Diploma Project

PPA: A FRAMEWORK FOR DEFINING AND DESIGNING FORMAL PRIVACY POLICIES

Marios Eftychiou

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2019

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

PPA: A FRAMEWORK FOR DEFINING AND DESIGNING FORMAL PRIVACY POLICIES

Marios Eftychiou

Diploma Project Supervisor

Anna Philippou

The current Diploma Project was submitted for partial fulfillment of the requirements of obtaining a degree of Computer Science of the Department of Computer Science of University of Cyprus.

May 2019

Acknowledgment

I would first like to thank my thesis advisor Dr. Anna Philippou .The door to Prof. Philippou office was always open whenever I ran into a trouble spot or had a question. She consistently allowed this paper to be my own work but steered me in the right direction whenever she thought I needed it. I would also like to thank her for the opportunities that she gave me not only through this thesis but also with other projects and activities the last four years.

I would also like to thank the experts who shared their knowledge with me and guided me for this research project: Dr. Dimitris Kouzapas and Mrs. Evangelia Vanezi. Without their passionate participation and input, this thesis could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Summary

In this thesis we address a main problem of our time, namely privacy, and specifically privacy policy languages that describe how systems and companies handle our data. Additionally, we have overviewed existing languages, most of them old and obsolete and decided to introduce our own formal privacy policy language based on another one, namely Privacy Calculus. The necessity for this was mainly the lack of mathematical formal languages, that can be later used for verification such as model checking, and the complexity of most languages to declare an enormous system of today. The main difference and novelty, between our proposal and languages that already exist is the dynamicity of policies. Opposed to most languages, the user can declare a set of policies, not only one, and match them with some actions. This modelling of our language help us track the policy that the system is currently at, by knowing the sequence of steps or actions that occurred. In addition to that, dynamicity of policies gives us the features of composition and analysis of policies. Moreover, we have built a framework that accompanies the language and helps the designers of systems to declare the privacy of their system in a graphical and easy way. This framework helps designers declare the privacy incrementally, an entity's policy at a time, giving this framework a simplicity against already introduced languages. After the graphical declaration the tool is responsible to produce the global policy by merging the partial policies to one global policy of the system with the help of some structures and algorithms. Moreover, we transform the global privacy created to some other forms such as text and graph to be used later for more formal checks like model checking, static checking for compliance with today's regulations, and monitoring. As we will see further on in this thesis, creating all the above will not only support policy designers but it will also help developers and companies to be assured about the policy that their systems follow and even spot vulnerabilities and breaches while the system is running.

Contents

Chapter 1

Introduction

1.1 Importance and purpose of the research

Living in the Big Data era, with tons of data transmitted every day and an expectation of 50 billion online devices by 2020, privacy has become a major concern of our society. The EU General Data Protection Regulation which came into effect a year ago, May 2018, forced industry to also take that concern more into account, with stricter rules and more considerable penalties that cannot be overlooked. With all this into consideration we can easily understand that systems from now on should be designed with these and even future regulations in mind as first rule and the already implemented systems must comply with them.

In order to achieve the above industry spends an big amount of money and man-hours to read, comprehend and then follow these rules. The main problem for that is the huge gap between the legal experts who read the rules and write the policy of the system and the developers of the system who need to follow the guidelines of the legal team.

An easy way to close that gap is the development of a language or tool that can be easily understand by both groups of experts. Further on we will see some efforts of researchers, trying to do so, with the creation of new languages and frameworks. Afterwards we will discuss the creation of a new basic privacy policy language and a tool inspired by previous

work, that will try to be as friendly as possible to the user creating the policy, without lacking any formality in order to keep alive the concept of formal checking for the future.

The Privacy Policy Assistant tool (PPA) will give the opportunity (to user) to access a graphical interface and complete some actions such as design the policy in a user-friendly environment, produce a text-written policy and produce a graph of the policy for the developing team, making their life easier as we will see later on.

1.2 Methodology

At the beginning a variety of research papers about Privacy Policies and their enhancing role on today systems was studied, to fully understand the notion of Privacy Policy and how these policies are enforced inside companies or systems. The General Data Protection Regulation (GDPR) was examined next to help us get informed about the regulations and main principles a Privacy Policy should describe. Research concluded with the study of newly introduced formal Privacy Policies frameworks as enunciated by Kouzapas & Philippou[9], Diver & Schafer [6] and Fernández,Jaimunk & Thuraisingham [8], with the concept of checking either for the compliance of a policy with a regulation or for particular properties of the policy.

Later on, a new formal Privacy Policy language was introduced based on Privacy Calculus as it is described in [9] with the ability to define the policy for each role of the system than a whole system policy. Subsequently a part of PPA was created to help users design and comprehend easier the parts of policy. After the creation of previous part, a parser and an algorithm based on parallel composition of time-automata [4] was built to synchronize the partial policies of the subsystems or roles to a bigger, global policy. At the end, a tool for representing the policy as a graph was created to help future work of property checking via model checkers, monitoring or static check of the policy for template compliance e.g. GDPR.

1.3 Structure of the thesis

In this paragraph the structure of the thesis and the main idea of the following chapters will be given. In chapter 2, a review on previous work and mainly privacy policies is given, examining their properties and features and reason why we had to built our own language. In chapter 3, our formal privacy policy language is introduced, with the provision of some examples in order to be comprehended easier and later on in chapters 4 and 5 , the tools for designing the graphical policy and synthesize the partial automata are described, respectively. In chapter 6, the author suggests some proposals for future work, that will have a high impact on privacy policies and in the final chapter some conclusions for the current thesis are taking place.

Chapter 2

Related Work

2.1 Privacy Policy

A Privacy Policy is a legal document that includes a set of statements which set rules for the handling of personal data of a legal person. More specifically, a privacy policy, in the context of IT, is a document that tells readers how a technology or other product or service will use their personal information.

A typical privacy policy restricts the use of personal information to an explicit list of purposes. These restrictions refer to access of a certain purpose e.g. read ,on the private data of a legal person, such as name of student, by an agent with a certain role such as a teacher.

Privacy Policies can be categorized as role-based policies and attribute-based policies. role-based policies are policies defined around roles and privileges. A role can be created for an entity or a group of people. The permissions to perform certain operations are assigned to specific roles. System, student or server are assigned particular roles, and through those role assignments acquire the permissions needed to perform particular system functions.

On the other hand attribute-based policies are policies whereby access rights are granted to users through the use of policies which combine attributes together. This model supports Boolean logic, in which rules contain "IF, THEN" statements about who is making the request, the resource, and the action.
For example: IF the requestor is a manager, THEN allow read/write access to sensitive data.

The main difference between the two is that the latter provides dynamic, context-aware and risk-intelligent access control to resources allowing access control policies that include specific attributes from many different information systems to be defined to resolve an authorization and achieve an efficient regulatory compliance, allowing enterprises flexibility in their implementations based on their existing infrastructures.

2.1.1 Review of current Privacy Policy Languages

As a first step, we did a research for existing privacy policy languages based on [10], presenting a brief review of the main languages studied, including their purpose, and their negative and positive elements.

Sophisticated ACL	Enterprise	SAML, XACML, XACL
	User	XACML
Web	Enterprise	P3P
	User	APPEL, XPref
Enterprise		CPExchange, PRML, E-P3P, EPAL, DPAL
Context Sensitive	Enterprise	Geo-Priv, Rei
	User	Geo-Priv

Table 1: A summary of Privacy Policy Languages by Kumaraguru et al in[10]

Some Privacy Policy Languages are:

1. P3P [16]

The Platform for Privacy Preferences Project (P3P) is an emerging industry standard that enables websites to express their privacy practices in a standardized format that can be automatically retrieved and interpreted by user agents.

2. APPEL, Xpref [2]

As said in [2]:

“The designers of P3P simultaneously designed a preference language called APPEL to allow users to express their privacy preferences, thus enabling automatic matching of privacy preferences against P3P policies”

Basically, these languages are equivalent to P3P but from the user’s perspective.

3. Security Assertion Markup Language (SAML) [14]
   eXtensible Access Control Markup Language (XACML) [15]
   XML Access Control Language (XACL)

All of them are similar XML-based semi-formal languages with XACML the one most commonly used among them.

XACML is a general-purpose access control policy language which provides an XML defined syntax for managing access to resources.

4. PRML (Privacy Rights Markup Language) [17]

PRML is an XML based semi-formal language. PRML is a privacy-aware access control language. A PRML policy, called PRML declaration, contains of the linking of objects. Objects in PRML can be roles, operations, data groups, subjects, purposes, constraints, actions and transformations.

5. EPAL (Enterprise Privacy Authorization Language) [3]

EPAL is a formal language created for representing the internal privacy policies of organizations and to control the internal actions adherence to them. EPAL is an XML-based formal language based on PRML.

In order to comprehend better those languages some examples are provided below.

Example: The following policies describe that <<User1 role is authorized to read the 'contents' data attribute, but not authorized to write them>>.

XACL:

```
<contents>
 <entry>
  <name>User1</name>
  <officeTel>111-1111</officeTel>
  <homeTel>123-4567</homeTel>
 </entry>
<contents>
<policy>
 <xacl>
  <object href="/contents"/>
  <rule>
   <acl>
    <subject>
     <uid>User1</uid>
    </subject>
    <action name="read" permission="grant"/>
    <action name="write" permission="deny"/>
   </acl>
  </rule>
 </xacl>
</policy>
```

PRML:

```
<declaration-set>
 <declaration>
 <oid>DECL-1</oid>
 <name>USER-READ</name>
 <role-idref>#USER</role-idref>
 <operation-idref>#OPERATION-READ</operation-idref>
 <data-set>
 <data>
 <data-group-idref>#CONTENT</data-group-idref>
 </data>
 </data-set>
 </declaration>
```

EPAL:

```
<rule id="rule1" ruling="allow">
<data-user id="User1"/>
<data-category id="contents"/>
<action id="read"/>

<rule id="rule2" ruling="deny">
<data-user id="User1"/>
<data-category id="contents"/>
<action id="write"/>
```

2.1.2 Evaluation of existing languages

As we will discuss further , existing languages have some potentially useful features but also some properties, that make them unsuitable for us to use them furthermore for the creation of a new formal privacy policy language, easily understandable both from the privacy policy writer and the developer perspectives.

- A P3P statement lacks the element of Role and Operation because it was designed to provide a way for a website to encode its data-collection and data-use practices in a machine-readable XML format as it is described in [1].
- P3P lacks robustness as it is described in [2].
- XACML is complex in some ways and verbose.
- “EPAL forces consistency by terminating evaluation mid-policy, sacrificing safety, local reasoning, and closure under combination” [5] .
- As explained in EPALs technical specification [3]: “There are certain situations for which EPAL is not intended as an appropriate language. EPAL was not designed for the following applications: “Directly encoding or enforcing specific privacy legislation in a generic and completely application and enterprise independent way”

These languages were not created to be user-friendly nor to be formal enough in order to check them against regulations, primarily due to the fact than none of these languages except EPAL is formal.

2.1.3 Formal-methods based languages

As we have discussed, scientists moved away from the text-based, non-formal policy languages, which have the main problems that natural language has, such as ambiguity and tried to create more formal structures or languages to better describe privacy policies.

Most times with these formal languages or structures, they are trying to achieve more things than just the description of a privacy policy, such as analyzing the policy to obtain a clearer understanding for the developers or checking properties of the policy to verify a system's compliance against a regulation by model checking or monitoring techniques .

2.1.3.1 Graph-based approach

In [8] the authors are using a graph-based structure to visualize Category Based Data Collection (CBDC) policies. Through that, they are able to analyze the policy with the use of queries by obtaining policy information at the same time as the query is established. As we can see, the above proposal is at *prima facie* a closer approach to developers , helping them analyze and understand better the policy.

By moving closer to the developers, the model of the CBDC policy is not easily understandable to a legal team, which is more probably to be authoring the policy. Another reason that is difficult to use this language is the difficulty of the language to model a full working complex system and the lack of scalability due to non-combined policies, that forces the user to write a new big and complex policy each time, even for a small change.

2.1.3.2 Petri Nets approach

A different approach proposed for defining privacy policies can be found in [6], gives a privacy by design proposal with the use of Petri Nets, to interlink software and the legal model in order to bring closer the two parties involved in the creation of a system. The benefit of that linkage is that it gives designers a powerful opportunity to test the compliance of their designs against data protection laws or certain templates, at an early

stage in the product development process, so that wasted investment can be kept to a minimum.

Another advantage in the use of Petri Nets in comparison to other structures, is the provision of a balance between modelling power and analyzability. In other words, many things one would like to know about concurrent systems can be automatically determined for Petri nets although other properties such as reachability, liveness and boundedness can be hard to decide.

As we can see this approach is far more eligible to solve the problems that we have displayed before, with our main concern to be that two Petri Nets need to be designed. One displaying the legal net while the other describing the software net with the eligibility to interconnect and communicate which rerise again the concern of time and work invested for the design of a single policy and the extensive need for communication between the two parties.

A better solution to that problem would be the design of a privacy policy in a more easy and non-software structure, which can be written by the legal team ,with a small collaboration with the developers' team, and then it could be translated to another structure clearer for the developers. In that way, both parties can get a solution closer to their field without the need of communication after the designing phase.

2.1.4 Privacy calculus

The Privacy Calculus as it is introduced in [9] gives a complete formal framework based on the π-calculus with the extension of groups for studying privacy. Keeping that in mind and trying to construct a new formal privacy policy language, we can use the basis of the Privacy Calculus and we can extend their Privacy Policy to introduce a new formal policy.

In order to introduce the language later, we will now discuss the policy language introduced in the Privacy Calculus, with the provision of some examples. In the current thesis we are only taking into account the use of private data between parties so every piece of information that can be characterized as non-private will be excluded.

At first, we can easily see that the Privacy Calculus policy language is a role-based privacy language where the existence of a permission on private data expresses what a certain role can do on the data, and the absence of it, the lack of that possibility. The notion of role and private data are free for the user to complete as he likes, where permissions are standardized in order to be type-checked later. Further on, we will present the permissions of the Privacy Calculus that will be used in our language, with slight modifications, in order to either become easier for the user, or to fit better in the proposed language of next chapter.
The permissions that will be used in this thesis are the followings:

1. Permission “read”:
   When associated with a type of private data and a role, it indicates that the private data may be read by processes belonging to the specific role. Same as in Privacy Calculus.
2. Permission “update”:
   Gives the possibility of updating the contents with a new piece of private data. Same as in Privacy Calculus.
3. Permission “disseminate(E)”: when associated with a type of private data, it indicates that the private data may be disseminated to an entity E. The permission proposed in Privacy Calculus “disseminate(E,λ)” also encloses λ which is the amount of times the private data is disseminated to entity E. Due to the nature of the proposed language in the next chapter the use of λ will be omitted.
4. Permission “usage(p)”:
   Defines the right to match private data against constants of type p. Same as in Privacy Calculus.

Let us now give an example of the privacy policy language. For consistency we will use the same example as the one given in the previous section.
<<User1 role is authorized to read the ‘contents’ data attribute, but not authorized to write them.>>

User1 >> contents{read}

As we can see the latter example is far shorter, expressing the same clause, as the previous ones. This can help us design a smaller and more simple language for end-users than the ones described before.

Chapter 3

A formal Privacy Policy Language

In this section we will introduce the new formal privacy policy language. The language is based on the Privacy Calculus and more thoroughly on the notions described in the previous section. We also take into account the newly-introduced privacy policy languages of Chapter 2 and the structures that they use to take advantage of their features for future work. More specifically we keep the idea of the graphs from the graph-based approach of the previous chapter, as also the formality that the authors of [6] tried to implement with the use of the Petri Nets.

3.1 Definition of the language

As we said before a typical privacy policy restricts the use of personal information to an explicit list of permissions and these restrictions refer to access for a certain purpose private data of a legal person. In our privacy policy language, the notion of restriction is combined with a set of actions to describe a more flexible global privacy policy which can select the active policy of the system, changing between a set of declared policies. In addition, even though we introduce a larger language in the aspect of terms than the one we were based on, the simplicity of the original language remains.

Privacy Policy Language

The privacy policy language consists of :

1. An owner entity
2. A finite set of non-owner entities

The discrimination between an owner entity and non-owner entities is that the certain privacy policy describes only the handling of the owner entity private data, and thus no handling of private data for non-owner entities will be described in the policy.
We will refer to the owner entity of the policy as OE and the non-owner entities as NOE.As such, we write Entities E=OE ∪ NOE and we refer to members of the set Entities by E(i).

