Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

POLITECNICO
MILANO 1863

A Model-based Graphical Designer for Engineering Forensic-ready Systems

Relatore: Prof. Luciano BARESI

Correlatore: Dr. Liliana PASQUALE

Tesi di laurea di:

Luigi Marco LATONA Matr. 836568

Anno Accademico 2016–2017

Abstract

Nowadays there is an increasing trend towards embedding computational capabilities into devices to make them effectively communicate and perform useful tasks. This is the so-called pervasive computing environment and, since software systems are becoming more pervasive, there is the need of protecting such systems from incidents perpetrated by malicious individuals. Unfortunately, it is not feasible to prevent all possible incidents because they are increasing in number and complexity and this is the reason why a digital investigation should be performed to explain how an incident occurred and who was responsible for it. A forensic-ready system would help in the investigation minimising the costs and using the relevant digital evidence. In particular, such system is able to monitor data that can be useful to satisfy or refuse an incident hypothesis before it happens. The objective of the thesis is developing a graphical designer for engineering forensic-ready system that helps describing the environment and some incident hypotheses that may occur within the environment. These descriptions are then used by an external tool that generates rules to preserve the relevant data for the investigation. The graphical designer aims to solve some limitations of the tool concerning usability, correctness of the descriptions and reusability of the software. To address these problems, I use the Model-Driven Engineering (MDE) approach which allows substituting code with meta-models and specifically it ensures correction of the models based on the meta-model by design. I use the frameworks Eclipse Modeling Framework (EMF) for the creation of the meta-model and Graphical Modeling Framework (GMF) to build a graphical designer that facilitates the representation of the environment and incident hypotheses addressing the usability limitation. Finally, to foster reusability, the graphical designer was

implemented as an Eclipse plugin. The approach has been evaluated by describing the environment and the incident hypotheses of three scenarios inspired by existing digital forensic corpora.

Sommario

Nella società odierna si sta verificando una crescita verso l'integrazione di capacità computazionale nei dispositivi elettronici per facilitarne la comunicazione ed eseguire operazioni utili. Questo tipo di integrazione costituisce l'aspetto caratterizzante dei sistemi pervasivi. La crescita nell'utilizzo di sistemi pervasivi sta anche rendendo più necessario proteggere questi sistemi da incidenti a scopo criminale. Sfortunatamente, prevenire tutti i possibili incidenti non è una soluzione possibile poiché sono sempre più numerosi e complessi. Per questo motivo, dopo che un incidente avviene in un sistema software, deve essere eseguita un'investigazione digitale per spiegare come un incidente è avvenuto e chi ne è il responsabile. Un sistema forensic-ready aiuterebbe in tale investigazione, minimizzando i costi e utilizzando solo dati pertinenti all'incidente. In particolare, tale sistema permetterebbe di monitorare dati che possono essere utili per dimostrare o rifiutare le ipotesi di un potenziale incidente prima che questo avvenga. L'obiettivo del lavoro di tesi è sviluppare un designer grafico per lo sviluppo di sistemi software forensic-ready. L'obiettivo di un designer grafico e' quello di facilitare la rappresentazione del sistema in cui un incidente può avvenire e delle ipotesi di possibili incidenti. Tali rappresentazioni sono indispensabili per creare la specifica software che un sistema forensic-ready deve soddisfare. Tale specifica prescrive quali dati un sistema forensic-ready deve preservare e quando. Durante una investigazione i dati preservati possono essere utili per dimostrare come uno degli incidenti rappresentati è avvenuto. Il designer grafico ha come obiettivo la risoluzione di alcuni limiti che presenta un tool che genera questo tipo di specifiche. Questi limiti riguardano l'usabilità, la correttezza delle descrizioni e la riusabilità del software. Per risolvere questi problemi utilizzo un approccio di ingegneria gui-

data dal modello (MDE) che permette la sostituzione di codice con meta-modelli. Precisamente, assicura la corretta definizione di modelli, dato che si basano sul meta-modello, che descrivono il sistema e le ipotesi di incidenti che possono avvenire in quel sistema. Utilizzo i frameworks Eclipse Modeling Framework (EMF) per la creazione del meta-modello e Graphical Modeling Framework (GMF) per sviluppare un designer grafico che facilita la definizione del sistema e degli incidenti risolvendo così il problema dell'usabilità. Infine, per affrontare il problema della riusabilità, il designer grafico è stato implementato come un plugin per Eclipse. L'approccio è stato testato descrivendo l'ambiente e le ipotesi di incidenti di tre scenari ispirati da data-sets disponibili online che descrivono scenari di incidente realistici.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

1.1 The context

Nowadays there is an increasing trend towards embedding computational capabilities into devices to make them effectively communicate and perform useful tasks. A pervasive computing environment can be defined as *one saturated with computing and communication capabilities, yet so gracefully integrated with users that it becomes a "technology that disappears"* [1]. Pervasive computing applications and devices have been increasingly adopted to support public services, such as transport or health, as well as for private use, such as for online payment. With software systems becoming more pervasive, an increasing number of assets, which are transmitted, manipulated, or stored digitally, are being compromised by incidents perpetrated by malicious individuals.

An incident can be defined as *an anomalous behaviour of a system that suggests a violation or imminent threat of a violation of policies (e.g., security or use policies) or regulations* [2]. The German Steel Mill cyber attack is a famous example of a security incident. More precisely, the attacker(s) infiltrated the steel facility using phishing emails taking control of the physical equipment and ultimately causing the furnace to shut down. [3].

Unfortunately, it is not feasible to prevent all possible incidents because they are increasing in number and complexity. Moreover when an incident happens it is important to start a digital investigation to explain how an incident occurred and who was responsible for it. More precisely, a digital investigation is defined as *"the collection, preservation, analysis, interpretation and presentation of digital data from digital sources, for proof of incident and ultimately for prosecution of criminal activity"* [4].

However, performing a digital investigation is difficult. For an organisation to perform a digital investigation there is the need of trained staff to understand what data should be collected to support the hypotheses of an investigation. They should also be aware of the companies internal policies and regulations protecting the privacy of data collected. The digital data that need to be analysed during an investigation can be volatile and be easily lost or distorted because the collection has not been planned. Therefore, there is a need to preserve data that might be an evidence during an investigation before they might be tampered with by an offender or be lost [5].

