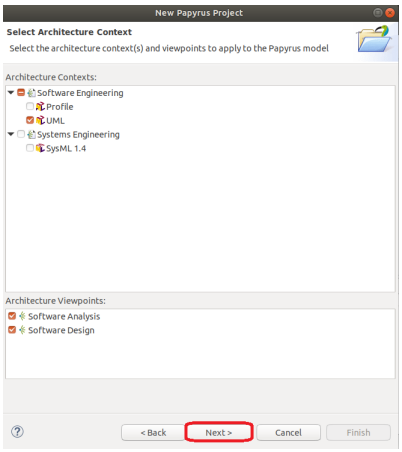

Figure 4.2: Architecture Context

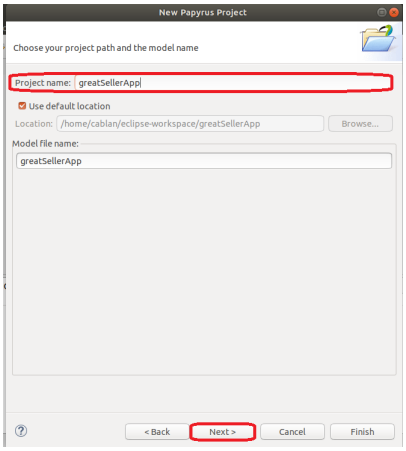

Figure 4.3: Project Name Definition

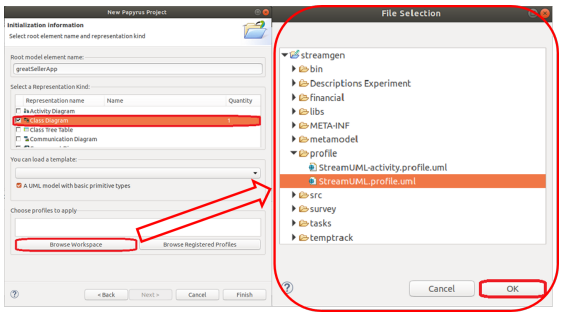

Figure 4.4: Representation Kind

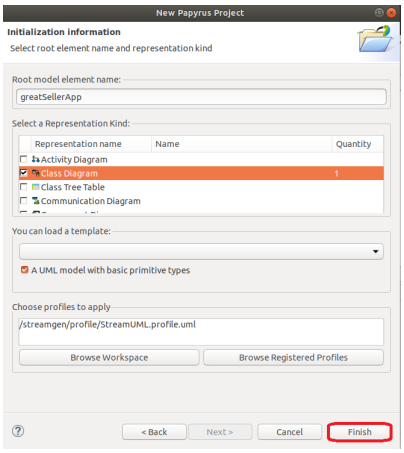

Figure 4.5: Papyrus Project Definition

to be applied as in the previous case. First of all, inside the properties window, in the profile field, the applied stereotype is defined. Inside this package all the datatypes that the application requires have to be inserted. The first data type is the InputTransactions which is composed of a transaction id (integer), a data subject (string), an amount (double) and a recipient id (string). The second data type is the IssuedTransactions which contains the data subject (string) and the number of transactions performed by such data subject which can be seen by means of the variable NTransactions (integer). The third data type is the SpentAmount which is composed of the data subject (string) and the total amount of money spent by the data subject which can be seen by means of the variable TotalAmount (double). Finally, the last data type is the NumberUsers which contains the number of users who spent more than 1000 dollars and it is represented by means of the variable NTopUsers (integer). Then, we need to insert one DataType node inside the package created in the previous step with each of these data types. Each data type can be seen as a tuple which is composed by several values. The name of the DataType node is going to be the same that the one corresponding to the tuple and each of the values that compose the tuple are going to be a property inside of the owned attributes that are specified in the UML field of the properties of each DataType node.

DataType	Properties	PropertyType
InputTransactions	transactionId dataSubject amount recipientId	integer string double string
IssuedTransactions	dataSubject nTransactions	string integer
SpentAmount	dataSubject totalAmount	string double
NumberUsers	NTopUsers	double

Table 4.1: DataTypes Composition

4.1 Files Generation

4.1.1 Source Files Generation

In order to generate the text file that feeds the Great Seller DIA, two Python codes have been developed. In addition, a Map Transformation is implemented in the DIA in order to split the input strings generated by the Python codes and to generate the InputTransaction data type.

The first Python code builds a Python list which represents the Great Seller stock of 25 products and it assigns to each product a price. Each product is represented with a dictionary variable that contains a name variable and a price variable. The name variable is a string with the word 'product' immediately followed by an integer number from 1 to 25 which points to the product of the stock. The price is an integer number between $1 and $200 which is assigned randomly. Once the name and the price are assigned to the dictionary, the product is added to the stock list. Finally, the Great Seller stock list is saved in a binary shelf file in order to be accessible from the other Python code.