3. An automaton K(i)∀E(i) which is a tuple K(i)= (Qi,Ai,Ci,Pi,q0i,p0i,Li,Ri) consisting of:

i. Q is a finite set of states with q0: ∈ Q the initial state

ii. A: is a finite set of edges or actions A=(a,c) which consists of the name of the action a and a condition c

iii. C is a finite set of conditions

C -> true|false| X op V | (X op V) && C:

a. X is a type variable which can take the value of a private data type.
b. Op is a logical operator between {=,≠}.
c. V is a constant of a certain private data type.

iv. P is a finite set of policies with p0∈P the initial policy.

In order to express the policies, we employ the following entities based on the Privacy Calculus :

Policy-> private_data{Permissions}Policy | ε
Permissions-> Permission | Permission, Permissions
Permission -> read | update | disseminate() | usage()
(private_data : owner's private data)

v. L: Q →P is a function that connects every q∈ Q with a p∈ P.

($L(q_0)\rightarrow p_0$) The initial policy relates to the initial state.

vi. R : P x A ->P is a transition function.

(e.g. from a state *q* with an action *a* whose condition is true we can transit to the state *p*)

Each automaton *K(i)* describes the privacy policy of a single agent (entity).
The current state of the automaton implies the policy that is currently active for that certain agent. If the automaton *K(i)* is at a state q_i with a policy p_i ($L_i(q_i) \rightarrow p_i$) and an action α occurs, if $R_i(q_i,\alpha) \rightarrow q_i' \in R_i$ and the condition of α is *true* then the automaton will transit to state q_i' with the policy p_i' ($L_i(q_i) \rightarrow p_i$) being active from that time onwards until a new transit.

4. V(E(i),K(i)) ∀E(i) ∈ S, V is a function which connects an entity E(i) with a certain automaton K(i).

4. An automaton F which is the synthesis of ((K(1)||K(2))||...||K(n)) where operator || is defined below.

Definition: Given automata $K(1)= (Q_1,A_1,C_1,P_1,q_0^1,p_0^1,L_1,R_1)$ and $K(2)= (Q_2,A_2,C_2,P_2,q_0^2,p_0^2,L_2,R_2)$ we- define K1||K2 as $F= (Q^1 \times Q^2, A^1 \cup A^2, C^1 \cup C^2, P^1 \cup P^2, (q_0^1,q_0^2), (p_0^1,p_0^2), L^1 \cup L^2, R)$ where $q_1 \in Q_1$ and $q_2 \in Q_2$ and $(q_1',q_2') = R((q_1,q_2),\alpha)$.

a. if $\alpha \in A_1 \cap A_2$ then $q_1'=R_1(q_1,\alpha)$ and $q_2'=R_2(q_2,\alpha)$
b. if $\alpha \in A_1-A_2$ then $q_1'=R_1(q_1,\alpha)$ and $q_2'=q_2$
c. if $\alpha \in A_2-A_1$then $q_1'=q_1$ and $q_2'= R_2(q_2,\alpha)$

Note that || is a symmetric and transitive relation between automata.

As we can see above, when an automaton K(1) and an automaton K(2) have an action α ($(q_i,\alpha) \rightarrow q_i' \in R_i \wedge (q_j,\alpha) \rightarrow q_j' \in R_j$) which can transit them from state q_i,q_j to a state q_i',q_j' respectively then in the synthesized automaton F, an action α ($((q_i,q_j),\alpha) \rightarrow (q_i',q_j') \in R_f$) will move the automaton from a state (q_i,q_j) to a new state (q_i',q_j').In case of absence

of the action α inside the set of actions of an automaton we assume the transition $(q_i,\alpha)\rightarrow q_i \in R_i$ where q_i is the current state of the automaton before α can occur.

3.2 Examples

We will illustrate the language via an example. The following example represents a Notification System (NS) of a bank.
The participants of the following example are the client, the ATM machine and the system of the bank. The NS will only collect client's data, thus the client entity will be the owner of the data and the rest of the entities will participate as non-owner entities. The private data being communicated between these entities will be client's id, credit card number, telephone number and transaction amount and should be communicated with the following specifications.

1. In order to proceed with any functionality, the client needs to first provide their consent.

If the client gives their consent the following should hold:

2. The client can view their id, transactions' amount, credit card number and telephone number. They can also change their telephone number at any time. If the client restricts their consent, no functionality is provided from the NS.

3. Whenever a transaction occurs the ATM can view client's id, credit card number and update the transaction amount.

4. When an update on the transaction amount is received from the system, there are two possible cases:
    a. if the transaction amount is less than a threshold, then no further action is taken by the NS and the transaction ends.
    b. otherwise the NS will notify the associated client: Clients are notified about the transaction through their phone numbers. So, the system should send a message at the telephone number of the client, which will provide information about the credit card number and the transaction amount of the client and then the transaction ends.

To implement the above system behavior as a policy, we will first define the roles of the policy and match them with the set of entities of the language. As an owner entity we will set the Client who use the ATM and his/her personal data restrictions will be described in the policy. Next, we should set the non-owners' entities, with the remaining roles that interact in the example, such as ATM and System entities. Afterwards we will set the actions that can occur, that change our policy and the data that we will use in our language.

From the client's perspective we have the actions Consent, Restrict and Delete which represent the consent of the client and the restriction and deletion of the client from the system ,respectively. From the Atm's perspective we have the Start Transaction and End Transaction which represent the start of a transaction between the client and the Atm machine and the ending of it. Also, for the Atm we have the Update Transaction action which represent the update of the transaction amount in the database and the Update Finished action that denotes the ending of the update. Finally, for the system we have the Try To Send action which will occur only if the transaction amount is bigger than a threshold and denotes the check and the Send Transaction action which represent the sending of the message to the client.

Now that we have the entities of the language, we will proceed with showing the set of policies that each entity can implement and the set of actions of the whole policy.

Set of actions that can occur with their condition:

```
Actions{
        Consent:[true];
        Restrict:[true];
        Delete:[true];
        StartTransaction:[true];
        UpdateTransaction:[true];
        UpdateFinished:[true];
        EndTransaction:[true];
        TryToSend:[ TransactionAmount == aboveThreshhold];
        SendTransaction:[true];
         }
```

Client Entity's Policies

Client's Policy before the consent action.

Client_1	>>	{}	

Client's Policy after the consent.

Client_2	>>	Id	{read}
		Credit card	{read}
		Telephone	{read, update}
		TransactionAmount	{read}

Non-owners' Policies

System's Policy before the consent action.

System_1	>>	{}	

System's Policy after the consent of the client and before the start transaction of the Atm.

System_2	>>	{}	

System's Policy after update Transaction action from the Atm.

System_3	>>	TransactionAmount	{read, usage(threshold)}

System's Policy if the try to send action of the system occurs.

System_4	>>	Id	{read}
		Credit card	{read, disseminate(Client)}
		Telephone	{read}
		TransactionAmount	{read, disseminate(Client)}

System's Policy after the start transaction of the Atm and before the Update transaction of the Atm.

System_5	>>	{}	

Atm_1	>>	{}	
Atm_2	>>	{}	
Atm_3	>>	Id	{read}
		Credit card	{read}
		TransactionAmount	{read, update}

In order to fully complete the automaton for each agent/entity we will now declare the transition function *R* of each role which will connect two policies with an action. The following transition function will be in the form of (q.) which represents the initial policy /state or (q:α:q') where *q*' is a policy/state that the automaton can transit to when it is on state *q* and an action *α* occurs . Note that if an action is used in more than one automata, then we want the automata to synchronize at that action, and no other prohibitions exist. Though, we advise to not use the same action from a state to transit to multiple states,

in order to keep the policy unambiguous.

Client(1).
Client(1): Consent :Client(2).
Client(2): Delete :Client(1).
Client(2): Restrict :Client(1).

Atm(1).
Atm(1): Consent :Atm (2).
Atm (2): Delete :Atm (1).
Atm (2): Restrict :ATM(1).
Atm (2): StartTransaction :Atm (3).
Atm (3): EndTransaction :Atm (2).

System(1).
System(1): Consent :System(2).

System(2): StartTransaction :System(5).

System(5): EndTransaction :System(2).

System(2): Delete :System(1).

System(2): Restrict :System(1).

System(5): UpdateTransaction :System(3).

System(3): TryToSend :System(4).

System(4): SendTransaction :System(3).

System(3): UpdateFinished :System(5).

At this point the declaration of the policy is completed and we can visualize the automata of each entity and the synthesized automaton that will represent the entire policy of the example.

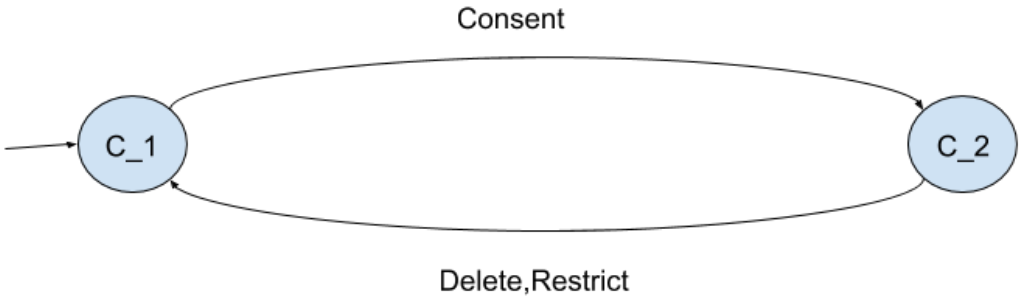

Figure 3.1 :Client automaton

We represent the states of the automaton with circles, while the actions are represented with the arrows connecting the circles. The arrow who abuts only with one circle denotes the initial state of the automaton.

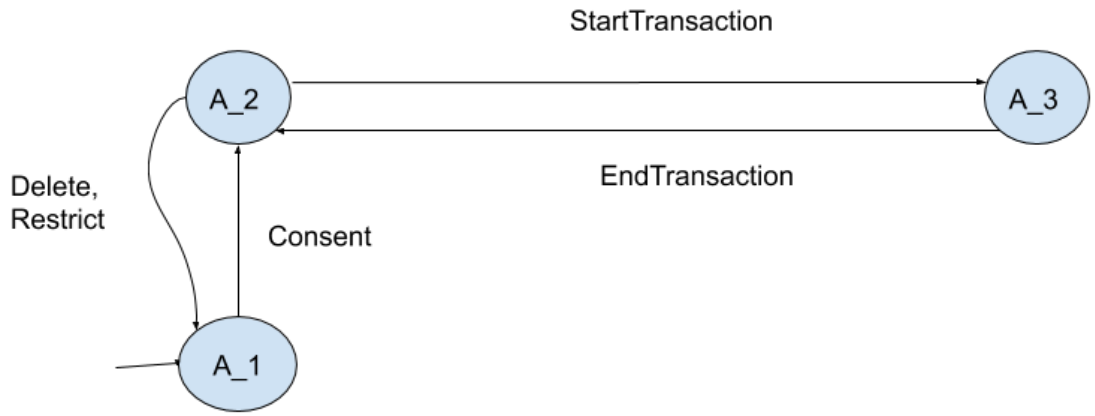

Figure 3.2 :Atm automaton

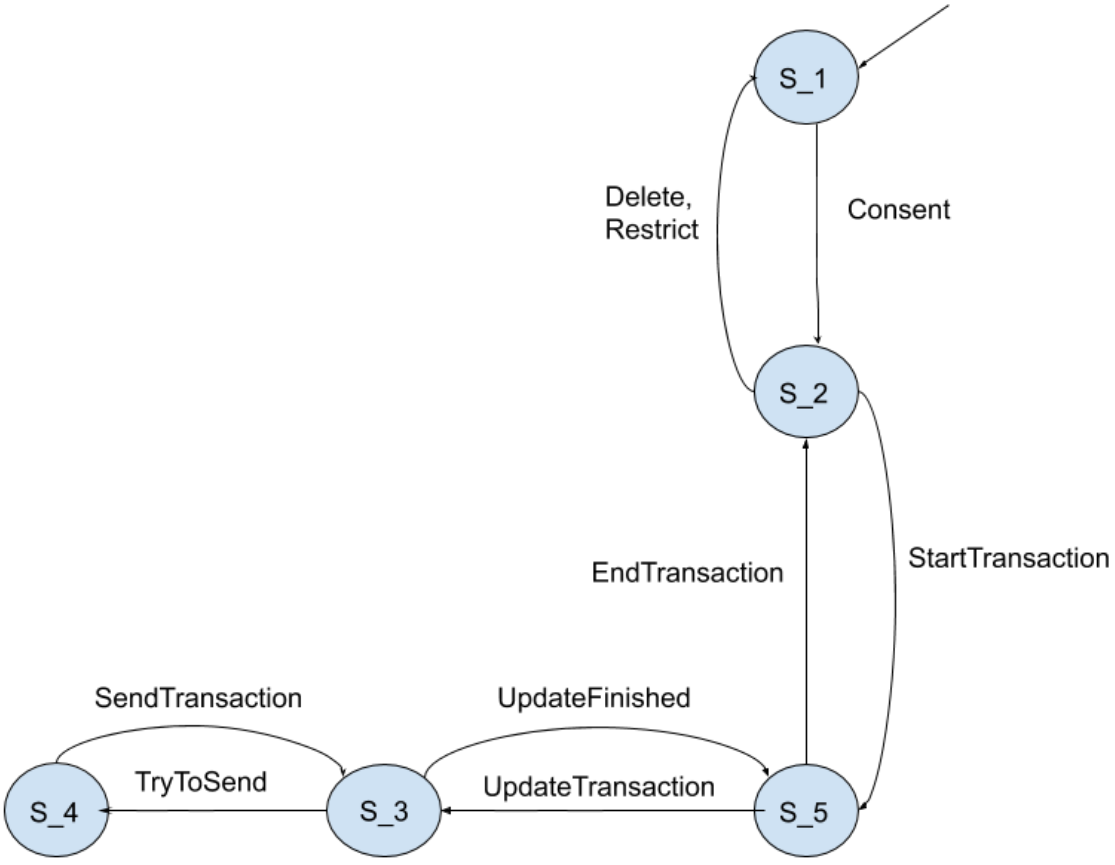

Figure 3.3 :System automaton

Now the synthesized automaton can be built using the rules described in the declaration of the language based on parallel composition of time automata [4].

Synthesized automaton – Global Policy :

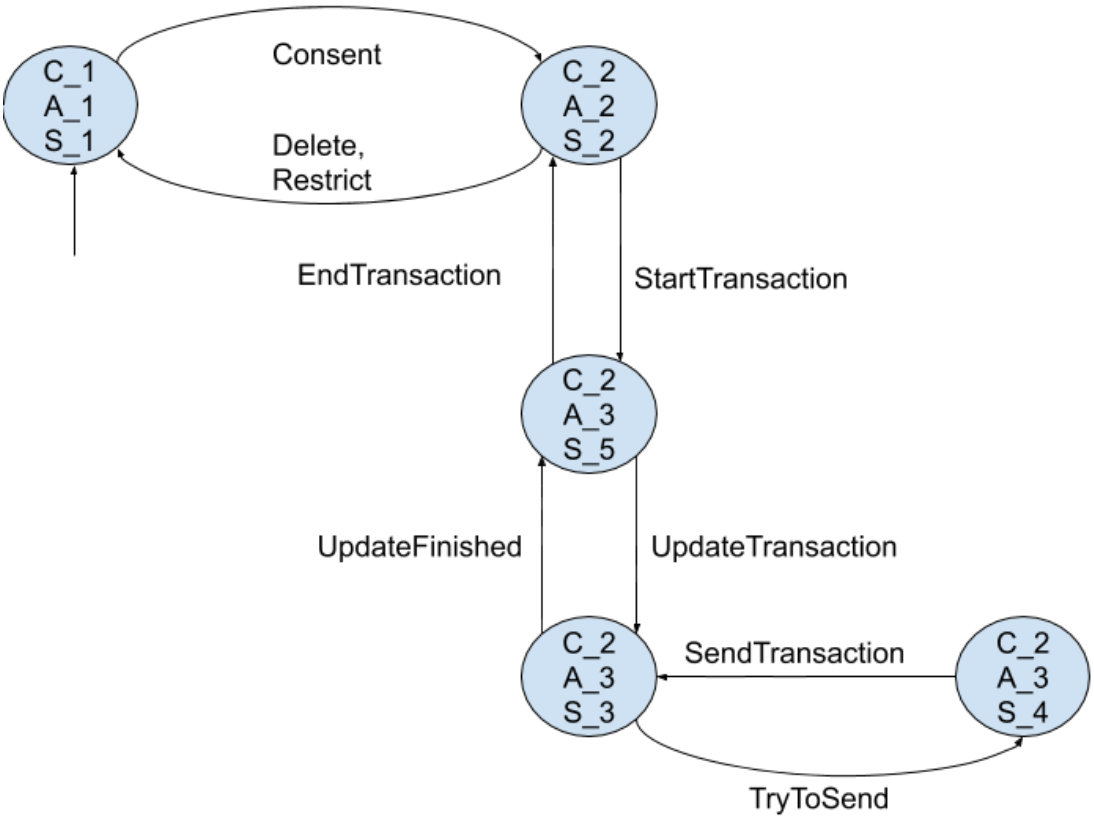

Figure 3.4 :Global automaton

Chapter 4

PPA: The part for designing Privacy Policies

As we have explained before, in this thesis we will introduce a new tool called PPA, which will be an assistant for designing privacy policies and transforming them in a format easily understandable for software developers. At this section we will describe the part of the tool that will help the designer of the policy to describe the policy for a system. We keep in mind that the people creating the policy for a system may not have any previous experience and knowledge with Computer Science and their terms, so we will try to keep the language and the notions of the tool as simple as possible.