Although digital forensic investigations are usually performed after an incident occur, some of their activities can be executed proactively before an incident occurs to reduce the costs and the time of potential future digital forensic investigations.

Forensic-readiness is indeed the ability of minimise such costs and it has been defined as *"the ability of an organisation to maximise its potential to use digital evidence whilst minimising the costs of an investigation*" [6].

Therefore, building a forensic-ready system that is capable of preserving in advance data that is important for a future digital investigation can speed-up the entire investigation process and minimise its costs [7].

1.2 The problem

Existing research has been performed to identifying guidelines for building forensic-ready systems. For example Rowlingson [6] describes the 10 key activities to build a forensic-ready software system. These include the identification of available sources

of digital evidence and the data to be collected during an investigation.

Another approach [8] describes a model to build a framework for an enterprise that ensures forensic-readiness. The construction of the framework is driven by the main scope of the model that is "*preserving the ability to prosecute malicious cyber intrusion successfully, while reducing current effort expended on digital forensic investigations*".

Forensic-readiness aims to develop systems that are prepared to support digital investigations of **major incidents** that are classified as such when it is reasonable to suspect that an incident brings to an intrusion, loss/theft of data or misuse of an organisation's ICT system. In particular, since the risk for a major incident to happen is high and it can cause serious damages, a forensic-ready system should forbid any activity that could alter or destroy a proof of evidence when a major incident occurs. [9]. Major incidents could differ depending on the environment they can happen and they are not the same for all the domains. Some likely scenarios could be: denial of service attack, loss or theft of a significant amount of personal information, compromised host resulting in unauthorised programs or processes running on the host, copy of sensitive or access restricted files.

Unfortunately most of the research performed on the development of forensic-ready systems only captures operational and infrastructural capabilities for organisations to achieve forensic readiness. Operational capabilities refer to the provision of training and equipment to the people involved in the investigation. Infrastructural capabilities instead refer to the collection of all the relevant data for the investigation [10]. What it lacks here is a methodology capable of defining how a software system that operates in a specific domain could be forensic-ready.

An existing work [4] in this direction focuses the attention to the data preservation phase of a digital investigation. In particular, there is a tool called kEEPER (on EvidEnce PrEservation Requirements for forensic-ready systems) that automatically generates specifications that satisfy the preservation requirements of forensic-ready systems. The FR requirements ensure that, if an incident occur, the data to explain how the incident took place are preserved. A preservation specification is given to a

software component, the Forensic Readiness (FR) Controller that receives the data observable from the digital sources in the environment. The preservation specification identifies the conditions that enforce the preservation of the data received by the FR Controller. The tool requires the users (e.g., software engineers, system administrators) to provide a description of the environment in which incidents may occur and hypotheses explaining how such incidents can happen and automatically generates the preservation specification. However, kEEPER, had the following limitations:

- **Usability:** the tool is based on a command-line interface and it requires the user to define the environment and the incident hypotheses in a predicate logic language, such as the Event Calculus. Therefore it is very hard for the final users, who are not aware of the syntax of the language, to use the kEEPER tool correctly.
- **Correctness:** the tool does not provide any guide for the definition of the environment and incident hypotheses in Event-Calculus. Therefore, it is very hard to generate correct preservation specifications
- **Reusability:** the tool does not provide any mechanism to support modifications in the future. Therefore it would be a complex task if there was the need of any extension.

1.3 Thesis objectives

The work I have done in this thesis aims to provide an approach to make the developing of forensic-ready systems more usable. To achieve this objective I want to develop a graphical designer for engineering forensic-ready systems. In particular, my work provides kEEPER with a graphical designer for the description of the environment and incident hypotheses.

The objectives of my thesis are to solve the limitations of kEEPER listed in the previous paragraph concerning usability, correctness and reusability. In particular, I use a model-driven approach that allows to substitute code with meta-models and two important frameworks that are part of the Eclipse Projects: EMF (Eclipse Modeling

Framework) that I use to build the meta-model and GMF (Graphical Modeling Framework) that I use to develop the graphical designer.

To address the **usability** problem, I have provided a graphical interface: all the entities of the environment and the hypotheses have a graphical representation associated, each of them can be defined inside specific diagrams. The translation of the diagrams into Event-Calculus is then automated by an algorithm of the software.

I ensure **correctness** by design because my graphical designer is based on the concepts of model-driven engineering. This ensures that all the instances built on top of the meta-model are correct by design. Moreover, since the user is obliged to use specific graphical elements and the translation into Event-Calculus is automatic, it is impossible to make mistakes.

I address the **reusability** problem by implementing the graphical designer as an Eclipse plugin. A plugin is defined as: *executable program that extends and strengthens the function of the software without changing the platform* [11]. This allows the graphical designer to be extended and modified easily in the future.

The effective functionality of the graphical user interface has been actually demonstrated and evaluated through the instantiation of three different incident scenarios inspired by existing digital forensic corpora [12] [13].

1.4 Thesis structure

This thesis is structured as follows:

- In the second chapter I describe the approach used by kEEPER to generate the preservation specifications. I analyse the tool in its main aspects and I provide an explanation of how it works.

- In the third chapter I illustrate the design choices I made to build the graphical designer. In particular, I describe the frameworks I used to build the meta-model and its graphical representation.

- In the fourth chapter I describe the technical solution: from the meta-model

used to the graphical elements of the graphical designer and eventually I describe the parser that finally outputs the results into Event-Calculus.

- In the fifth chapter I explain the three case studies I performed to evaluate the effectiveness of the graphical designer. Then I describe the contributions I made to the project and I discuss some limitations that the graphical designer has and the possible extensions that could be added to be improved.

- In the sixth chapter I list other works done in this field. I describe a domain-specific language project and other forensic-readiness approaches.

Chapter 2

Forensic-Ready Systems

In this chapter I first illustrate some related works. Then I motivate the use of the approach using an example and in the last section I describe the approach in more detail.

2.1 Related works

A lot of research has been done in the field of forensic-readiness but no one at the moment has defined a systematic approach about how a forensic-ready system should be designed. In particular, in spite of the existence of the forensic readiness process standardisation (ISO/EIC 27043:2015 [14]) where the collection of possible incidents is prescribed, the only approaches that nowadays exist focus the attention to a high-level definition of a forensic-ready system without implementing it.