The second Python code generates the strings that represent the tuples produced by each consumer. In order to generate such tuples an integer number for the transactionId is assigned following an increasing numerical order from 1 to the maximum number of generated transactions which is input by command line to the code, it can be 10, 100 or 1000. After that, randomly, one of the four possible data subjects (Bob, Carlos, Elisabetta and Michele) and a product from the stock saved in the binary shelf file are assigned. Finally, each value is added to a string where each of these values are separated by a comma. The generated string represents the tuple produced by each consumer and it is written in the text file which is called from the Great Seller DIA in order to feed it.

Once, the DIA inputs each of this strings in a non-parallel stream, a Map Transformation, taking advantage of the split Java method, splits the strings by the comma generating the InputTransaction data type which is composed of four fields: transactionId, dataSubject, amount and product.

4.1.2 Java Codes Generation

In order to generate the Java codes from the Papyrus model, first of all we need to create a new Java project in Eclipse (File -> New -> Java Project). This project is going to be called 'greatSellerCodes' as it can be seen in the figure 4.6.

Once the Java project is created, its structure has to be modified in order to allow to be compatible with Maven. As the figure 4.7 shows, firstly the Java project is composed of a JRE System Library and a src folder. This src folder has to be deleted, leaving the Java project only with the JRE System Library as it can be seen in the figure 4.8.

The next step is to create a source folder with a predefined structure as it is going to be specified now. Firstly, in the properties for the project, in the Java Build Path field, a folder has to added as the figure 4.9 specifies. In the default output folder a specific structure has to be written, greatSellerCodes/src/main/java. After this is written, the changes have to be applied and then click on 'Add Folder...'.

In order to create the source folder, the java folder has to be selected and then click on 'OK' as it is shown in figure 4.10. Then the changes have to be applied and,