4.1 Motivation

After the introduction of our formal privacy policy language, the problem of describing the language without a previous knowledge on the notions of formal languages, automata and their synthesis arose. Our idea to solve that problem was to remove the text-writing part of the language from the users and give them a graphical and more comprehensible way to describe the policy as a graph or a state machine directly. So, we needed a tool that a user could use to design an automaton or a graph in order to produce directly each entity's policy and transitions. After an extensive literature review on tools we found out that none of the existing tools can help us describe exactly what we want basically because most of the tools extract the graph or state machine as a picture. Some tools that were researched more methodically were open-source code on GitHub repository.

After some more research we concluded to the point that we can create our own graphical modeling workbench. With that we can link the notions of the language with graphical shapes, something much better than taking a tool that designs and describes something different and try to enforce our language on it. After the research we have concluded to two open source designing tools for this purpose. Those were Papyrus-RT (Papyrus real time) [12] and Obeo Designer [13].

The two tools are similar with slight differences. In our tool we have used Obeo Designer for two basic reasons.

1. Obeo Designer is written mainly in the Java programming language, using Eclipse IDE as the environment for designing and creating the models, which is a benefit for us due to previous experience with Eclipse IDE and Java.
2. Research personnel of the university used Obeo Designer before, giving it a good review, especially for the needs of this thesis.

4.2 Design and implementation

As a first step, in Obeo Designer we need to design a UML diagram representing the metamodel of our design, which will denote how the different parts of the policy will link later on the design.

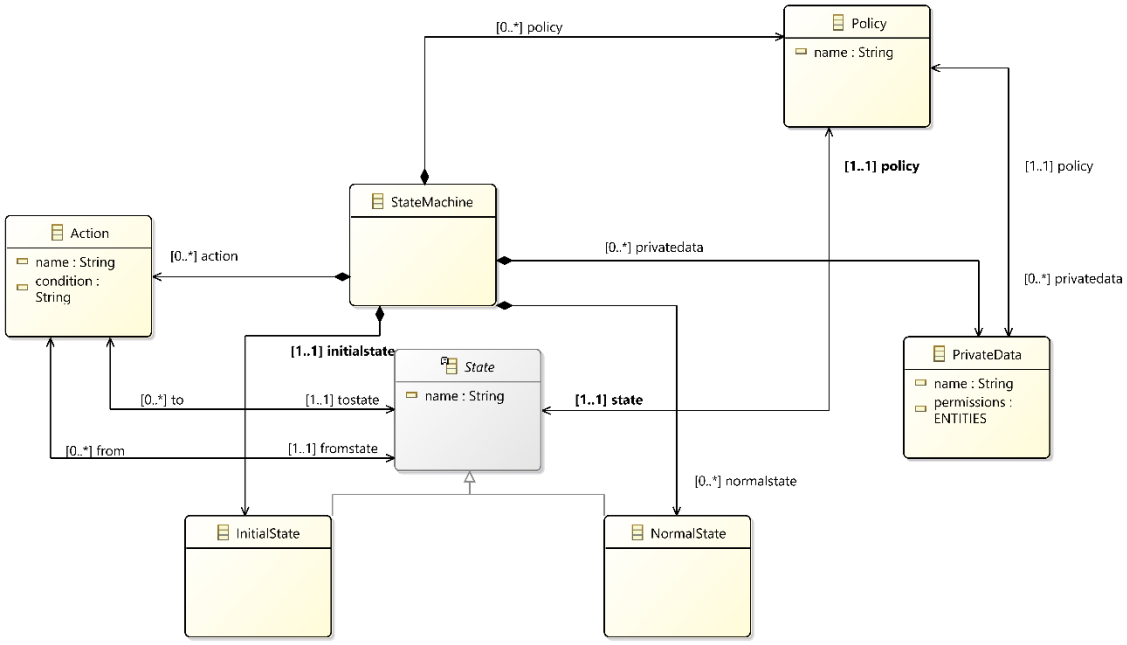

Figure 4.1: UML diagram of the state machine

As we can see in Figure 4.1, our state machine class is composed by numerous policies, actions, private data, states and an initial state. We describe these below.

Normal State/Initial State

Normal State and Initial State both inherit from the abstract class State and they are the states of our state machine.

Attributes: Name

Edges:

1. State→Policy Each state is directly linked with only one policy.
2. State→Action Each state can be linked with numerous from and to actions. By that we mean that in our state machine a state may have zero to infinite transitions coming to or going from it.

Link with formal language: State = Q

Action

The action class references to the transitions of our state machine.

Attributes: Name,Condition

Edges:

1. Action→State Each action begins only from one ‘from’ state and abuts to only one ‘to’ state.

Link with formal language: Action = A

Policy

Policy class represents the policy linked with every state of our state machine.

Attributes: Name

Edges:

1. Policy→State Each policy is directly linked with only one state.
2. Policy→PrivateData Each policy can be linked with numerous private data.

Link with formal language: Policy = P

PrivateData

The PrivateData class references to a certain private data contained in a single policy of our state machine.

Attributes: Name,Permissions

Edges:

1. PrivateData→Policy Each private data can belong to only one policy.

Link with formal language: PrivateData = a private data of the policy P

Keeping these rules in mind the graphical workbench was created giving to the user a palette with the above options to choose from and create a state machine.

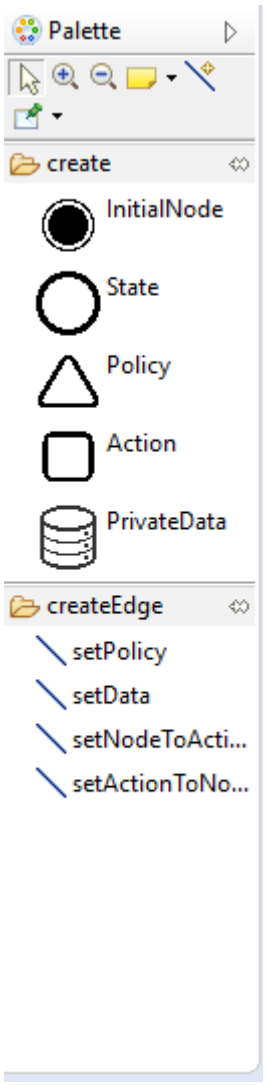

Figure 4.2 The palette of the workbench

Create section helps the user to create the main objects of the state machine as we will see further on.

Initial Node/State

The user can select the icon named Initial Node in (Figure 4.2) from the palette to create the initial state of the state machine on the workbench. Only one initial state can exist in each entity's state machine. The user can change the name of the state once the object is selected.

Normal State

The user can select the icon named State in (Figure 4.2) in order to create the states of the state machine except the initial. There is not a limit in the amount of normal states that

can simultaneously exist in an entity's state machine. The user can change the name of the state once the object is selected.

Action

The user can select the icon named Action in (Figure 4.2) to add a new action on the state machine. Each action must relate to two states. The user can change the name and the condition of the action once the object is selected.

Policy

The user can select the icon named Policy in (Figure 4.2) to add a new policy to the state machine workbench. Each policy must be related with one state of the state machine.

Private Data

The user can select the icon name Private Data in (Figure 4.2) to add a new set of private data and permissions on it to the state machine. All private data must relate to a policy. The user can change the name of the private data and also add the permissions once the object is selected under semantics tab.

After declaring the objects of the state machine, the user should declare the relation between those objects for the state machine to be complete. To help user relate objects the palette contains another sector called "create edge". In this section the user has four options.

1. SetPolicy

   This edge starts from either a Normal State, or an Initial State of the state machine and ends up on a Policy object. It describes the 1-1 relationship between a state and a policy.

2. SetData

   This edge starts from a Policy object and ends up on PrivateData object. It describes the relationship between a policy and its own data.

3. SetNodeToAction

This edge starts from either a Normal State, or an Initial State and end up on an Action object. It describes the relation that an action has only one starting state.

4. SetActionToNode
   This edge starts from an Action object and ends up on either an Initial State, or a Normal State. It describes the relation that an action ends up on only one state.

Now that we have declared the state machine as a design pattern, we can move on to provide a simple manual and some examples on how to start, create, design and save policies via PPA.

4.3 Simple Manual

At first the user needs to launch the Obeo Designer software by its launcher that will be provided in the package of PPA. Once the Obeo Designer Software is open the user should click on the play button (Figure 4.3), that can be found inside the toolbar on the top of the application window and a new application window will pop-up.

Figure 4.3 The play button

At this point the user should start creating the entities of the system that they want to design. To launch the design workbench of an entity the user should first create a new Modelling Project under the Model Explorer window, preferable with a name referencing to the entity name that will be designed. The next step is to click on the new folder that was created and click New/ Other (Figure 4.4)/ StateMachineModeller Model (Figure 4.5).

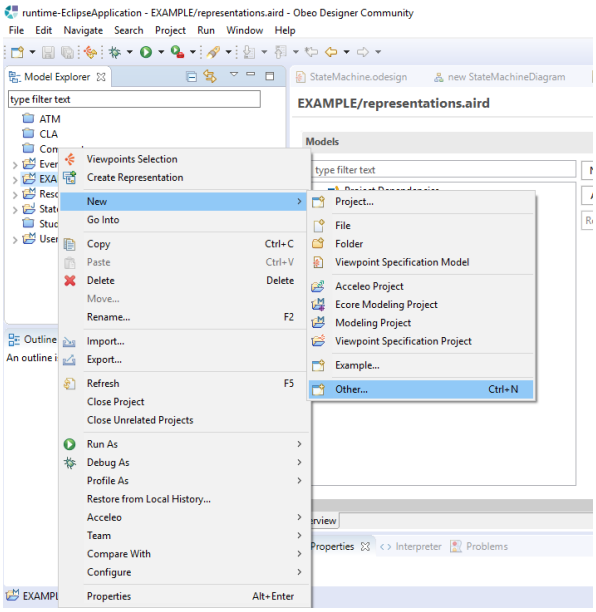

Figure 4.4 Choosing non-existing models

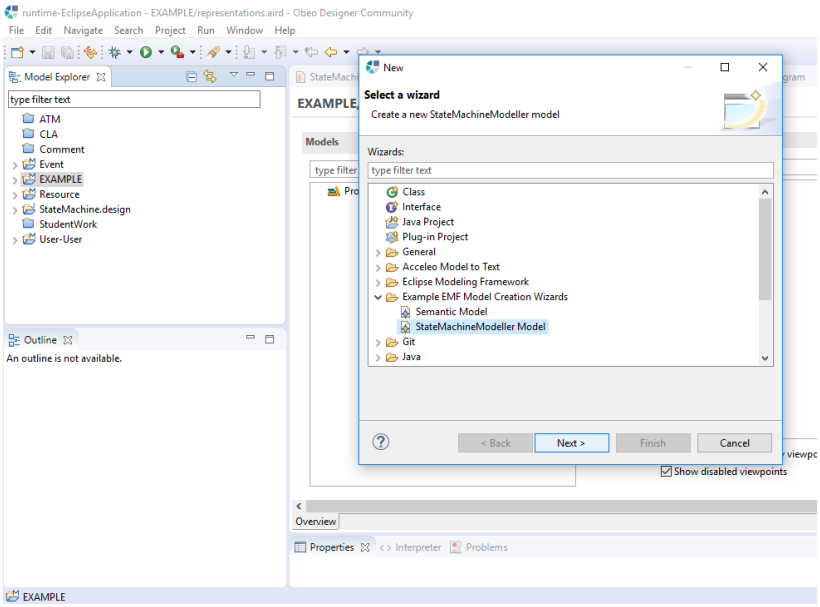

Figure 4.5 Choosing State Machine Model

At this point the user needs to give the name of the entity that will be designed and if the entity is the owner of the data the prefix “User-“ must be included in the name. At the next pop-up screen the user should enter State Machine in the Model Object dropdown menu and click Finish (Figure 4.6).

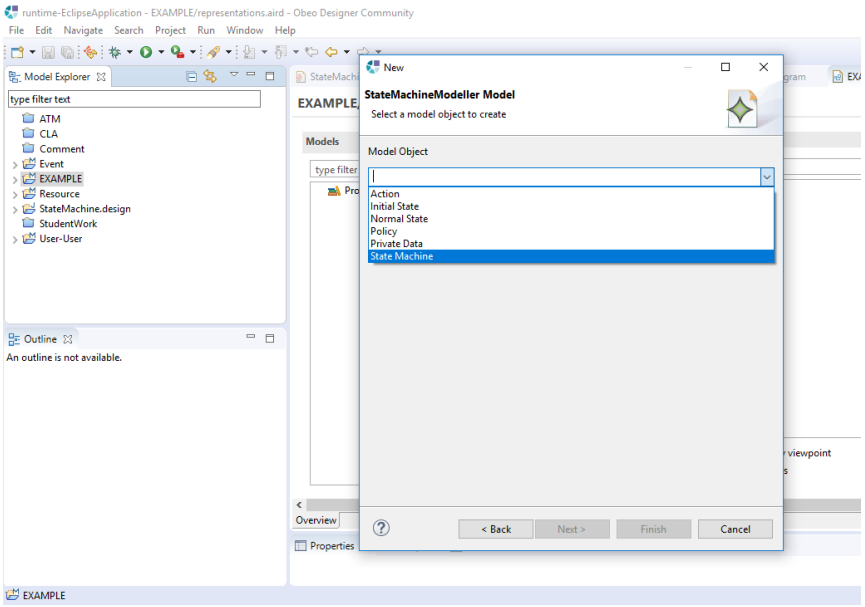

Figure 4.6 Choosing the object we want to model

Now the model of the entity is created and the design workbench for that entity can be launched. In order to do so, the user should right click on the Modelling Project created before and choose Create Representation (Figure 4.7).

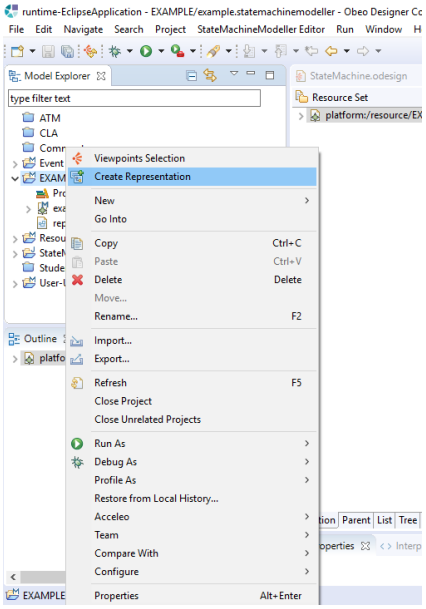

Figure 4.7 Creating the representation of the model

The state machine options should be chose and then click next and finish (Figure 4.8).