Reddy and Venter, for example, present an architecture for a digital forensic-readiness management system that promises to achieve an optimal level of management for digital forensic-readiness [15].

Shield et al. [16] propose continuous data preservation within an organisation but sometimes it is not a feasible solution because it could be hard to analyse all the data . Pasquale et al. propose instead an approach where evidence preservation aims to detect only potential attack scenarios that can lead to the violation of security policies [17]. However this approach prescribes to preserve any type of event that happen

and lead to an incident without considering previous events that have occurred or preserved.

2.2 Motivating example

I use a motivating example to justify the need of using the approach. The example describes an environment within an enterprise building. There are two employees, *bob* and *alice*, that have two laptops, *m2* and *m3*, respectively. A desktop *m1* is located in the room *r01* and it stores a sensitive document *doc*. Access to the room *r01* is controlled by an NFC reader *nfc* and it is recorded by a CCTV camera *cctv*. Both *alice* and *bob* have the authorisations to access the room *r01* and to login to *m1*. The environment is illustrated in the figure 2.1.

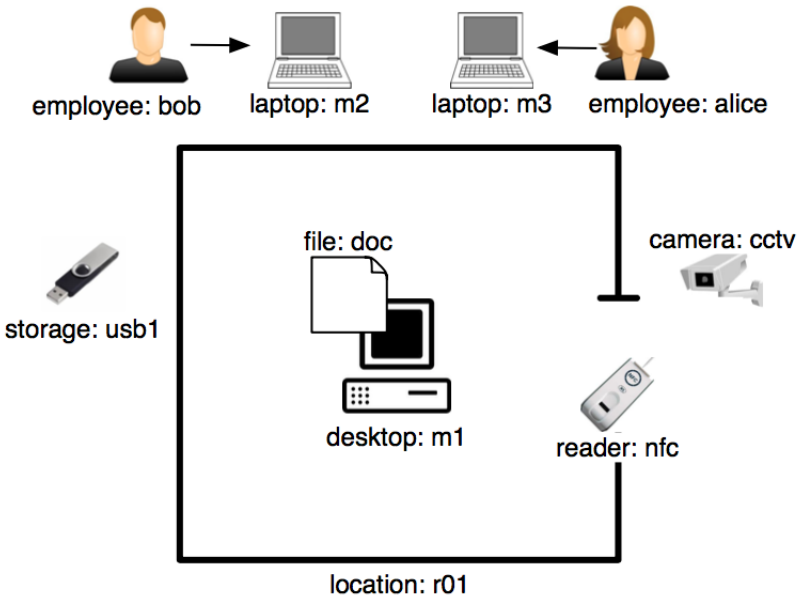


Figure 2.1: Motivating example environment

Supposing that the document *doc* has been leaked, a digital investigation initiates and the investigator hypothesises that the document *doc* has been copied into an external device that has been mounted in the computer *m1*. At this point, the investigator tries to list the possible scenarios that led to the incident hypothesis. Once the scenarios have been identified, the investigator needs to understand which

digital device contains the relevant information to support the scenarios. Then, in this example, the investigator has to search through the logs of the system to look for a recently mounted device or he/she has to look for the accesses in the room recorder by *nfc1* and *cctv* devices. Unfortunately, the large number of events that can happen (a lot of people enter in the room, a lot of devices have been mounted in the computer recently) and the impossibility of storing all events caused by the inability of monitoring them all, makes this investigation not feasible. Furthermore, there is the need of a systematic approach that automates the preservation of the relevant events.

2.3 Description of the tool

The systematic approach generates specification for software systems that are forensic-ready. It does not perform all the activities of an investigation process but it only prescribes which data is relevant and has to be collected.

The approach, described by the figure 2.2, needs a domain expert to give the description of the **environment** ε where the incidents can occur, one or more **hypotheses** $\mathcal{H}$ describing possible incident scenarios and an initial **preservation specification** *PS*. It automates the synthesis of *preservation specifications* for systems to meet the *preservation requirements* that explicitly prescribe preservation of the minimum amount of data that are necessary to support the hypotheses of an investigation.

Taken those as inputs, the tool based on a satisfiability solver and a logic-based learner, generates a preservation specification. In particular the approach provides as output either:

1. a confirmation that (some) hypotheses are not supported in the environment.
2. a confirmation that the FR controller does not have the capabilities to monitor data in the preservation specification.
3. a modified preservation specification that guarantees to satisfy its preservation requirements with respect to the hypotheses in the given environment.

Regarding the third case, after the preservation specification is generated, the **Forensic-Ready Controller** (FR) receives data from the devices in the environment and eventually it decides whether to store the information in a secure storage. The information is then used for the investigation by an investigator.

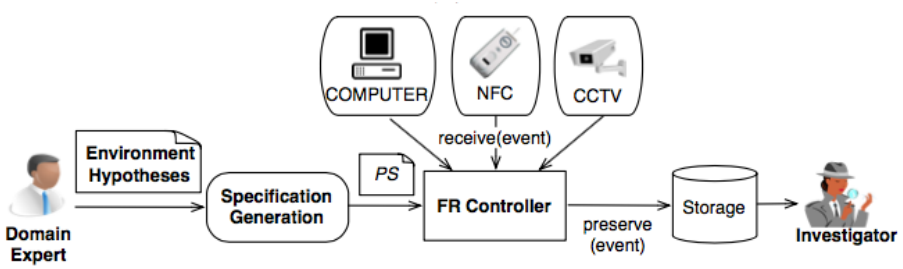


Figure 2.2: Tool Mechanism

2.4 Environment Description

The description of the environment is written in Event-Calculus that is *a logic-based formalism for representing actions and their effects* [18]. The definition of the environment is a set of descriptive statements about: the context in which an incident may occur, the behaviour that may be exhibited within the environment and their interactions. It includes a context and a behavioural description.

The first part is a collection of descriptive declarations about the **types** and **instances** of the entities present in the environment and about the **context relations** that define the state of the environment.