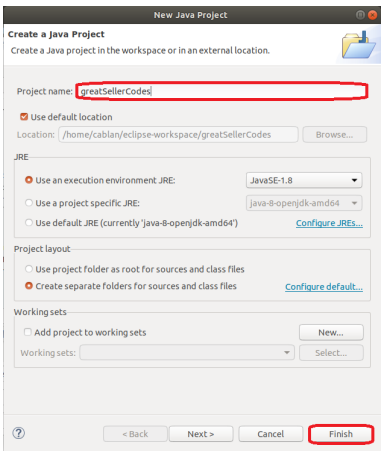

Figure 4.6: Java Project Creation

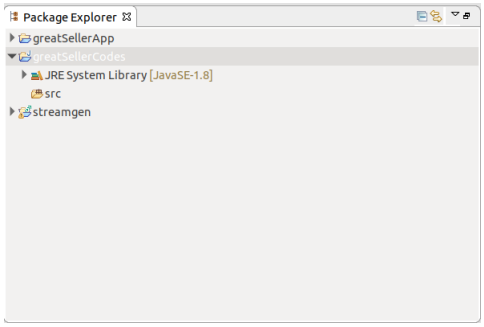

Figure 4.7: Java Project Initial Structure

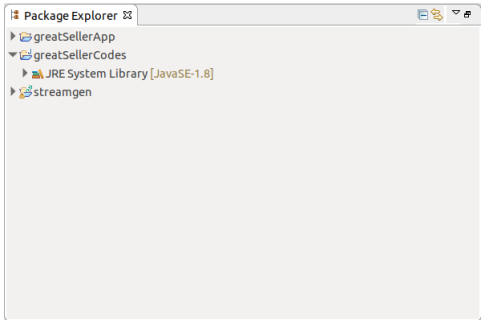

Figure 4.8: Java Project Final Structure

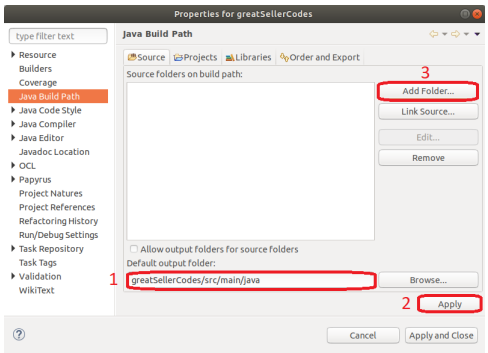

Figure 4.9: Maven Project Structure Creation

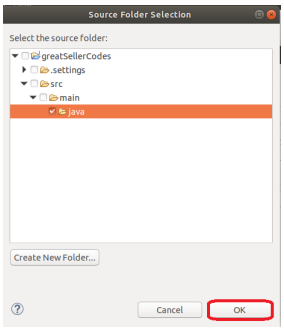

Figure 4.10: Maven Project Source Folder Creation

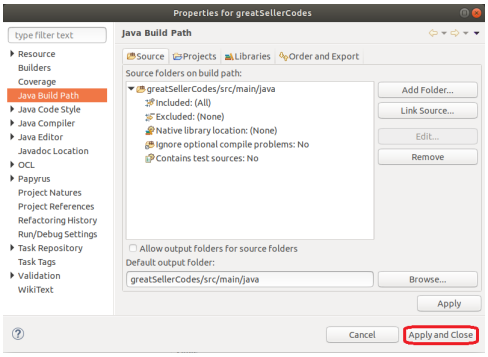

Figure 4.11: Maven Project Properties

then, the window Properties for greatSellerCodes has to be closed (figure 4.11).

After these steps, the project folder should look like it is shown in the figure 4.12 at the Package Explorer. And then, the project is ready to convert it into a Maven project. In order to do this, going to the 'Configure' option of the project and then clicking on 'Convert to Maven Project'. Then, a POM file has to be created and it is going to be created the default one as it can be seen in the figure 4.13. At this point, the structure of the project has to be the same that the one shown in the figure 4.14.

Now it is time to run the configuration (Run -> Run Configurations...). In the figure 4.15 is shown how this window has to be filled.

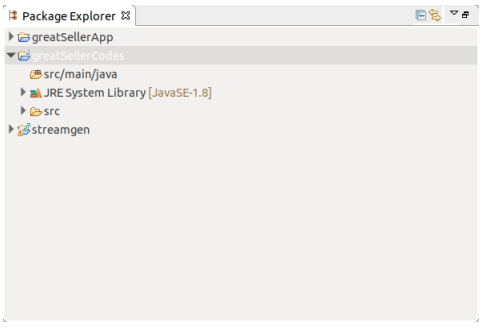

Figure 4.12: Maven Project Structure

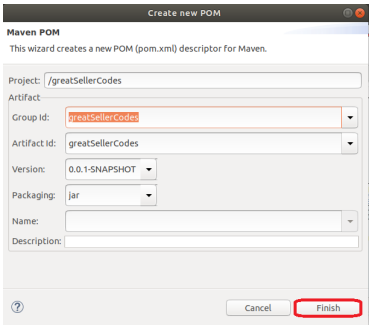

Figure 4.13: POM File Creation

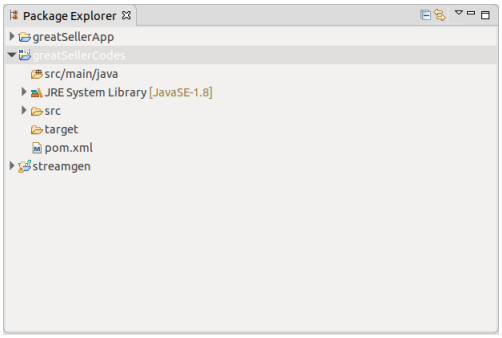

Figure 4.14: Final Maven Project

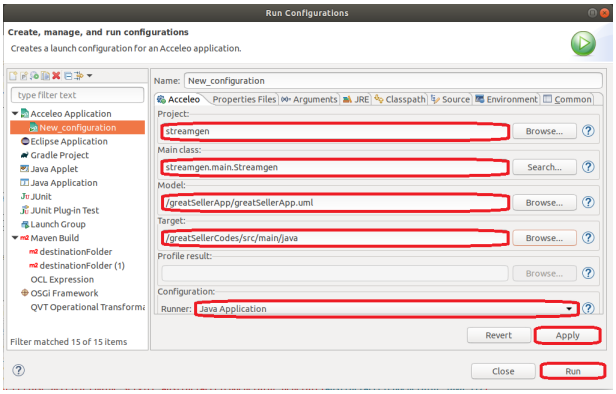

Figure 4.15: Run Configuration

Chapter 5

Evaluation

Chapter 6

Conclusion

Bibliography

[1] Hadoop Tutorial
`https://data-flair.training/blogs/hadoop-tutorial/`

[2] Spark Tutorial
`https://data-flair.training/blogs/spark-tutorial/`

[3] Apache Flink Tutorial
`https://data-flair.training/blogs/apache-flink-tutorial/`

[4] Michele Guerriero, Damian Andrew Tamburri, Elisabetta Di Nitto. *Defining, Enforcing and Checking Privacy Policies In Data-Intensive Applications*. 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2018.

[5] Modeling Tutorial
`https://www.transentis.com/methods-techniques/models-and-metamodels/`

[6] Unified Modeling Language Tutorial
`https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/`

[7] Dimitris Karagiannis, Harald Kühn *Metamodeling Platforms* Springer-Verlag, 2002