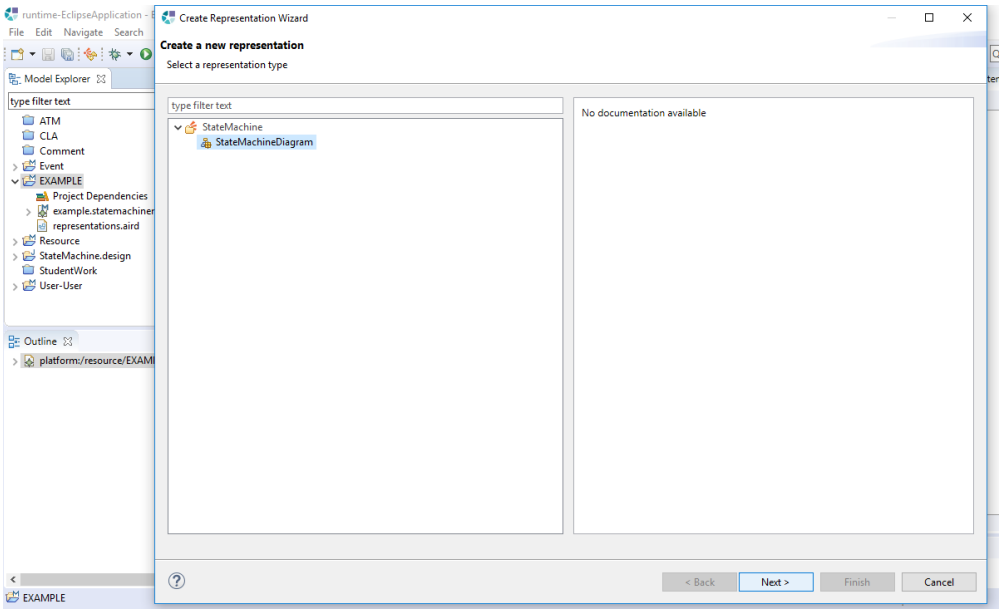

Figure 4.8 Choosing State Machine as a diagram for the model

Subsequently, an empty design workbench appears on the screen with the palette described before on the right side (Figure 4.9). Now the user can design the policy of the entity. It is clear, that by creating or deleting Modelling Projects on the current window, the user can create multiple entities thus a global system policy could be designed.

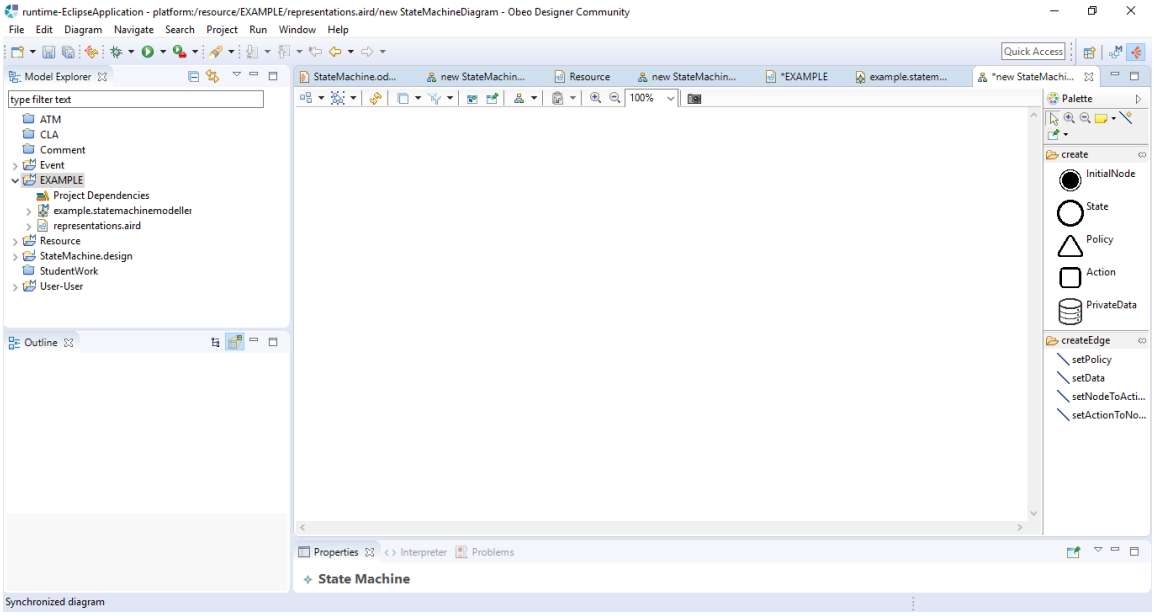

Figure 4.9 Empty workbench with the palette on the right side

4.3.1 Examples

For consistency purposes the examples provided describe the policy that was declared in Chapter 3, this time declared on our policy designer.

1. Client Entity

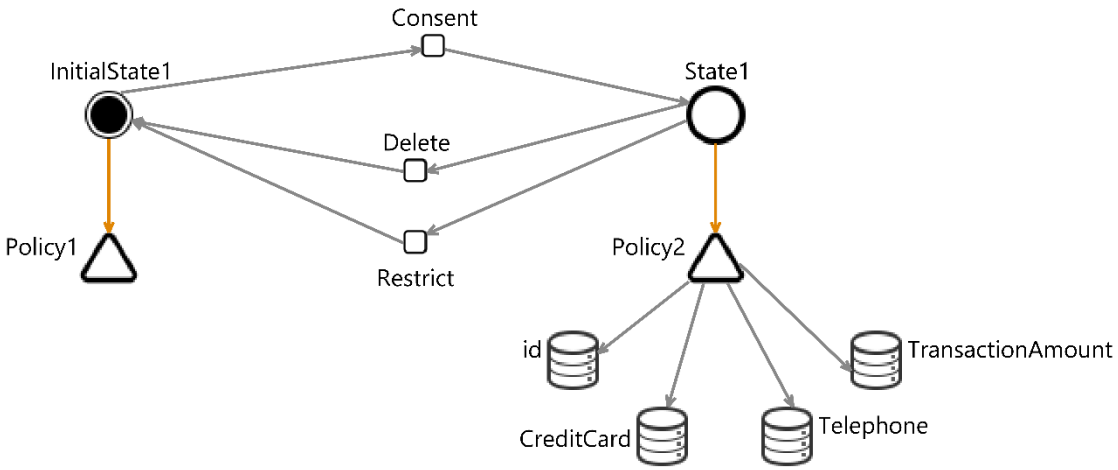

Figure 4.10 Client Policy

2. Atm Entity

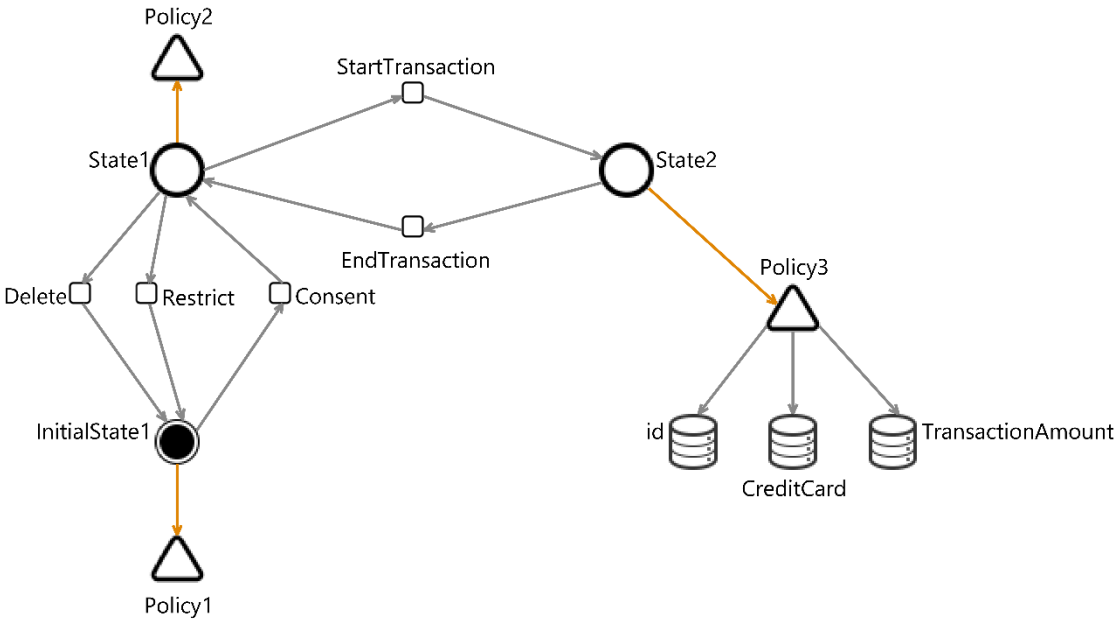

Figure 4.11 Atm Policy

3. System entity

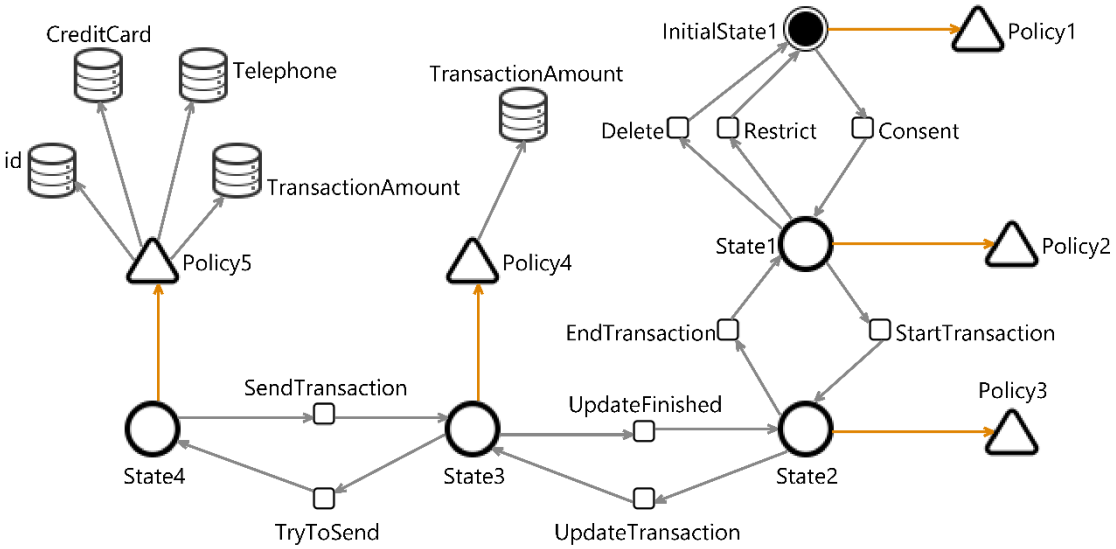

Figure 4.12 System Policy

As we can see our tool keeps the clarity of the formal text-based language as all the information is enclosed into the design, but also gives a more visual comprehension of the policy.

Chapter 5

PPA: Parsing policies and synthesizing global Policy Graphs

In this chapter we will introduce another part of the PPA framework. We will introduce the parser of the designed policies created in the previous step and the transformation of the policy to text or graph for future work.

5.1 Motivation

A visual policy as we have mentioned in the previous section helps people comprehend more easily the way an entity in the system changes its permissions when a series of actions occurs in the system and keeps the work of the policy "author" simpler in terms of scalability, as they don't have to design the entire policy again if a change in an entity's policy occurs later. On the other hand, having the policy of a system broken into parts hardens the ability to monitor the behavior of the global system. Except from the fact that an image of a visual policy would not so easily get accepted by clients of the systems or data protection enforcements across the globe. In order to solve these problems, we will introduce a parser of the visual policies that will transform the policy to two different forms, each one serving its own purpose.

5.2 Parsing and creating the graph

Before we introduce the parser and the other methods, we need to understand the way our workbench represents the visual policies we designed before. The Obeo Designer represents the objects and edges we created in XML form. For example, the Client entity

policy from Chapter 3, is represented in visual and XML form as showing in Figures 5.1 and 5.2.

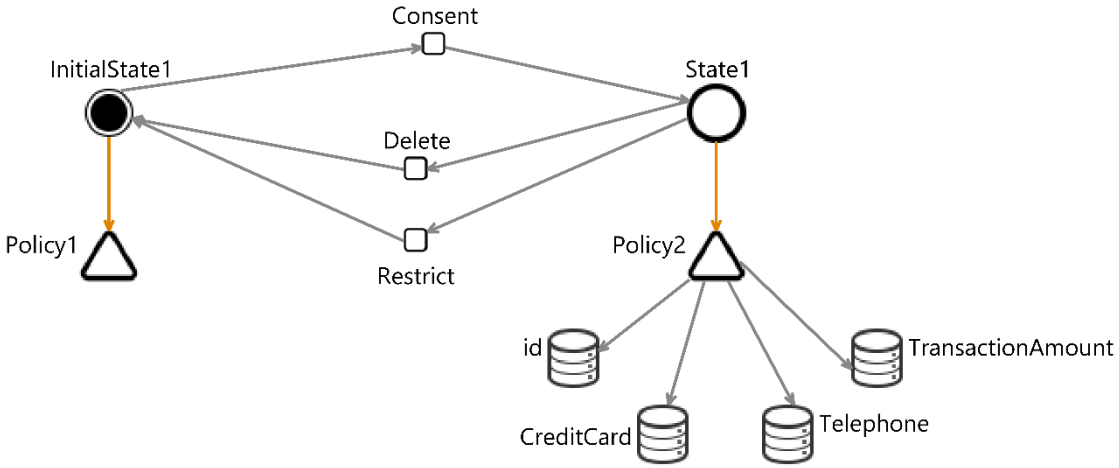

Figure 5.1 Client Visual Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<stateMachineModeller:StateMachine xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:stateMachineModeller="http://www.example.org/stateMachineModeller">
<action name="Consent" condition="true" fromstate="//@initialstate" tostate="//@normalstate.0"/>
<action name="Delete" condition="true" fromstate="//@normalstate.0" tostate="//@initialstate"/>
<action name="Restrict" condition="true" fromstate="//@normalstate.0" tostate="//@initialstate"/>
<initialstate name="InitialState1" policy="//@policy.0" from="//@action.0" to="//@action.1
//@action.2"/>
<normalstate name="State1" policy="//@policy.1" from="//@action.1 //@action.2" to="//@action.0"/>
<policy name="Policy1" state="//@initialstate"/>
<policy name="Policy2" privatedata="//@privatedata.2 //@privatedata.3 //@privatedata.1
//@privatedata.0" state="//@normalstate.0"/>
<privatedata name="CreditCard" permissions="read" policy="//@policy.1"/>
<privatedata name="Telephone" permissions="read update" policy="//@policy.1"/>
<privatedata name="id" permissions="read" policy="//@policy.1"/>
<privatedata name="TransactionAmount" permissions="read" policy="//@policy.1"/>
</stateMachineModeller:StateMachine>
```

Figure 5.2 Client XML Policy

We took advantage of the XML-form that Obeo uses to represent visual policy in order to parse it. To do so we used ANTLR (ANother Tool for Language Recognition) [11] as a parser and lexer and Java. In the Appendix A you can find the grammar used from

ANTLR to parse the XML. After the parsing of the XML we used Java code to produce again the policy in our text-based formal language that we introduce in Chapter 3.You can also find that in the Appendix A. Having now the formal language we created another parser with the help of ANTLR and Java again and represented the policies as graphs for each entity. You can find these in Appendix B. Keeping in mind the parallel composition of the entities automata that we introduce in our language definition we tried this time to use that algorithm in order to produce one global graph, which is the synthesis of all entities' graphs. The philosophy of the algorithm remains the same, with minor changes to make it more suitable for graphs.

Pseudocode:

```
1.  for G ∈ E
2.     IGG ← initState(G)      /*Add initial state to global graph initial state/*
3.  enqueue(P,IGG)                              /* enqueue initial state to queue/*
4.  while P ≠ ∅
5.      h ← dequeue(P)                                    /*dequeue a global state/*
6.      L ← h                                                   /* add h to list L/*
7.      for s ∈ h
8.          for  a ∈ start(s,a)
9.              if a ∈ s ∧  (a ∉ allGraphs(GxE)- graph(s))
10.                 GS ← to(a,s) U (h-s)
11. /* create state with a transit state from s and the rest of states in h/*
12.             else
13.                 synchronize ← true
14.                 for G with a ∈ G
15.                     h ← h-q                          /* remove state q from h/*
16.                     N ← q                                   /* add q to list N/*
17.                     if a ∉ q
18.                         synchronize ← false
19.             if synchronize = true
20.                     GS ← to(a) U h
21. /* create state with all the transit state that a jumps to and states in h/*
22.                 else
23.                     continue;
24.             if t ∈ L
25.                t ← transit(a,s)
26.             else
27.                t ← transit(a,s)
28.                enqueue(P,t)
```

As we can see in the pseudocode the global graph will synchronize all the partial entities' graphs to transit simultaneously to the next state if they are to execute the same action without interrupting states that don't synchronize with them.

After the parsing of the language and the synthesis of the graphs to the global graph we have encountered our last problem, which was the form that would be more suitable to represent the policy. We understood that we had to form the policy in more than one way, in order to keep it easy to read for users and developers and flexible enough for us and other researchers to model check or monitor in the future. We came up to two forms of representation except from the visually represented partial policies of Obeo Designer and our formal text-based transformation of it.

Firstly, we introduce a new user-friendly text representation of the global policy for users that describes the permissions every entity has at any point inside the system. Secondly with the help of another tool called Graphviz [7] we produced a pdf visual representation of the graph for developers. Finally, with the help of Graphviz and the input file that we provide to it to draw the graph, researchers can easily run graph algorithms, most of them already implemented in Graphviz to check for properties validity.

5.3 Examples

We will now provide the formal policy that was produced by the transformation of the visual policies, the pdf outcome of the Graphviz and the user-friendly text policy after the synthesis of the entities' policies for the example that was provided before.

The formal policy produced by the parser, with input the examples of Chapter 4 is shown below.