The behavioural description specifies the events that may occur within the environment. We distinguish between two types of events: **primitive events** that represent the occurrence of an atomic action that can be observed by an investigator from a digital device and **complex events** that indicate the execution of complex human activities and that can involve one or more primitive events.

2.4.1 Types and Instances

The description of the environment starts from the definition of all types and instances. The types can be computers, devices, readers, people. Each instance instead has a type associated to it and it represents indeed that particular computer, person.

The correspondent statement in Event-Calculus stating that *m1*, *m2* and *m3* are computers, is:

```
comp(m1; m2; m3)
```

2.4.2 Context Relations

What it follows is the definition of the status of the system described by the context relations or fluents. All the instances need to be somehow correlated to each other and this is partially achieved with the context relations that tell you which is the **status of the system**. A fluent specifies the current state of the system and it is used in the composite definition section to specify if and when a particular relation between instances holds. One possible context relation between the types "employee" and "location" could be for example "in(emp,loc)" and it specifies a relation between two instances of those types. It states that an employee is in a location.

The statement in Event-Calculus is:

```
fluent(in(E,L)):-
        emp(E), loc(L).
```

2.4.3 Events

Events describe actions that can be performed within the environment.

Primitive events normally refer to actions that the system performs such as a system call that mounts an external device. Complex events are instead normally human actions always performed within the environment.

As for the fluents, events are defined with one or more parameters, each of them representing a "type". An example of primitive event could be "sys_mount(dev,

comp)" where "dev" indicates the type device and "comp" the type computer. An example of complex event could be instead "copy(emp, fi, comp)" where "emp" indicates the type employee, "fi" the type file and "comp" the type computer.

The statements in Event-Calculus for events sys_mount and copy follow:

```
pe(sys_mount(S, C)):-
        st(S), comp(C).


ce(copy(E, F, C)):-
        emp(E), fi(F), comp(C).
```

2.4.4 Composite Definitions

A composite definition is always associated with a complex event. It defines the preconditions for a complex event to occur. The preconditions are represented in the description file as a list of events (primitive and complex) and context relations. Events can only be associated with the predicate **"Happens**" while context relations can be associated with the predicates **"Holds At**" and **"Holds At Between"**. In particular, an event can only happen in a specific time instant and a context relation can hold in one or between more than one time instants. The predicates "Holds At" and "Holds At Between" can also assume the negative form in which case the precondition would be satisfied if the context relation is not holding.

In conclusion each predicate represents one precondition and all the predicates of each composite definition represents all the preconditions that need to be satisfied for the complex event (associated to the composite definition) to happen.

In a composite definition all the parameters of each event and context relation are associated to an existing type defined previously and a chronological order to the time instants is provided.

The following is an example of a composite definition:

```
happens(login(E,C),T,TR):-
        trace(TR),
```

```
        emp(E),
        comp(C),
        loc(L),
        emp(E2),
        time(T),
        happens(sys_Login(E,C),T,TR),
        holdsAt(in(E2,L),T,TR),
        holdsAt(isLocatedIn(C,L),T,TR),
        holdsAt(hasPermission(E,C),T,TR).
```

It is made up by essentially three parts: on the first line the "login(E,C)" indicates the complex event that is described in this composite definition. In the next six lines all parameters and time instants are defined and in particular each parameter is linked to the correspondent type and at the end, on the remaining lines all the preconditions are expressed. All the precondition statements are in logical AND so all of them must be true to allow the event "login" to occur.

It is important to notice that in this particular case two different employees are involved (E and E2) and, consequently two distinct definition statements are expressed in order to point out the distinction. Also, in this case only one time instant is defined and it means that all the events and context relations must happen/hold at the same time as the complex event "login".

2.4.5 Context Relation triggering conditions

This part of the descriptor file essentially states which complex event turns a context relation to be true or false. In particular, a complex event can initiates or terminates a complex event.

The statement below in Event-Calculus says: if an employee enters in a location, then s/he is in the location. In this case the context relation "in" turns to be true.

```
initiates(enter(E,L), in(E,L), T):-
        emp(E), loc(L), time(T).
```

A similar behaviour happens when the event "exit" occurs but in this case the context relation "in" is not satisfied anymore and it turns to be false. Its Event-Calculus notation is the following:

```
terminates(exit(E,L), in(E,L), T):-
        emp(E), loc(L), time(T).
```

2.4.6 Initial States

Each statement tells which context relation is true with which instances. An example of that is the following:

```
initially(hasBadge(bob, r01)).
```

Since Context Relation "hasBadge" has two parameters indicating the types employee and location, it needs two instances of those types that, in this example are employee "bob" and room "r01".

2.5 Hypothesis Description

As for the environment, the hypotheses of an incident should also be provided as input to the tool to create the preservation specifications. While the environment describes all the elements that are part of the considered system, the hypotheses describes a possible incident and it is syntactically defined in the same way as a composite definition. The tool, given the environment description is able to understand if an hypotheses is feasible (the incident could happen) or it is not, in which case the hypotheses cannot happen or the definition of the environment needs to be reviewed.

The following is an example in Event-Calculus of an hypotheses:

```
hypothesis(h1,T,TR):-
        trace(TR),
        time(T),
        emp(E),
```

```
st(S),
holdsAt(mounted(S,m1),T,TR),
happens(copy(E,doc,m1),T,TR).
```

The hypotheses is clearly referring to the incident in which:

- A storage device is mounted into the computer *m1*
- An employee is using the computer *m1* to perform a copy of a document *doc.*

2.6 Preservation Specifications

Preservation specifications (**PS**) prescribe when an event has to be stored by the FR controller and under which conditions. In particular, we are interested in the execution of the operations of the form "preserve(a, ts)" where *a* indicates the occurrence of a primitive event and *ts* represents the time-stamp at which the event occurs. Taken it into account the *PS* indicates the pre- and post-conditions that precede and follow the preservation of that event at the specified time-stamp.

The following is an example of *domain pre- and post- conditions* expressed in LTL (Linear Temporal Logic) for the preservation of the occurrence of the event *sys_ copy:*

$\forall ts$: *Timestamp, e : Emp, d : Doc, m : Comp*

$G(preserved(sys\_copy(e,d,m),\ ts) \rightarrow \neg preserve(sys\_copy(e,d,m),ts))$

$G(preserve(sys\_copy(e,d,m),\ ts) \rightarrow X preserved(sys\_copy(e,d,m),ts))$

The pre-condition (1) specifies that the preservation of the event cannot take place if it has already been preserved while the post-condition (2) indicates that, at the time instant following the occurrence of the event, this one has already been preserved.