```
Actions{
TryToSend:[TransactionAmount==AboveThreshold];
SendTransaction:[true];
StartTransaction:[true];
Delete:[true];
EndTransaction:[true];
UpdateFinished:[true];
Restrict:[true];
Consent:[true];
```

```
UpdateTransaction:[true];
}
User{
client{
client_InitialState1          >>                    [].

client_State1          >>          CreditCard        [read]
                                   Telephone         [read,update]
                                   id                [read]
                                   TransactionAmount [read].
}
}

Entities{
atm{
atm_InitialState1      >>                    [].

atm_State1             >>                    [].

atm_State2             >>          id                [read]
                                   CreditCard        [read]
                                   TransactionAmount [read,update].

}
system{
system_InitialState1          >>                    [].

system_State1          >>                    [].

system_State2          >>                    [].

system_State3          >>          TransactionAmount [read,usage(threshold)].

system_State4          >>          id                [read]
                                   CreditCard        [read,disseminate(client)]
                                   Telephone         [read]
                                   TransactionAmount [read,disseminate(client)].

}
}
Transitions{
atm{
atm_InitialState1.
atm_InitialState1:Consent:atm_State1.
atm_State1:Restrict:atm_InitialState1.
atm_State1:Delete:atm_InitialState1.
atm_State1:StartTransaction:atm_State2.
atm_State2:EndTransaction:atm_State1.
}
system{
system_InitialState1.
system_State1:Restrict:system_InitialState1.
system_InitialState1:Consent:system_State1.
system_State1:Delete:system_InitialState1.
system_State1:StartTransaction:system_State2.
system_State2:EndTransaction:system_State1.
system_State2:UpdateTransaction:system_State3.
system_State3:UpdateFinished:system_State2.
system_State3:TryToSend:system_State4.
```

system_State4:SendTransaction:system_State3.
}
client{
client_InitialState1.
client_InitialState1:Consent:client_State1.
client_State1:Delete:client_InitialState1.
client_State1:Restrict:client_InitialState1.
}
}

Below we can see the pdf for the same example as before produced by Graphviz:

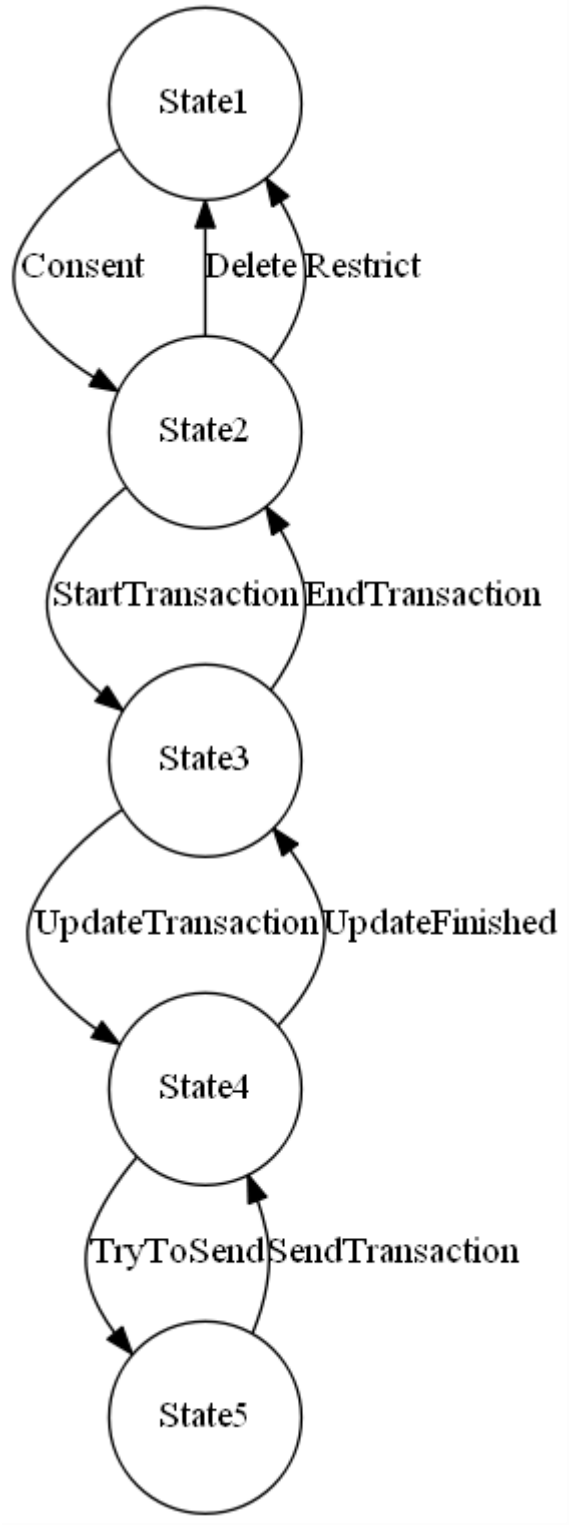

Figure 5.2 Graphviz pdf production

As we have mentioned before, our tool can represent the policy in a more user-friendly way than the formal policy. Below you can find the same example as before but from a different perspective, a bit closer to the natural language. The policy is explained state by state, with the permissions that apply in every state.

The policy with id 1 has the following permissions on data:
The entity atm can :
1. Do nothing
The entity system can :
1. Do nothing
The entity client can :
1. Do nothing

After the Consent action the policy with id 2 will take the place of the current policy

The policy with id 2 has the following permissions on data:
The entity client can :
1. (read) on private data CreditCard
2. (read,update) on private data Telephone
3. (read) on private data id
4. (read) on private data TransactionAmount
The entity system can :
1. Do nothing
The entity atm can :
1. Do nothing

After the Delete action the policy with id 1 will take the place of the current policy
After the Restrict action the policy with id 1 will take the place of the current policy
After the StartTransaction action the policy with id 3 will take the place of the current policy

The policy with id 3 has the following permissions on data:
The entity client can :
1. (read) on private data CreditCard
2. (read,update) on private data Telephone
3. (read) on private data id
4. (read) on private data TransactionAmount
The entity system can :
1. Do nothing
The entity atm can :
1. (read) on private data id
2. (read) on private data CreditCard
3. (read,update) on private data TransactionAmount

After the EndTransaction action the policy with id 2 will take the place of the current policy
After the UpdateTransaction action the policy with id 4 will take the place of the current policy

The policy with id 4 has the following permissions on data:
The entity client can :
1. (read) on private data CreditCard
2. (read,update) on private data Telephone
3. (read) on private data id
4. (read) on private data TransactionAmount
The entity system can :
1. (read,usage(threshold)) on private data TransactionAmount
The entity atm can :
1. (read) on private data id
2. (read) on private data CreditCard
3. (read,update) on private data TransactionAmount

After the UpdateFinished action the policy with id 3 will take the place of the current policy
After the TryToSend action the policy with id 5 will take the place of the current policy

The policy with id 5 has the following permissions on data:
The entity client can :
1. (read) on private data CreditCard
2. (read,update) on private data Telephone
3. (read) on private data id
4. (read) on private data TransactionAmount
The entity system can :
1. (read) on private data id
2. (read,disseminate(client)) on private data CreditCard
3. (read) on private data Telephone
4. (read,disseminate(client)) on private data TransactionAmount
The entity atm can :
1. (read) on private data id
2. (read) on private data CreditCard
3. (read,update) on private data TransactionAmount

After the SendTransaction action the policy with id 4 will take the place of the current policy

5.4 Evaluation

After the designing and the implementation of the language and tools, our framework was tested with a full working system provided to us by our university. The system required six entities including the owner entity and four states were designed for each entity. So, the user of our tool designed twenty-four states in our tool. After the parsing of the visual declaration of the policies and the synchronization of the entities, the global graph of the system was produced, containing eighty-one states from the available four thousand and

nighty-six states that were able to occur. The graph of the system is provided in the Appendix C.

After the collection of this crucial information, we can clearly state that declaring and designing a policy partially, reduces the time and work a policy designer needs to declare the policy of a system. Apart from that, with the partial declaration of the policy as an entity's policy at a time, a small change on an entity's policy that may occur in the lifetime of our system will be dealt more easily.

Chapter 6

Conclusion and Future work

6.1 Conclusion

In this thesis we have seen that most of the existing privacy languages, need to change in order to follow up the extremely demanding world of 2020. Our framework, consisting of a formal privacy policy language and a tool, respects that and implements some features in order to help the experts of today to design and create policies for their systems. As we have seen our framework helps the designing team of a system to declare an easier, visual privacy policy and then transform it to different forms to be easily comprehended by other groups of people.

The most respectful part of our framework is the novelty that associates the policy of a system with its behavior. Until now policies were only describing the permissions of a system without the ability to be model checked and a variety of questions about compatibility of policies were raised. As we will see in the next section, a new generation of policies may come to life, involving both the designers of a system and the legal teams declaring the policies into one, creating policies that are not answering only the question of what private information a system uses but also how and when, with formal rules enforcing that.

6.2 Future Work

6.2.1 Model checking/compliance

As we discussed earlier a crucial feature of new privacy policy languages will be the model checking of policies for specific properties and the check against regulations for compliance.

Firstly, model checking or property checking is, for a given model of a system, to exhaustively and automatically checking whether this model meets a given specification. Typically, one has hardware or software systems in mind, whereas the specification contains safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. Model checking is a technique for automatically verifying correctness properties of finite-state systems. In terms of that our language can be translated (the graph) to a finite-state system in order to be model checked. To this end, the properties or specifications are formulated in some precise mathematical language and afterwards the structure is checked for satisfaction of the given formula. As we can see our language and the products of our tool can be subject to model check if the system and a regulation or a property we want to check, are both formulated in a logic formula. To verify that model checking can be achieved on our language as future work we will formulate a property into a linear temporal logic in order to be checked against the example we provided through this thesis.

Example

For this part we will take as example a GDPR property.

A system never has any permissions on private data before a consent from the owner's data is given.

We will use two predicates to formulate this sentence.

1. Perm(X) which represent that X is a permission
2. Consent which represent the action of consent the owner's data

LTL

$\forall x \neg perm(x)$ U Consent

It is obvious for our example that the current property is satisfied, mainly because the first state of our graph does not have any permissions and a consent action follows that allowing any permissions from that point onwards.

With the same logic as we have shown, a more complex and realistic model check can be completed, checking the policy or even the system against any property or a group of properties and as an extension of that checking if the system complies with a regulation.

6.2.2 Static check

In this thesis we gave a huge emphasis to develop a tool that will help different departments and teams at system's design phase. A future development of our framework will be to help also the development of a system, not only its design.
A way of doing that is by creating a type check system that can type check our policy against a completed program in a specific programming language and point out any breaches of the policy that was initially designed.

More specifically static type checking is a way of type-checking a program at compile-time, indicating errors or bugs that can be found at this point. In the future a type check system can be created translating our produced graph to a programming language's grammar and checking if the current program complies with the policy that was declared giving developers a safety net during coding time.

6.2.3 Monitoring

As we indicated static check can check if a program complies with a policy during compile-time resolving a great amount of errors and breaches, but several errors cannot be checked during that time. Run-time errors and breaches can only be discovered by monitoring the system while running. In order to do so our policy and more specifically the graph produced by our tool will have to be interpreted by a system into some sort of

code, and then monitor the system or the program as it runs for violations or breaches of those rules. It will be extremely helpful to an owner to be ensured that his/her system always complies with a policy and when a violation happens a safety program will be there to detect it.

If the proposals that we have suggested come to life, a new era of privacy policy frameworks will come both for companies and users. If that happens, a designer of a system will only have to worry about the declaration of the policy as the textual form of the policy, the programs of the policy, and the run-time system will follow automatically the visual policy that was declared. Except of the above, the designer will also have the ability to check the visual policy against a regulation and by extension the whole system. In order to understand better the problems and give better solutions to privacy policies "problems" we will need to address to the experts involved in the creation of policies to evaluate our work and give feedback to us and other researches to keep us on track for the future. Especially, if we want to associate this proposal with companies and give it a more practical relevance a lot of work need to follow beginning with the work that we have mention and the useful information that experts can provide.

References

[1] R. Agrawal, J. Kiernan, R. Srikant and Y. Xu, "An XPath-based Preference Language for P3P," in *In Proceedings of the Twelfth International Conference on World Wide Web*, 2003.

[2] R. Agrawal, J. Kiernan, R. Srikant and Y. Xu, "XPref: a preference language for P3P," *Computer Networks*, vol. 48, pp. 809-827, 2005.

[3] P. Ashley, S. Hada, G. Karjoth, C. Powers and M. Schunter, "Enterprise Privacy Authorization Language (EPAL 1.2)," World Wide Web Consortium (W3C), [Online]. Available: https://www.w3.org/Submission/2003/SUBM-EPAL-20031110/. [Accessed 19 May 2019].

[4] C. Baier and J.-P. Katoen, Principles of Model Checking (Representation and Mind Series), The MIT Press, 2008, pp. 38-40.

[5] A. Barth, J. C. Mitchell and J. Rosenstein, "Conflict and combination in privacy policy languages," in *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, 2004.

[6] L. Diver and B. Schafer, "Opening the black box: Petri nets and Privacy by Design," *International Review of Law, Computers & Technology*, vol. 31, pp. 68-90, 2017.

[7] J. Ellson and E. Gansner, "Graphviz - Graph Visualization Software," [Online]. Available: https://www.graphviz.org/. [Accessed 3 May 2019].

[8] M. Fernández, J. Jaimunk and B. Thuraisingham, "Graph-Based Data-Collection Policies for the Internet of Things," in *Proceedings of the 4th Annual Industrial Control System Security Workshop*, 2018.

[9] D. Kouzapas and A. Philippou, "Privacy by typing in the pi -calculus," *arXiv preprint arXiv:1710.06494*, 2017.

[10] P. Kumaraguru, L. Cranor, J. Lobo and S. Calo, "A survey of privacy policy languages," in *Workshop on Usable IT Security Management (USM 07): Proceedings of the 3rd Symposium on Usable Privacy and Security, ACM*, 2007.

[11] T. Parr, "ANTLR (ANother Tool for Language Recognition)," [Online]. Available: https://www.antlr.org/. [Accessed 10 February 2019].

[12] "Papyrus Real Time," Eclipse Foundation, [Online]. Available: https://www.eclipse.org/papyrus-rt/. [Accessed 12 February 2019].

[13] "Obeo Designer," OBEO, [Online]. Available: https://www.obeodesigner.com/en/. [Accessed 12 February 2019].

[14] OASIS, "Security Assertion Markup Language (SAML) V2.0 Technical Overview," OASIS, 25 March 2008. [Online]. Available: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html. [Accessed 20 April 2019].

[15] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0 Specification," OASIS, 22 January 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. [Accessed 20 April 2019].

[16] World Wide Web Consortium (W3C), "Platform for Privacy Preferences (P3P) Project," (W3C), [Online]. Available: https://www.w3.org/P3P/. [Accessed 19 May 2019].

[17] Zero-Knowledge Systems, "Privacy Rights Markup Language Specification," Zero-Knowledge Systems, June 2001. [Online].
Available: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.217.2902. [Accessed 20 April 2019].

Appendix A

The grammar of ANTLR and the java code for the parser that transforms the visual privacy policy to formal text policy.

ANTLR grammar

```
grammar ParseXml;


program : Encode sms;
Encode: '<?xml version="1.0" encoding="UTF-8"?>';

sms:'<stateMachineModeller:StateMachine xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:stateMachineModeller="http://www.example.org/stateMachineModeller">' sm
'</stateMachineModeller:StateMachine>';

sm :  actions initial normalStates policies privateDatas ;

actions: action actions | ;

action : '<action' name condition fromState toState '/>';

name : 'name="' words '"'|;

condition : 'condition="' cond '"'|;


cond : cinside | cinside '&&' cond;

cinside : 'true' | 'false' | Word '==' Word | Word '!=' Word |Word  '!=' Num
|Word  '==' Num|Word '<' Word | Word '>' Word |Word  '<' Num
|Word  '>' Num ;

fromState: 'fromstate="//@' Word '.' Num '"'|'fromstate="//@' Word '"' | ;

toState : 'tostate="//@' Word '.' Num '"'| 'tostate="//@' Word '"' | |;

initial : '<initialstate' name policyInside fromAction toAction '/>';

policyInside : 'policy="//@' Word '.' Num '"' | ;

fromAction: 'from="' actionsInside '"' |;

actionsInside: fromtoActions actionsInside | fromtoActions;

toAction : 'to="' actionsInside '"' |;

fromtoActions: '//@' Word '.' Num;

normalStates : normalState normalStates |;
```

```
normalState : '<normalstate' name policyInside fromAction toAction '/>';

policies :policy policies|;

policy: '<policy' name privateDataInside state '/>';

privateDataInside: 'privatedata="' datas '"'|;

datas: data datas |data ;

data : '//@' Word '.' Num;

state : 'state="//@' Word '.' Num '"'|'state="//@' Word '"'|;

privateDatas: privateData privateDatas|;

privateData : '<privatedata' name permissions policyInside '/>';

permissions: 'permissions="' accessWords '"'|;

accessWords: AccessWord accessWords|;

words: Word+;

AccessWord: 'read'|'update'|'delete'|'disseminate' '.' Word '.'|'usage' '.'
Word '.' ;

Num: ('0'..'9')+;


Word : [A-Za-z0-9]+;


WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
```

MyVisitor class that visits the tree created by ANTLR to create the policy in memory.

```
import java.util.ArrayList;

import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.RuleNode;
import org.antlr.v4.runtime.tree.TerminalNode;

public class XmlMyVisitor<T> extends ParseXmlBaseVisitor<T> {

	XmlGraph g; // the state machine that will be created
	int action = 0; // ids for actions,policies,states and private data.
	int policy = 0;
	int state = 0;
	int data = 0;
	ArrayList<String> permissions;

	public XmlMyVisitor() {
		g = new XmlGraph();
```

```
			permissions = null;
		}

		@Override
		public T visit(ParseTree tree) {
			return super.visit(tree);
		}

		@Override
		public T visitChildren(RuleNode arg0) {
			return super.visitChildren(arg0);
		}

		@Override
		public T visitErrorNode(ErrorNode node) {
			return super.visitErrorNode(node);
		}

		@Override
		public T visitTerminal(TerminalNode node) {
			return super.visitTerminal(node);
		}

		@Override
		public T visitProgram(ParseXmlParser.ProgramContext ctx) {
			return super.visitProgram(ctx);
		}

		@Override
		public T visitSms(ParseXmlParser.SmsContext ctx) {
			return super.visitSms(ctx);
		}

		@Override
		public T visitSm(ParseXmlParser.SmContext ctx) {
			return super.visitSm(ctx);
		}

		@Override
		public T visitAccessWords(ParseXmlParser.AccessWordsContext ctx) {
			if (ctx.AccessWord() == null)
				return super.visitAccessWords(ctx);
			String t = ctx.AccessWord().getText(); // take the access
permissions of a data
			permissions.add(t); // add it to the permissions arraylist
			return super.visitAccessWords(ctx);
		}

		@Override
		public T visitActions(ParseXmlParser.ActionsContext ctx) {
			if ((g.a.size() != 0) && (g.a.get(g.a.size() - 1).id == -1)) {
				XmlAction a = g.a.get(g.a.size() - 1); // if an action
was declared the program will give it the next id
				a.id = action;
				action++;

			}
```

```
		return super.visitActions(ctx);
	}

	@Override
	public T visitAction(ParseXmlParser.ActionContext ctx) {
		String name;
		try {
			name = ctx.name().words().getText(); // gets the name of
the action and if the action has no name an empty

		// string will be the name of the action
		} catch (NullPointerException e) {
			name = "";
		}
		String condition;
		try {
			condition = ctx.condition().cond().getText(); // gets the
condition of an action.
		} catch (Exception e) {
			System.out.println("Warning:Condition on action " + name
					+ " is incorrect and true condition  will
replace the current condition");
			condition = "true"; // if the action doesn't have a
condition , the true conditon will take the
                                                        // place of condition
		}
		int from;
		try {
			from = Integer.valueOf(ctx.fromState().Num().getText());
// gets the from state (identity) of an action
		} catch (NullPointerException e) {
			if
(ctx.fromState().Word().getText().equals("initialstate"))
				from = -2;
			else
				from = -1;
		}

		int to;
		try {
			to = Integer.valueOf(ctx.toState().Num().getText()); //
gets the to state (identity) of an action
		} catch (NullPointerException e) {
			if
(ctx.toState().Word().getText().equals("initialstate"))
				to = -2;
			else
				to = -1;
		}
		XmlAction action = new XmlAction(name, condition, from, to); //
creates the action and adds it in the arraylsit of the

                                // state machine
		g.a.add(action);
		return super.visitAction(ctx);
	}

	@Override
```

```
	public T visitCond(ParseXmlParser.CondContext ctx) {
		return super.visitCond(ctx);
	}

	@Override
	public T visitCondition(ParseXmlParser.ConditionContext ctx) {
		return super.visitCondition(ctx);
	}

	@Override
	public T visitData(ParseXmlParser.DataContext ctx) {
		return super.visitData(ctx);
	}

	@Override
	public T visitDatas(ParseXmlParser.DatasContext ctx) {
		return super.visitDatas(ctx);
	}

	@Override
	public T visitFromAction(ParseXmlParser.FromActionContext ctx) {
		return super.visitFromAction(ctx);
	}

	@Override
	public T visitFromState(ParseXmlParser.FromStateContext ctx) {
		return super.visitFromState(ctx);
	}

	@Override
	public T visitInitial(ParseXmlParser.InitialContext ctx) {
		String name;
		try {
			name = ctx.name().words().getText(); //gets the name of
the initial state
		} catch (NullPointerException e) {
			name = "InitialState1";
		}
		int policy;
		try {
			policy =
Integer.valueOf(ctx.policyInside().Num().getText()); //gets the policy of the
state
		} catch (NullPointerException e) {
			policy = -1;
		}
		XmlState s = new XmlState(name, true, policy); //creates the
state and adds it in the arraylist
		g.s.add(s);
		g.initial = s; //add the state as the initial state
		return super.visitInitial(ctx);
	}

	@Override
	public T visitName(ParseXmlParser.NameContext ctx) {
		return super.visitName(ctx);
	}
```

```
	@Override
	public T visitNormalState(ParseXmlParser.NormalStateContext ctx) {
		String name;
		try {
			name = ctx.name().words().getText(); //gets the name of a
state
		} catch (NullPointerException e) {
			name = "";
		}
		int policy;
		try {
			policy =
Integer.valueOf(ctx.policyInside().Num().getText()); //gets the policy of the
state
		} catch (NullPointerException e) {
			policy = -1;
		}
		XmlState s = new XmlState(name, false, policy); //creates the
state and adds it in the arraylist
		g.s.add(s);
		return super.visitNormalState(ctx);
	}

	@Override
	public T visitNormalStates(ParseXmlParser.NormalStatesContext ctx) {

		if ((g.s.size() != 0) && (g.s.get(g.s.size() - 1).id == -1) &&
g.s.get(g.s.size() - 1).initial == false) {
			XmlState s = g.s.get(g.s.size() - 1);  //gives identity
to normal states
			s.id = state;
			state++;

		}

		return super.visitNormalStates(ctx);
	}

	@Override
	public T visitPermissions(ParseXmlParser.PermissionsContext ctx) {
		permissions = new ArrayList<String>(); //creates the arraylist
of the permissions
		return super.visitPermissions(ctx);
	}

	@Override
	public T visitPolicies(ParseXmlParser.PoliciesContext ctx) {

		if ((g.p.size() != 0) && (g.p.get(g.p.size() - 1).id == -1)) {
			XmlPolicy p = g.p.get(g.p.size() - 1); //gives identity
to policies
			p.id = policy;
			policy++;

		}

		return super.visitPolicies(ctx);
	}
```

```
	@Override
	public T visitPolicy(ParseXmlParser.PolicyContext ctx) {
		String name;
		try {
			name = ctx.name().words().getText(); //gets the name of
the policy
		} catch (NullPointerException e) {
			name = "";
		}
		XmlPolicy p = new XmlPolicy(name); //creates the policy and adds
it to the arraylist
		g.p.add(p);
		return super.visitPolicy(ctx);
	}

	@Override
	public T visitPolicyInside(ParseXmlParser.PolicyInsideContext ctx) {
		return super.visitPolicyInside(ctx);
	}

	@Override
	public T visitPrivateData(ParseXmlParser.PrivateDataContext ctx) {
		String name;
		try {
			name = ctx.name().words().getText(); //gets the name of
the private data
		} catch (NullPointerException e) {
			name = "";
		}

		int policy;
		try {
			policy =
Integer.valueOf(ctx.policyInside().Num().getText()); //gets the policy in
which the data belongs to
		} catch (NullPointerException e) {
			policy = -1;
		}

		XmlPrivateData d = new XmlPrivateData(name, null, policy);
//creates the private data and adds it to the arraylist
		g.d.add(d);
		return super.visitPrivateData(ctx);
	}

	@Override
	public T visitPrivateDatas(ParseXmlParser.PrivateDatasContext ctx) {

		if ((g.d.size() != 0) && (g.d.get(g.d.size() - 1).id == -1)) {
			XmlPrivateData d = g.d.get(g.d.size() - 1);  //gives
identity to the data
			d.id = data;
			data++;
			ArrayList<String> access = null;
			if (permissions != null) {
				access = permissions;
				permissions = null;
```

```
                }
                g.d.get(g.d.size() - 1).permissions = access;

            }
            return super.visitPrivateDatas(ctx);
        }

        @Override
        public T
visitPrivateDataInside(ParseXmlParser.PrivateDataInsideContext ctx) {
            return super.visitPrivateDataInside(ctx);
        }

        @Override
        public T visitState(ParseXmlParser.StateContext ctx) {
            return super.visitState(ctx);
        }

        @Override
        public T visitToAction(ParseXmlParser.ToActionContext ctx) {
            return super.visitToAction(ctx);
        }

        @Override
        public T visitToState(ParseXmlParser.ToStateContext ctx) {
            return super.visitToState(ctx);
        }
}
```

State, Action, Policy, Graph and Private data classes created to model the policy in memory.

```
public class XmlState {
        boolean initial;
        String name;
        int policy;
        int id;

        public XmlState(String name, boolean initial, int policy) {
            this.name = name;
            this.id = -1;
            this.policy = policy;
            this.initial = initial;

        }
}

public class XmlPolicy {
        String name;
        ArrayList<Integer> data;
        int id;

        public XmlPolicy(String name) {
            this.name = name;
            this.data = new ArrayList<Integer>();
```

```
			id=-1;
		}
	}

	public class XmlAction {
		int id;
		String name;
		int from;
		int to;
		String condition;

		public XmlAction(String name, String condition, int from, int to) {
			this.name = name;
			this.condition = condition;
			this.from = from;
			this.to = to;
			id = -1;
		}

		@Override
		public boolean equals(Object obj) {

			if (obj instanceof XmlAction)
				if ((this.name.equals(((XmlAction) obj).name)) &&
	this.condition.equals(((XmlAction) obj).condition)) {
					return true;
				}
			return super.equals(obj);
		}

		@Override
		public int hashCode() {
			 int j;
			 j=name.hashCode()+condition.hashCode();
			return j;
		}
	}

	public class XmlGraph {
		String name;
		ArrayList<XmlAction> a;
		ArrayList<XmlState> s;
		ArrayList<XmlPolicy> p;
		ArrayList<XmlPrivateData> d;
		XmlState initial;
		boolean user;

		public XmlGraph() {
			user=false;
			 a= new ArrayList<XmlAction>();
			 s=new ArrayList<XmlState>();
			 p=new ArrayList<XmlPolicy>();
			 d=new ArrayList<XmlPrivateData>();
		}
	}

	public class XmlPrivateData {
		String name;
```

```
	ArrayList<String> permissions;
	int id;
	int policy;