The following instead, is an example of *required pre- and trigger-conditions.* The former conditions the execution of *preserve(a,ts)*, it basically says when an event can be collected. The latter indicates the conditions on the (non-)preservation of related events or, in other words, it says when an event must be collected.

Both the conditions are part of the preservation specifications generated by the tool.

$\forall ts : Timestamp,\ e : Emp,\ d : Doc,\ m : Comp$

$G(\neg received(sys\_copy(e,d,m),ts) \rightarrow X \neg preserve(sys\_copy(e,d,m),ts))$

$\forall ts : Timestamp,\ e : Emp,\ d : Doc,\ m : Comp$

$\exists ts_1, ts_2 : Timestamp. ts_1 < ts_2 \wedge ts_2 < ts$

$G(received(sys\_copy(e,d,m),\ ts) \wedge$

$preserved(sys\_login(e,d,m),\ ts_1) \wedge preserved(sys\_mount(s,m),\ ts_2) \wedge$

$\nexists ts_3, ts_4 : Timestamp.\ (ts_3 > ts_1 \wedge ts_3 \leq ts \wedge ts_4 > ts_2 \wedge ts_4 \leq ts \wedge$

$preserved(sys\_logout(e,m),\ ts_3) \wedge$

$preserved(sys\_unmount(s,m),\ ts_4)) \rightarrow Xpreserve(sys\_copy(e,d,m),\ ts))$

Chapter 3

Design Choices

In this chapter I describe the design choices I made to develop my software. First I describe how the Model-Driven Engineering approach works and why I decided to use it on my application. Later on I give an explanation of how the Eclipse environment is structured and how all its components interact between each other in order to provide a stable and fully functional environment where to build a software. Then I describe the two main frameworks my software is based on (Eclipse Modeling Framework and Graphical Modeling Framework) that are the main important concepts to be understood clearly.

3.1 Model-Driven Engineering

Approach to software development where models rather than programs are the principal outputs of the development process.

I consider the one above a good definition of what a Model-Driven Engineering (MDE) method signifies. Indeed, using this approach, it is possible to build a meta-model that can actually be reused afterwards to build various different programs for distinct aims.

The aim is to provide a meta-model that satisfies the requirements of a specific domain application and then, develop programs with different features on top of the meta-model. This results in code cheaper to maintain and adapt to new platforms

thanks to the generated code provided by an high level of abstraction.

The meta-model needs to "define the relationships among concepts in a domain and precisely specify the key semantics and constraints associated with these domain concepts" [19]; the developer can then use the elements of the meta-model building his/her own application adapted to the needs and being sure that all the constraints of the domain application are not violated.

3.1.1 Why the need of MDE

In the past, a lot of efforts have been made trying to obtain an higher level of abstraction while developing software.

The first step has been done by the creation of the so-called computer-aided software engineering (CASE) that allowed the developers to generate code by the use of general-purpose graphical programming representations such as state machines and data-flow diagrams [19]. At the time the current operating systems did not provide the support for QoS (Quality of Service) properties such as fault tolerance and security, consequently the generated code needed to face these problems and as a result it was dramatically increasing in complexity. Of course the objective of these tools was providing an approach to make the developing of a software easier and more understandable by using graphical elements. Unfortunately, the use of CASE implied also difficulty in the debugging and developing phase because the generated code was too complex for the reason I just explained. Furthermore, CASE was based on the generation of code starting from very simple graphical elements that began to be insufficient to build a complex system.

Recently, further important progresses have been made in high-level programming languages such as Java an C++ providing an higher abstraction level to the developers.

The reuse of the huge amount of class libraries minimises the need to reinvent middleware services like the ones I mentioned before (security and fault tolerance) but it brings in the problem of maintenance of big systems. Developers indeed, are too busy trying to understand APIs and platforms they are using because of their

complexity given by the high interaction between all the software components of a big system and they usually ends up ignoring some important issues such as the correctness.

An MDE approach solves the problem of correctness thanks to the generated code based on the meta-model.

3.1.2 How I used the Model-Driven Engineering approach

My project makes heavy use of this approach because the software is based on the meta-model that ensures the correctness of all entities and constraints between them. Therefore, thanks to MDE the software is correct by design and no violation in terms of domain application inconsistency can be done when building an instance of the environment/hypothesis description. In particular, thanks to the level of abstraction given by MDE method, it is possible to build a graphical designer that represents an intermediate step before the environment/hypothesis representation expressed in Event-Calculus, and that it gives a more user-friendly interface.

All of this is done with the help of two important frameworks that I describe in the next paragraphs, which work in the Eclipse environment and that heavily use the concepts of extensions and extension points of Eclipse.

3.2 Extensions and Extension Points

"*Eclipse is a good example of a modern component-based complex system that is designed for long-term evolution, due to its architecture of reusable and extensible components*" [20]. The Eclipse environment works thanks to plugins; all of its components are made up of plugins that are called **extensions**. All the extensions in Eclipse need to have some features that other extensions can use to modify somehow the functionalities of the plugin. In particular, within the Eclipse environment, these features are called **extension points** and all the extensions need to provide one or more of them. This is basically the most important characteristic of Eclipse that gives a very high level of extensibility and maintainability to the software. Fig-

ure 3.1 shows three different plugins (extensions) using one or more functionalities provided by the extension point of another plugin.