	public XmlPrivateData(String name, ArrayList<String> permissions,int
policy) {
		this.name = name;
		this.permissions = permissions;
		this.policy=policy;
		id=-1;
	}
}
```

And finally the main method.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Pattern;

import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.ParseTree;

public class JavaParser {
	public static void main(String[] args) throws Exception {
		try {
			File file = new File("ObeoErrorFile.txt");
			PrintStream fileOut = new PrintStream(file);
			// System.setOut(fileOut);
			System.setErr(fileOut);
		} catch (FileNotFoundException e1) {
			e1.printStackTrace();
		}
		ArrayList<XmlGraph> program = new ArrayList<XmlGraph>();
		String input=new File(".").getCanonicalPath();
		File dir = new File(input+"\\runtime-EclipseApplication"); //
the workspace of obeo designer // where the models are
		File[] files = dir.listFiles();
		for (int i = 0; i < files.length; i++) {
			if (files[i].isDirectory()) { // if the file is a
directory the program opens it to check inside
				File[] insideFiles = files[i].listFiles();
				for (int j = 0; j < insideFiles.length; j++) {
					if
(insideFiles[j].getName().endsWith(".statemachinemodeller")) {

						XmlGraph g =
callAntlr(files[i].getName(), insideFiles[j].getName(),input); // call ANtlr
to produce
```

```
                        // the parsetree
                                                String n = insideFiles[j].getName();
                                                n = n.toLowerCase();
                                                if (n.startsWith("user-")) { // check
if it is a user file or regular
                                                       g.user = true;
                                                       n = n.substring(5);
                                                }

                                                String names[] = n.split("[.]");
                                                g.name = names[0];
                                                program.add(g);
                                         }
                                  }
                           } else {
                                  if
(files[i].getName().endsWith(".statemachinemodeller")) {
                                         XmlGraph g = callAntlr(files[i].getName(),
null,input);
                                         String n = files[i].getName();
                                         n = n.toLowerCase();
                                         if
(files[i].getName().toLowerCase().equals("user-")) {
                                                g.user = true;
                                                n = n.substring(5);
                                         }
                                         String names[] = n.split("[.]");
                                         g.name = names[0];
                                         program.add(g);
                                  }

                           }
                    }
                    printPolicy(program); // print the policy as text
                    System.setErr(System.err);

             }

             /**
              * The main method for printing the policy. Uses the arraylist of
graphs to
              * produce the policy language
              *
              * @param p the arraylist of graphs/state machines of each entity
              */
             private static void printPolicy(ArrayList<XmlGraph> p) {
                    PrintWriter writer = null;
                    try {
                           writer = new PrintWriter("policy.txt", "UTF-8"); // the
output file
                    } catch (FileNotFoundException | UnsupportedEncodingException e)
{
                           System.out.println("Output file could not be
created.Program will exit");
                           System.exit(0);
                           e.printStackTrace();
```

```
			}
			writer.println("Actions{");
			printActions(p, writer);
			writer.println("}");
			writer.println("User{");
			printUser(p, writer);
			writer.println("}");
			writer.println("Entities{");
			printEntities(p, writer);
			writer.println("}");
			writer.println("Transitions{");
			printTransitions(p, writer);
			writer.println("}");
			writer.close();

		}

		/**
		 * The method that prints the transitions of the policy.Uses the
action
		 * id->object from and to to do so.
		 *
		 * @param p      the arraylist of graphs/state machines of each entity
		 * @param writer The Printwriter of the output file
		 */
		private static void printTransitions(ArrayList<XmlGraph> p,
PrintWriter writer) {

			for (int i = 0; i < p.size(); i++) {
				XmlGraph g = p.get(i);
				writer.println(p.get(i).name + "{");
				writer.println(g.name + "_" + g.initial.name + ".");
				ArrayList<XmlAction> actions = g.a;
				for (XmlAction a : actions) {
					printTransition(a, g.s, g, writer); // print each
transition
				}
				writer.println("}");
			}
		}

		/**
		 * Prints a single transition.Uses the action from and to integers to
locate the
		 * id of the from and to states and prints them
		 *
		 * @param a      The action that its transition will be printed.
		 * @param s      The arraylist of states.
		 * @param g      The graph/state machine that we are currently on.
		 * @param writer The Printwriter of the output file
		 */
		private static void printTransition(XmlAction a, ArrayList<XmlState>
s, XmlGraph g, PrintWriter writer) {

			XmlState from = null;
			XmlState to = null;
			for (XmlState state : s) {
				if (state.id == a.from)
```

```
                        from = state;
                    if (state.id == a.to)
                        to = state;

                }

                if (a.from == -2)
                    from = g.initial;
                if (a.to == -2)
                    to = g.initial;
                writer.println(g.name + "_" + from.name + ":" + a.name + ":" +
g.name + "_" + to.name + ".");

        }

        /**
         * Print the policies of the entities.
         *
         * @param p      The arraylist of state machines
         * @param writer The Printwriter of the output file
         */
        private static void printEntities(ArrayList<XmlGraph> p, PrintWriter
writer) {

                for (int i = 0; i < p.size(); i++) {

                    if (!p.get(i).user) {
                        writer.println(p.get(i).name + "{");
                        printStates(p.get(i), writer);
                        writer.println("}");
                    }
                }

        }

        /**
         * Prints the policies of the owner of the data that the policies are
giving
         * permissions to.
         *
         * @param p      The arraylist of state machines
         * @param writer The Printwriter of the output file
         */
        private static void printUser(ArrayList<XmlGraph> p, PrintWriter
writer) {
                boolean user = false;
                for (int i = 0; i < p.size(); i++) {
                    if (p.get(i).user) {
                        writer.println(p.get(i).name + "{");
                        user = true;
                        printStates(p.get(i), writer);
                        writer.println("}");
                    }
                }

                if (!user) {
                    System.err.println("No user graph inside the system.");
                    System.exit(0);
```

```
			}

		}

		/**
		 * Print each policy of an entity.
		 *
		 * @param g      The state machine that we want to print the policies
from.
		 * @param writer The Printwriter of the output file
		 */
		private static void printStates(XmlGraph g, PrintWriter writer) {
			ArrayList<XmlState> states = g.s;
			for (int j = 0; j < states.size(); j++) {
				writer.print(g.name + "_" + states.get(j).name + "\t\t" +
">>" + "\t\t");
				printData(g, states.get(j), writer);
				writer.println();

			}

		}

		/**
		 * Prints the data and the permissions of a policy
		 *
		 * @param graph  The state machine that we want to print the data
from.
		 * @param state  The state/policy whose data will be printed
		 * @param writer The Printwriter of the output file
		 */
		private static void printData(XmlGraph graph, XmlState state,
PrintWriter writer) {
			int policy = state.policy;
			int counter = 0;
			boolean check = false;
			for (XmlPrivateData p : graph.d) {

				if (p.policy == policy) {
					check = true;
					if (counter == 0)
						writer.print(p.name + "\t" + "[");
					else {
						writer.println();
						writer.print("\t\t\t\t\t" + p.name + "\t" +
"[");
					}
					int counter2 = 0;
					for (String s : p.permissions) {
						if (counter2 > 0) {
							String t =
s.replaceFirst(Pattern.quote("."), "("); // replace the "." of obeo designer
with the parenthesis of the language
							t = t.replaceFirst(Pattern.quote("."),
")");

							writer.print("," + t);
						} else {
```

```
                                        String t =
s.replaceFirst(Pattern.quote("."), "("); // replace the "." of obeo designer
with the parenthesis of the language
                                        t = t.replaceFirst(Pattern.quote("."),
")");
                                        writer.print(t);
                                  }
                                  counter2++;
                            }
                            writer.print("]");
                            counter++;
                      }

               }
               if (!check)
                     writer.print("\t[]"); // if the policy is empty (no data
permissions) we print "[]".
               writer.println(".");

        }

        /**
         * Prints the actions of all state machines.Uses set to eliminate
duplication.
         *
         * @param p      The araylist containing all graphs/state machines.
         * @param writer The Printwriter of the ouptut file.
         */
        private static void printActions(ArrayList<XmlGraph> p, PrintWriter
writer) {
               Set<XmlAction> a = new HashSet<XmlAction>();
               for (int i = 0; i < p.size(); i++) {
                     ArrayList<XmlAction> actions = p.get(i).a;
                     for (int j = 0; j < actions.size(); j++) {
                           a.add(actions.get(j));
                     }
               }
               for (XmlAction t : a) {
                     writer.println(t.name + ":[" + t.condition + "];");
               }

        }

        /**
         * The current method calls antlr to produce the parse tree and
creates a
         * visitor instance to use the MyVisitor file.
         *
         * @param name  The name of the file inside the working space
         * @param name2 If the name file is a directory the name of the file
will also
         *              be sent here elsewhere null will be given.
         * @return the state machine of the current file.
         */
        private static XmlGraph callAntlr(String name, String name2,String
input) {
               org.antlr.v4.runtime.CharStream stream = null;
               try {
```

```
                    if (name2 != null)
                          stream = CharStreams.fromFileName(input+"\\runtime-
EclipseApplication\\" + name + "\\" + name2);
                    else
                          stream = CharStreams.fromFileName(input+"\\runtime-
EclipseApplication\\" + name);
             } catch (IOException e) {
                    System.out.println("Something went terribly wrong with
the file handling.\nProgram will exit");
                    e.printStackTrace();
                    System.exit(0);
             }
             ParseXmlLexer lexer = new ParseXmlLexer(stream);
             CommonTokenStream tokens = new CommonTokenStream(lexer);
             ParseXmlParser parser = new ParseXmlParser(tokens);
             parser.setBuildParseTree(true);
             ParseTree tree = parser.program();
             @SuppressWarnings("rawtypes")
             XmlMyVisitor visitor = new XmlMyVisitor();
             visitor.visit(tree);
             XmlGraph g = visitor.g;
             if (name2 == null)
                    checkErrors(name, g);
             else
                    checkErrors(name + "//" + name2, g);
             return g;
      }

      /**
       * Check for simple errors and warnings and tries to resolve the
warnings.
       *
       * @param name the filename.
       * @param g    the state machine that will be checked
       */
      private static void checkErrors(String name, XmlGraph g) {
             String warnings = "\n";
             String errors = "\n";
             boolean error = false;

             for (int i = 0; i < g.a.size(); i++) {
                    XmlAction a = g.a.get(i);
                    if (a.name.equals("")) {
                          warnings += "The action who was declared in the " +
Integer.toString(a.id + 1)
                                       + " line of actions has no name and
will be named for you\n";
                          a.name = "Action" + Integer.toString(a.id);
                    }
                    if (a.from == -1) {
                          errors += "The action who was declared in the " +
Integer.toString(a.id + 1)
                                       + " line of actions has no from
state\n";
                          error = true;
                    }
                    if (a.to == -1) {
```

```
                        errors += "The action who was declared in the " +
Integer.toString(a.id + 1)
                                + " line of actions has no to
state\n";
                        error = true;
                    }

            }
            if (!warnings.equals("\n"))
                System.out.print("Warning:" + warnings);
            if (!errors.equals("\n"))
                System.err.print(errors);
            warnings = "\n";
            errors = "\n";
            for (int i = 0; i < g.s.size(); i++) {
                XmlState s = g.s.get(i);
                if (s.name.equals("")) {
                    warnings += "The state who was declared in the " +
Integer.toString(s.id + 1)
                            + " line of states has no name and
will be named for you\n";
                    s.name = "State" + Integer.toString(s.id);
                }
                if (s.policy == -1) {
                    errors += "The state who was declared in the " +
Integer.toString(s.id + 1)
                            + " line of states has no policy\n";
                    error = true;
                }
            }
            if (!warnings.equals("\n"))
                System.out.print("Warning:" + warnings);
            if (!errors.equals("\n"))
                System.err.print(errors);
            warnings = "\n";
            errors = "\n";

            for (int i = 0; i < g.p.size(); i++) {
                XmlPolicy p = g.p.get(i);
                if (p.name.equals("")) {
                    warnings += "The action who was declared in the " +
Integer.toString(p.id + 1)
                            + " line of actions has no name and
will be named for you\n";
                    p.name = "Action" + Integer.toString(p.id);
                }
            }
            if (!warnings.equals("\n"))
                System.out.print("Warning:" + warnings);
            warnings = "\n";

            for (int i = 0; i < g.d.size(); i++) {
                XmlPrivateData d = g.d.get(i);

                if (d.name.equals("")) {
                    errors += "The data who was declared in the " +
Integer.toString(d.id + 1)
```

```
                                        + " line of private data has no name
and can not be named by the parser\n";
                                d.name += "Data" + Integer.toString(d.id);
                                error = true;
                        }
                        if (d.permissions == null) {
                                warnings += "The data who was declared in the " +
Integer.toString(d.id + 1)
                                        + " line of private data has no
permissions and can be deleted\n";
                        }

                        if (d.policy == -1) {
                                errors += "The data who was declared in the " +
Integer.toString(d.id + 1)
                                        + " line of private data isn't
connected with a policy\n";
                                error = true;
                        }

                }
                if (!warnings.equals("\n"))
                        System.out.print("Warning:" + warnings);
                if (!errors.equals("\n"))
                        System.err.print(errors);
                if (error) {
                        System.out.println("Please fix the errors in the file " +
name + " in order to proceed");
                        System.exit(0);
                }
                System.out.println("\nNo errors in file " + name);

        }
}
```

Appendix B

The grammar of ANTLR that parse the formal language, and the Java code that synthesizes the partial policies and produces the graph and the user-friendly text policy.

Firstly, the graph, action, edge,state and private data classes that help us represent the policy in memory.

Action.java

```
public class Action {
	String action;
	String condition;

	public Action(String name, String condition) {
		action = name;
		this.condition = condition;

	}



	@Override
	public boolean equals(Object obj) {

		if (obj instanceof Action)
			if ((this.action.equals(((Action) obj).action)) &&
this.condition.equals(((Action) obj).condition)) {
				return true;
			}
		return super.equals(obj);
	}

	@Override
	public int hashCode() {
		 int j;
		 j=action.hashCode()+condition.hashCode();
		return j;
	}
}
```

Edge.java

```
public class Edge {
	Action action;
```

```
	State from;
	State to;
	Graph g;

	public Edge(State from, Action a, State to) {
		this.from = from;
		this.action = a;
		this.to = to;

	}

	@Override
	public boolean equals(Object obj) {

		if (obj instanceof Edge)
			if (this.action.equals(((Edge) obj).action)) {
				return true;
			}
		return super.equals(obj);
	}

	@Override
	public int hashCode() {
		return super.hashCode();
	}
}
```

Graph.java

```
public class Graph {
	ArrayList<State> states;
	String name;
	State initial;
	ArrayList<Edge> edges;

	public Graph(String name) {
		this.name = name;
		states = new ArrayList<State>();
		edges=new ArrayList<Edge>();
	}
}
```

PrivateData.java

```
public class PrivateData {
	String name;
	ArrayList<String> permissions;

	public PrivateData(String data) {
		name=data;
	}
	@Override
	public boolean equals(Object obj) {

		if (obj instanceof PrivateData)
```

```
                if (this.name.equals(((PrivateData) obj).name)) {
                        return true;
                }
            return super.equals(obj);
        }

        @Override
        public int hashCode() {
                int j;
                j=name.hashCode();
               return j;
        }
}
```

State.java

```
public class State implements Comparable<State>{
        String name;
        ArrayList<PrivateData> statePolicy;
        ArrayList<Edge> fromState;
        ArrayList<Edge> toState;
        Graph g;

        public State(String name) {
               this.statePolicy = new ArrayList<PrivateData>();
               fromState = new ArrayList<Edge>();
               toState = new ArrayList<Edge>();
               this.name = name;
        }

        @Override
        public boolean equals(Object obj) {

               if (obj instanceof State) {
                       if (((State) obj).name.equals(this.name))
                               return true;
               }
               return false;
        }

        @Override
        public int hashCode() {
               int j = 0;
               j=name.hashCode();
               return j;
        }

        @Override
        public int compareTo(State arg0) {
               return this.name.compareTo(arg0.name);
        }
}
```

MyVisitor.class visits the produced tree of ANTRL and transforms the policy from text to memory graphs.

```
import java.util.ArrayList;
import java.util.Set;

import org.antlr.v4.runtime.tree.ErrorNode;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.RuleNode;
import org.antlr.v4.runtime.tree.TerminalNode;

public class MyVisitor<T> extends PrivacyPolicyBaseVisitor<T> {

	CombinedStateMachine sm;
	ArrayList<String> tempperm;
	Graph graph;

	public MyVisitor() {
		sm = new CombinedStateMachine();
		tempperm = null;
	}

	@Override
	public T visit(ParseTree tree) {
		return super.visit(tree);
	}

	@Override
	public T visitChildren(RuleNode arg0) {
		return super.visitChildren(arg0);
	}

	@Override
	public T visitErrorNode(ErrorNode node) {
		return super.visitErrorNode(node);
	}

	@Override
	public T visitTerminal(TerminalNode node) {
		return super.visitTerminal(node);
	}

	@Override
	public T visitAccess(PrivacyPolicyParser.AccessContext ctx) {
		if (ctx.access() == null) {
			tempperm.add(ctx.accMod().getText());
			Graph g = sm.graphs.get(sm.graphs.size() - 1);
			State s = g.states.get(g.states.size() - 1);
			PrivateData d = s.statePolicy.get(s.statePolicy.size() -
1);
			d.permissions = tempperm;
			tempperm = null;
			return super.visitAccess(ctx);
		}
		tempperm.add(ctx.accMod().getText());
		return super.visitAccess(ctx);
```

```
}

@Override
public T visitAccMod(PrivacyPolicyParser.AccModContext ctx) {
	return super.visitAccMod(ctx);

}

@Override
public T visitAction(PrivacyPolicyParser.ActionContext ctx) {
	return super.visitAction(ctx);
}

@Override
public T visitActions(PrivacyPolicyParser.ActionsContext ctx) {
	return super.visitActions(ctx);
}

@Override
public T visitCinside(PrivacyPolicyParser.CinsideContext ctx) {
	return super.visitCinside(ctx);
}

@Override
public T visitConstraint(PrivacyPolicyParser.ConstraintContext ctx) {
	return super.visitConstraint(ctx);
}

@Override
public T visitEdge(PrivacyPolicyParser.EdgeContext ctx) {
	return super.visitEdge(ctx);
}

@Override
public T visitEdges(PrivacyPolicyParser.EdgesContext ctx) {
	if (ctx.edge() == null)
		return super.visitEdges(ctx);
	String fromS = ctx.edge().Word(0).getText();
	String actionS = ctx.edge().Word(1).getText();
	String toS = ctx.edge().Word(2).getText();

	State from =getState(fromS, graph);
	Action a= getAction(actionS);
	State to =getState(toS, graph);
	Edge e= new Edge(from,a,to);
	graph.edges.add(e);
	e.g=graph;
	return super.visitEdges(ctx);
}

private Action getAction(String actionS) {
	Set<Action> actions = sm.actions;
	for (Action a : actions) {
		if (a.action.equals(actionS))
			return a;
	}
	return null;
}
```

```
	@Override
	public T visitEntities(PrivacyPolicyParser.EntitiesContext ctx) {
		return super.visitEntities(ctx);
	}

	@Override
	public T visitGlobalPolicy(PrivacyPolicyParser.GlobalPolicyContext
ctx) {
		return super.visitGlobalPolicy(ctx);
	}

	@Override
	public T visitP(PrivacyPolicyParser.PContext ctx) {
		if (ctx.p1() == null)
			return super.visitP(ctx);
		PrivateData p = new PrivateData(ctx.p1().Word().getText());
		Graph g = sm.graphs.get(sm.graphs.size() - 1);
		State s = g.states.get(g.states.size() - 1);
		s.statePolicy.add(p);
		tempperm = new ArrayList<String>();
		return super.visitP(ctx);
	}

	@Override
	public T visitP1(PrivacyPolicyParser.P1Context ctx) {
		return super.visitP1(ctx);
	}

	@Override
	public T visitPermissions(PrivacyPolicyParser.PermissionsContext ctx)
{
		return super.visitPermissions(ctx);
	}

	@Override
	public T visitPolicies(PrivacyPolicyParser.PoliciesContext ctx) {
		if (ctx.policy() == null)
			return super.visitPolicies(ctx);
		State p = new State(ctx.policy().Word().getText());
		sm.graphs.get(sm.graphs.size() - 1).states.add(p);
		p.g=sm.graphs.get(sm.graphs.size() - 1);
		return super.visitPolicies(ctx);
	}

	@Override
	public T visitPolicy(PrivacyPolicyParser.PolicyContext ctx) {
		return super.visitPolicy(ctx);
	}

	@Override
	public T visitSconstraint(PrivacyPolicyParser.SconstraintContext ctx)
{
		return super.visitSconstraint(ctx);
	}

	@Override
```

```
	public T visitTransitions(PrivacyPolicyParser.TransitionsContext ctx)
{
		return super.visitTransitions(ctx);
	}

	@Override
	public T visitUser(PrivacyPolicyParser.UserContext ctx) {
		return super.visitUser(ctx);
	}

	@Override
	public T visitActions1(PrivacyPolicyParser.Actions1Context ctx) {
		if (ctx.action() == null)
			return super.visitActions1(ctx);
		Action a = new Action(ctx.action().Word().getText(),
ctx.action().constraint().getText());
		if (sm.checkDuplicateActionName(a)) {
			System.out.println("You can't have the same action name
with different condition,please rename the action "
					+ a.action);
		}

		sm.actions.add(a);
		return super.visitActions1(ctx);
	}

	@Override
	public T visitGraph(PrivacyPolicyParser.GraphContext ctx) {

		return super.visitGraph(ctx);
	}

	@Override
	public T
visitGraphsTransitions(PrivacyPolicyParser.GraphsTransitionsContext ctx) {
		return super.visitGraphsTransitions(ctx);
	}

	@Override
	public T
visitGraphTransitions(PrivacyPolicyParser.GraphTransitionsContext ctx) {
		if (ctx == null)
			return super.visitGraphTransitions(ctx);
		String graphname = ctx.Word().getText();
		this.graph = getGraph(graphname);
		String state = ctx.initial().Word().getText();
		State s = getState(state, graph);
		this.graph.initial = s;
		return super.visitGraphTransitions(ctx);
	}

	private State getState(String state, Graph graph) {
		ArrayList<State> states = graph.states;
		for (State s : states) {
			if (s.name.equals(state))
				return s;
		}
		return null;
```

```
	}

	private Graph getGraph(String graphname) {
		ArrayList<Graph> graphs = sm.graphs;
		for (Graph g : graphs) {
			if (g.name.equals(graphname))
				return g;
		}
		return null;
	}

	@Override
	public T visitGraphs(PrivacyPolicyParser.GraphsContext ctx) {
		if (ctx.graph() == null)
			return super.visitGraphs(ctx);
		Graph g = new Graph(ctx.graph().Word().getText());
		sm.graphs.add(g);
		return super.visitGraphs(ctx);
	}
}
```