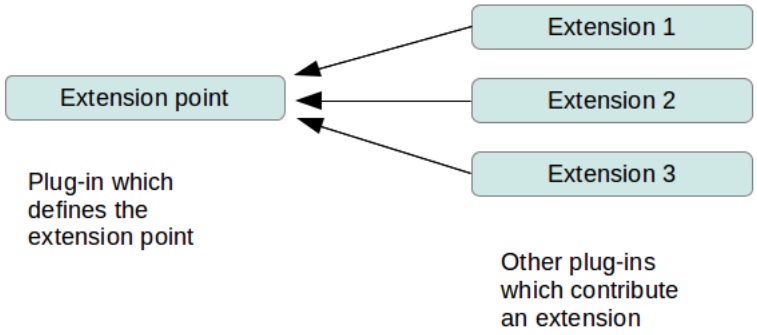


Figure 3.1: Extensions and Extension Point

Each plugin contains also a file called "plugin.xml" which declaratively states all the extension points needed for that extension. An example of the case would be the following:

```
<plugin>
   <extension point="org.eclipse.emf.ecore.generated_
          package">
      <!-- @generated model -->
      <package
            uri="https://github.com/lpasquale/kEEPER"
            class="model.ModelPackage"
            genModel="model/model.genmodel"/>
   </extension>
</plugin>
```

Inside the tag *<extension>* all of the extension points are stated. In the case above the correspondent extension point to which the plugin is attached is "org.eclipse.emf.ecore.generated_pa
The other tag *<package>* is only defining some attributes of the Java package of the plugin.

3.2.1 How I used Extensions and Extension Points

In my project I created several extensions that interact between each others and make the software works. In particular some of the extensions are used to make the meta-model works, other ones are used to make the graphical designer works and another one for the parsing of the instance of the meta-model into Event-Calculus representation.

Thanks to this design choice I guarantee reusability within other Eclipse developing environments and an easy future extensibility.

3.3 Eclipse Modeling Framework (EMF)

The main important decision in terms of design has been made when I chose to use the model-based framework called Eclipse Modeling Framework (**EMF**). "*EMF is a powerful framework and code generation facility for building Java applications based on simple model definitions*" [21]. It works within Eclipse and it heavily uses the concept of extensions and extension points.

EMF generates code automatically based on the definition of the meta-model which is defined by the developer. The meta-model is described in XMI (XML Metadata Interchange) which is a proposed use of XML to exchange metadata and in particular it fits perfectly for describing a model. EMF uses the XMI description to build Java classes for the meta-model and it provides also adapters for command-based editing functionalities in fact the creation/editing of an instance of the meta-model is handled by commands in a stack that are being executed following a FIFO policy.

In particular the description of the meta-model can be done using the hierarchical visualisation provided by EMF or writing the code by hand and it is stored in the ***.ecore file.** The next step is generating the ***.gencode file** that handles all the settings for generating the Java classes of the meta-model.

After the code generation, EMF provides the developer with a total amount of four plugins, all interacting between each others by extension points. The four

extensions created are the following:

- **Model Plugin:** all the Java entities are stored in this project. Here there are all the classes that need to be instantiated in order to create an instance of the meta-model.
- **Edit Plugin:** this plugin contains all the providers that can be used to show the entities within a UI. They are very useful if used with GMF, the other framework I use and that is described in the next paragraph.
- **Editor Plugin:** the editor extension provides basic editing functionalities for the instance of the meta-model.
- **Test Plugin:** this fourth plugin provides all classes to easily create tests for the software.

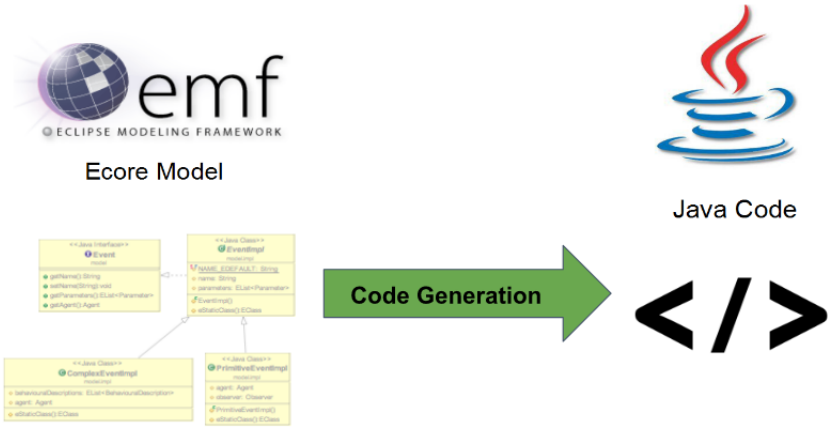


Figure 3.2: EMF Code Generation

The *.ecore* file describes the meta-model in XML. The basic more important elements that is possible to define are **EPackage**, **EClass**, **EEnum** that correspond to the classic Java Package, Class or Enum. In each EClass it is possible to define an **EAttribute**, **EReference** or **EOperation** that correspond to the Java Attribute, other Class Reference data structure, or Method. These are the main features you can define in the *.ecore* file that are later translated into Java source code. The *.ecore* file basically allows to build an instance of the "root" meta-model also called Ecore meta-model. The figure 3.3 illustrates the complete Class Diagram of the Ecore

meta-model. The root node **EObject** basically represents the java.lang.Object class of Java programming language.

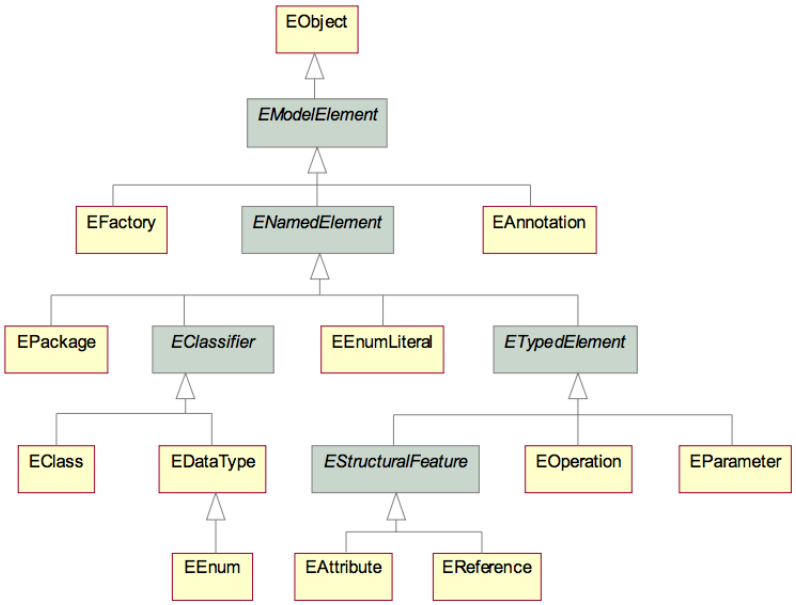


Figure 3.3: Ecore Meta-Model

The *.genmodel* file instead, describes how the source code has to be generated. It provides the platform specific information as opposed to the *.ecore* that is platform independent. It is possible to configure which packages to use and how to display the model structure.