The Combined State, Combined Edge and Combined State Machine classes responsible to represent the global policy after the synthesis.

CombinedEdge.java

```
public class CombinedEdge {
	Action action;
	CombinedState from;
	CombinedState to;


	public CombinedEdge(CombinedState from,Action a,CombinedState to) {
		this.from=from;
		this.action=a;
		this.to =to;

	}


}
```

CombinedState.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class CombinedState {
	Set<State> states;
```

```
	Set<PrivateData> policy;
	boolean visited;
	int id;

	public CombinedState(Set<State> states) {
		policy = new HashSet<PrivateData>();
		visited = false;
		id = 0;
		visited = false;
		this.states = states;
	}

	@Override
	public boolean equals(Object obj) {

		if (obj instanceof CombinedState) {
			for (State s : ((CombinedState) obj).states) {
				if (!(this.states.contains(s))) {
					return false;
				}
			}
		} else
			return false;
		return true;
	}

	@Override
	public int hashCode() {
		int j = 0;

		int counter = 1;
		List<State> al = new ArrayList<State>();
		al.addAll(states);
		Collections.sort(al);
		for (State s : al) {
			j+=(s.hashCode()*counter);
			counter++;
		}
		return j%Integer.MAX_VALUE;
	}

	public void printState() {
		System.out.println("State");
		for (State s : states) {
			System.out.println(s.name);
		}
	}

	public void setVisit(boolean b) {
		this.visited = b;

	}

	public boolean getVisited() {
		return this.visited;
	}

}
```

CombinedStateMachine.java

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

public class CombinedStateMachine {
	ArrayList<Graph> graphs;
	ArrayList<CombinedEdge> edges;
	Set<Action> actions;
	ArrayList<PrivateData> data;
	Set<CombinedState> states;
	ArrayList<CombinedState> clearstates;
	ArrayList<CombinedEdge> clearedges;
	int policyCounter = 1;

	public CombinedStateMachine() {
		graphs = new ArrayList<Graph>();
		edges = new ArrayList<CombinedEdge>();
		actions = new HashSet<Action>();
		data = new ArrayList<PrivateData>();
		states = new HashSet<CombinedState>();
	}

	public boolean checkDuplicateActionName(Action a) {

		for (Action sm : this.actions) {
			if (sm.action.equals(a.action) &&
(!(sm.condition.equals(a.condition)))) {
				return true;
			}
		}
		return false;
	}

	public void printEdges() {
		for (CombinedEdge e : edges) {
			System.out.println("Edge :");
			e.from.printState();
			System.out.println(e.action.action);
			e.to.printState();
			System.out.println();
		}
	}

	public void printStateMachine() {
		for (CombinedState cs : states)
			cs.printState();
	}

	public void prettyPrintPolicy() {
		PrintWriter writer = null;
		try {
```

```
                    writer = new PrintWriter("prettypolicy.txt", "UTF-8"); //
the output file
             } catch (FileNotFoundException | UnsupportedEncodingException e)
{
                    System.out.println("Output file could not be
created.Program will exit");
                    System.exit(0);
                    e.printStackTrace();
             }
             ArrayList<CombinedState> tempList = new
ArrayList<CombinedState>();
             Set<State> set = new HashSet<State>();
             for (Graph g : graphs) {
                    set.add(g.initial);
             }
             CombinedState initial = null;
             for (CombinedState cs : states) {
                    cs.visited = false;
                    if (cs.states.equals(set)) {
                           initial = cs;
                    }
             }

             Set<CombinedState> tempSet = new HashSet<CombinedState>();
             tempList.add(initial);
             tempSet.add(initial);
             initial.id = policyCounter++;

             while (expand(tempList, tempSet, writer)) {
             }
             for (CombinedEdge ce : edges) {
                    for (CombinedState cs : tempSet) {
                           if (ce.to.equals(cs)) {
                                  ce.to.id = cs.id;
                           }
                    }
             }
             writer.flush();
             writer.close();

      }

      private boolean expand(ArrayList<CombinedState> tempList,
Set<CombinedState> tempSet, PrintWriter writer) {
             if (tempList.isEmpty())
                    return false;
             CombinedState checked = tempList.get(0);
             tempList.remove(0);
             for (CombinedState cs : states) {
                    if (cs.equals(checked)) {//here
                           cs.id = checked.id;
                           checked = cs;
                    }
             }
             printCombinedState(checked, tempList, tempSet, writer);
             writer.println();
             return true;
      }
```

```
	private void printCombinedState(CombinedState checked,
ArrayList<CombinedState> temp, Set<CombinedState> tempSet,
			PrintWriter writer) {
		ArrayList<CombinedEdge> edges = new ArrayList<CombinedEdge>();
		for (CombinedEdge ce : this.edges) {
			if (ce.from.equals(checked)) {
				edges.add(ce);
			}
		}
		printStatePolicy(checked, writer);
		writer.println();
		for (CombinedEdge e : edges) {

			if (!tempSet.contains(e.to)) {
				e.to.id = policyCounter++;
				temp.add(e.to);
				tempSet.add(e.to);
			}
			printEdge(e, tempSet, writer);
		}

	}

	private void printEdge(CombinedEdge e, Set<CombinedState> tempSet,
PrintWriter writer) {
		CombinedState nextpolicy = null;
		for (CombinedState cs : tempSet) {
			if (e.to.equals(cs)) {
				nextpolicy = cs;
			}
		}
		writer.println("After the " + e.action.action + " action the
policy with id " + nextpolicy.id
				+ " will take the place of the current policy");

	}

	private void printStatePolicy(CombinedState checked, PrintWriter
writer) {
		writer.println("The policy with id " + checked.id + " has the
following permissions on data: ");

		for (State s : checked.states) {
			printPolicy(s, writer);
		}

	}

	private void printPolicy(State s, PrintWriter writer) {
		String[] temp = s.name.split("_");
		int counter = 1;
		writer.println("The entity " + temp[0] + " can :");
		if (s.statePolicy.isEmpty()) {
			writer.println(counter + ". Do nothing");
		} else {
			for (PrivateData d : s.statePolicy) {
				writer.print(counter++ + ". (");
```

```
                        int tempcounter = 0;
                        for (String st : d.permissions) {
                              if (tempcounter == 0)
                                    writer.print(st);
                              else
                                    writer.print("," + st);
                              tempcounter++;
                        }
                        writer.println(") " + "on private data " + d.name);
                  }
            }

      }

      public void printStates() {
            for (CombinedState cs : states) {
                  for (State s : cs.states) {
                        System.out.println(s.name);
                  }
                  System.out.println();
            }

      }

      public void addState(CombinedState cs) {
            states.add(cs);

      }

      public void addEdge(CombinedEdge ce) {
            edges.add(ce);

      }

      public void clearEdges() {
            clearedges = new ArrayList<CombinedEdge>();
            boolean found = false;
            for (CombinedEdge e : edges) {
                  for (CombinedEdge ce : clearedges) {
                        if ((e.action.equals(ce.action)) && (e.from.id ==
ce.from.id) && (e.to.id == ce.to.id)) {
                              found = true;
                              break;
                        }
                  }
                  if (found) {
                        found = false;
                  } else {
                        clearedges.add(e);
                  }
            }

      }
      public void clearStates() {
            clearedges = new ArrayList<CombinedEdge>();
            boolean found = false;
            for (CombinedEdge e : edges) {
                  for (CombinedEdge ce : clearedges) {
```

```
                        if ((e.action.equals(ce.action)) && (e.from.id ==
ce.from.id) && (e.to.id == ce.to.id)) {
                                found = true;
                                break;
                        }
                }
                if (found) {
                        found = false;
                } else {
                        clearedges.add(e);
                }
        }

    }

    public void visualizeGraph() {
        PrintWriter writer = null;
        try {
                writer = new PrintWriter("graph.dot", "UTF-8"); // the
output file
        } catch (FileNotFoundException | UnsupportedEncodingException e)
{
                System.out.println("Output file could not be
created.Program will exit");
                System.exit(0);
                e.printStackTrace();
        }
        writer.println("digraph finite_state_machine {");
        writer.println("rankdir=TB;");

        writer.println("node[height=\"2\"];");
        writer.println("size=\"20\"");
        writer.print("node [shape = circle]; ");

        clearEdges();
        for (CombinedEdge ce : clearedges) {
                writer.println(
                                "State" + ce.from.id + " -> " + "State" +
ce.to.id + " [ label = \"" + ce.action.action + "\" ];");
        }
        writer.println("}");
        writer.flush();
        writer.close();

    }
}
```

The main method combining everything together.

ParseText.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;

public class ParseText {

	public static Set<Graph> graphnames;


	public static void main(String[] args) {

		try {
			File file=new File("ErrorFile.txt");
			PrintStream fileOut = new PrintStream(file);
			//System.setOut(fileOut);
			System.setErr(fileOut);
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}

		org.antlr.v4.runtime.CharStream stream = null;
		try {

			stream = CharStreams.fromFileName(args[0]);

		} catch (IOException e) {
			System.out.println("Something went terribly wrong with
the file handling.\nProgram will exit");
			e.printStackTrace();
			System.exit(0);
		}
		PrivacyPolicyLexer lexer = new PrivacyPolicyLexer(stream);
		CommonTokenStream tokens = new CommonTokenStream(lexer);
		PrivacyPolicyParser parser = new PrivacyPolicyParser(tokens);
		parser.setBuildParseTree(true);
		ParseTree tree = parser.globalPolicy();
		System.err.println();//
		@SuppressWarnings("rawtypes")
		MyVisitor visitor = new MyVisitor();
		visitor.visit(tree);
		CombinedStateMachine sm = visitor.sm;
		synthesise(sm);
```

```
		sm.prettyPrintPolicy();
		sm.visualizeGraph();
		System.setErr(System.err);


	}

	private static void synthesise(CombinedStateMachine sm) {
		ArrayList<Graph> graphs = sm.graphs;
		graphnames = getGraphnames(sm);
		Set<State> set = new HashSet<State>();
		for (Graph g : graphs) {
			set.add(g.initial);
		}
		CombinedState cs = new CombinedState(set);
		sm.addState(cs);
		while (!exploreState(sm)) {
		}
		//System.out.println("finished");
		// sm.printEdges();
		// System.out.println(sm.states.size());
		// sm.printStates();

	}

	private static boolean exploreState(CombinedStateMachine sm) {
		CombinedState explore = null;
		boolean finished = true;
		for (CombinedState cs : sm.states) {
			if (!(cs.getVisited())) {
				explore = cs;
				finished = false;
			}
		}
		if (!finished) {
			expand(explore, sm);
			explore.setVisit(true);
		}
		return finished;

	}

	private static void expand(CombinedState explore, CombinedStateMachine
sm) {
		ArrayList<Edge> allEdges = new ArrayList<Edge>();
		for (State s : explore.states) {
			for (Edge e : s.g.edges) {
				if (e.from.equals(s)) {
					allEdges.add(e);
				}
			}
		}
		while (allEdges.size() != 0) {
			// System.out.println(counter++);
			Edge e = allEdges.get(0);
			if (!checkForSync(e, sm, allEdges)) {
				allEdges.remove(0);
				continue;
```

```
                    }
                    allEdges.remove(0);
                    ArrayList<Edge> same = getSynchronised(e, allEdges);

                    Set<Graph> graphsJump = new HashSet<Graph>(); // set of
the graphs to check that we put one state from each

                          // state machine in the combined state
                    graphsJump.addAll(graphnames);
                    Set<State> newstates = new HashSet<State>(); // the new
state that will be created
                    newstates.add(e.to); // adds the 'to' state of the
examined edge and removes the graph from the set.
                    graphsJump.remove(e.g);
                    addSyncStates(same, newstates, graphsJump, allEdges);
                    for (Graph g : graphsJump) {
                          for (State cs : explore.states) {
                                if (cs.g.equals(g)) {
                                      newstates.add(cs);
                                      break;
                                }
                          }
                    }
                    CombinedState cs = new CombinedState(newstates);

                    //System.out.println(cs);
                    //System.out.println(sm.states.contains(cs));
                    sm.states.add(cs); // adds the new state if it doesnt
already exists;
                    CombinedEdge ce = new CombinedEdge(explore, e.action,
cs);
                    sm.addEdge(ce);

              }

        }

        private static void addSyncStates(ArrayList<Edge> same, Set<State>
newstates, Set<Graph> graphsJump,
                    ArrayList<Edge> allEdges) {
              for (Edge e : same) {
                    newstates.add(e.to);
                    graphsJump.remove(e.g);
                    allEdges.remove(e);
              }

        }

        private static Set<Graph> getGraphnames(CombinedStateMachine sm) {
              Set<Graph> graphnames = new HashSet<Graph>();
              for (Graph g : sm.graphs) {
                    graphnames.add(g);
              }
              return graphnames;
        }

        private static ArrayList<Edge> getSynchronised(Edge e, ArrayList<Edge>
allEdges) {
```

```
		ArrayList<Edge> same = new ArrayList<Edge>();
		for (Edge en : allEdges) {
			if (en.equals(e)) {
				same.add(en);
			}
		}
		return same;

	}

	private static boolean checkForSync(Edge e, CombinedStateMachine sm,
ArrayList<Edge> allEdges) {
		boolean sync = true;
		ArrayList<Edge> syncEdges = new ArrayList<Edge>();
		for (Graph g : sm.graphs) {
			if (g.equals(e.g))
				continue;
			for (Edge en : g.edges) {
				if (en.equals(e)) {
					sync = false;
					syncEdges.add(en);
				}
			}
		}
		if (!sync) {
			for (Edge synced : syncEdges) {
				boolean found = false;
				for (Edge checked : allEdges) {
					if ((checked.action.equals(synced.action))
&& (checked.g.equals(synced.g))) {
						found = true;
						break;
					}

				}
				if (!found) {
					return sync;
				}
			}
			return true;
		}
		return sync;
	}


}
```

Appendix C