3.3.1 How I used EMF

Thanks to EMF it is possible to automate the translation of the instance of the meta-model into Event-Calculus obtaining a description of the environment with no errors or violation of the domain application constraints. Down to the details EMF automatically generates all the classes in Java, one for each element of the environment and another extension (Editor plugin) that can be used to create an instance of the meta-model.

3.4 Graphical Modeling Framework (GMF)

The Graphical Modeling Framework (**GMF**) is a framework working within Eclipse and on top of EMF and it provides the developer with a graphical designer. The graphical designer extends the Editor plugin generated by EMF and it builds a designer based on the use of diagrams that helps the description of an instance of the meta-model. The use of GMF is important to provide the user administrator in charge of describing an instance, with a friendly user interface. Furthermore, GMF uses GEF (Graphical Editing Framework) that is a framework for graphical visualisation that is based on SWT from org.eclipse. When building a graphical designer within GMF, it is important being able to manage different files proper of the framework.

GMF works on different levels: from the definition of each graphical element to the code generation, each of them is a very important aspect of the designer creation.

3.4.1 Appearance

The first thing to define is the look-and-feel of every element in the diagram. This is done thanks to the descriptor file ***.gmfgraph** that, like the *.ecore file, can be represented in a hierarchical structure.

A *gmfgraph* file describes the canvas of the diagram: all the figures that can be dragged and dropped from the palette. In particular we distinguish some important elements:

- **Figure Gallery:** it contains all the figure descriptors that can be associated to images or general shapes.
- **Figure Descriptor:** it contains one figure that can be a regular shape (rectangle, line, circle, etc...), a vectorial image or a label.
- **Figure:** it describes the figure with all of its characteristics like vertices, edges, length/width and so on. It can also be of type “Custom” in which case it has to be modelled writing code by hand in an appropriate Java class.

- **Label:** it describes the text associated to each figure and its relative positioning nearby the figure.
- **Node:** this is probably the most important part. This feature represents the node in the diagram and it has to be linked to its Figure.
- **Connection:** it represents the link between nodes in the diagram.
- **Diagram Label:** the difference between this feature and the "Label" is essentially the same as Figure-Node. The Label represents the text to show while the Diagram Label specifies the element in the diagram. Besides, the diagram label is needed to access the Label that is normally a child of Figure.

The figure 3.4 is an easy example with only the node "Person" of how this file can be visualised and modified.

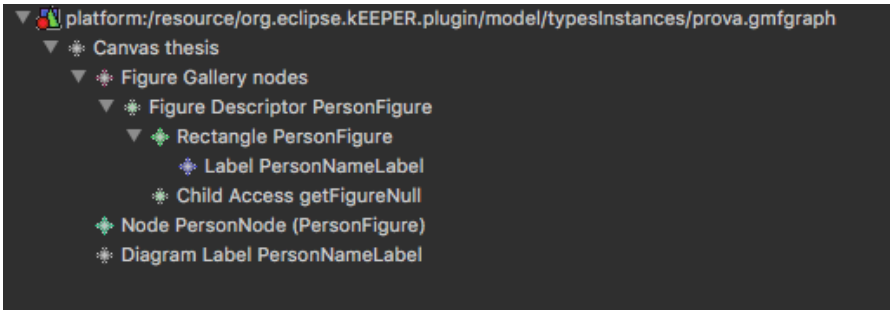


Figure 3.4: Hierarchical visualisation of a .gmfgraph file

3.4.2 Palette

After the look-and-feel, the palette is the second thing to be defined. The palette is substantially a portion of the working screen from which it is possible to drag-and-drop elements into the diagram. It is described by the file ***.gmftool** and, to be edited, it can be done from the same hierarchical visualisation of the *gmfgraph* file.

All the important elements are the following:

- **Palette:** it is the root node and only one can be created.
- **Tool Group:** it puts together similar elements. The developer can choose how to group the elements in the palette with this feature.

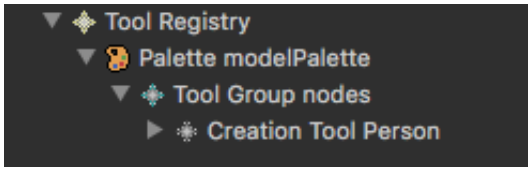


Figure 3.5: Hierarchical visualisation of a .gmftool file

- **Creation Tool:** it specifies the element that is possible to drop on the diagram.

The figure 3.5 is an easy example of how this file can be visualised and modified.

3.4.3 Mapping process

The third and probably the most important step constitutes in putting together all the previous elements within a mapping process. The *ecore, gmfgraph* and *gmftool* files are considered together in this phase. This process is performed with the help of another declarative file called ***.gmfmap** that considers each entity of the ecore file together with its appearance specified in the *gmfgraph* file and its correspondent element in the palette (*gmftoo*l). The mapping process is very important because the output file is at the basis for the creation of the source code for the designer that constructs the diagram. Figure 3.6 shows the mapping process schematically.

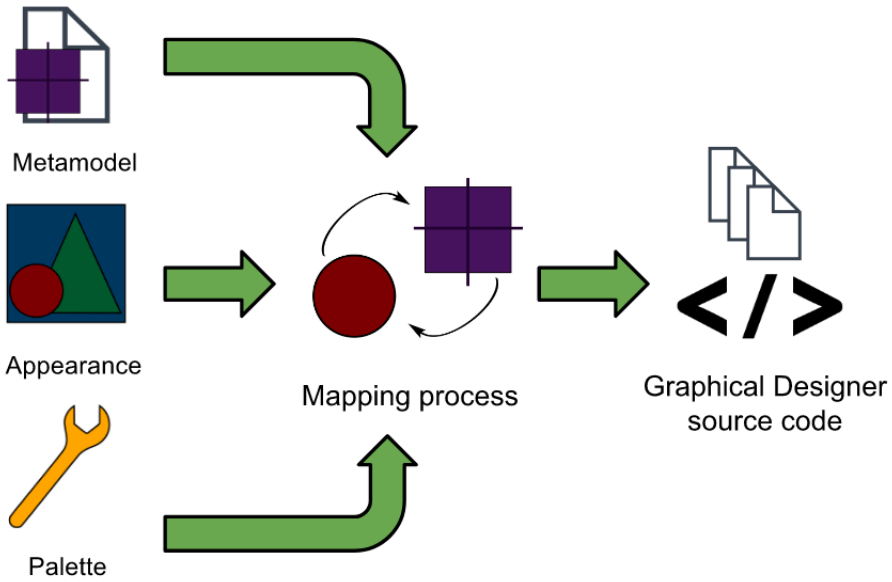


Figure 3.6: Mapping process graph

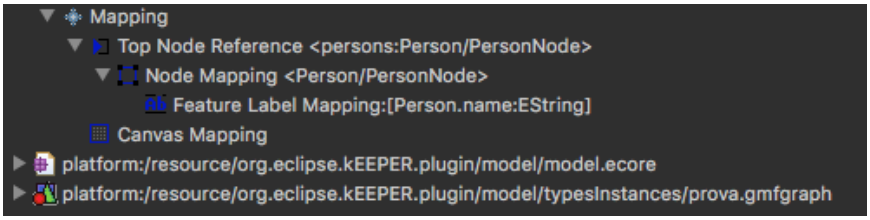


Figure 3.7: Hierarchical visualisation of a .gmfmap file

For the mapping process to work, the meta-model needs to provide some global data structures that contain all the entities of a specific type. These data structures are of type "**EList**" and they work similar to a more common java list. All the objects of each class created during the building of the diagram need to be stored in a specific global *EList*.

The following are the main features that need to be set in a *gmfmap* file:

- **Top Node Reference:** it specifies which is the global *EList* that the Node Mapping child element is referring to.

- **Node Mapping:** it basically puts together an entity from the meta-model in the *ecore* file with the look-and-feel of the node in the diagram expressed in the *gmfgraph* file and the palette element in the *gmftool* file. This is the core of the whole mapping process.

- **Feature Label Mapping:** it associates the Diagram Label of the *gmfgraph* file with a specific Java attribute of the entity mapped to the node in the diagram (mapping that is done in the Node Mapping feature).

Figure 3.7 illustrates a very basic example of how the *gmfmap* file appears when opened with the hierarchical visualisation.

When the mapping process has ended, a new Eclipse extension is automatically generated that interacts with the other plugins created by EMF and provides the user with a graphical interface thanks to which it is possible to build an instance of the meta-model visually expressed by a customisable diagram.

3.4.4 Graphical designer extension

The final result, as I said before, is the generation of the source code of the graphical designer. It constitutes another Eclipse extension that uses the meta-model and the Edit EMF plugins.

The plugins come with ten Java packages, each of them including classes to perform specific operations on the diagram. The most important are:

- ***.diagram.edit.commands:** each class of this package handles the command for the creation of a specific node/connection of the diagram.
- ***.diagram.edit.parts:** each instance of a class of this package represents each node/connection that belongs to the diagram. All the elements (node or connection) in the diagram are always associated to their EditPart objects: one per each element.
- ***.diagram.edit.policies:** the classes inside this package are in charge of handling the requests. All the operations (selection, resizing, moving) performed on an EditPart object are handled as requests.
- ***.diagram.navigator:** the classes in this package are responsible for the opening/closing of a diagram
- ***.diagram.part:** it contains the classes to manage the diagram (creation, editing, loading of external resources)
- ***.diagram.providers:** all the classes in this package are useful to handle the providers (part of the diagram workspace where it is possible to specify the attributes of the elements in the diagram) of the nodes/connections of the diagram.

In conclusion, what you obtain from the example I showed, is the most basic graphical designer that it is possible to build with GMF and it is illustrated in the figure 3.8. It represents the definition of an instance of the class Person.

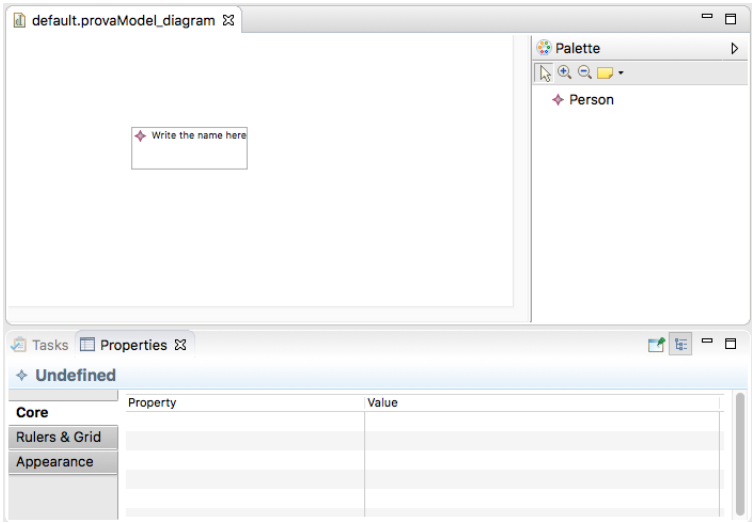


Figure 3.8: Basic graphical designer

3.4.5 How I used GMF

As it is easy to understand, The Graphical Modeling Framework helped me to develop a graphical designer that is able to describe the *environment* and the *incident hypotheses* with diagrams providing the user an easy-to-understand interface and a large range of definition possibilities. The graphical designer, being an Eclipse extension too, is easy to extend with other functionalities. In this case, as GMF stores the created diagrams also in XMI, it was easy to create a new plugin that parses the XMI files into Event-Calculus, providing finally a correct environment and hypotheses definition to the tool kEEPER.

Chapter 4

Technical Solution

In this chapter I go down to the details of my software. In particular, I describe the meta-model built within EMF and the plugins generated by GMF with the manual changes I was obliged to make. At the end I describe the encoding Eclipse plugin which has the important job of parsing the instances of the meta-model into Event-Calculus.

4.1 The Meta-Model

The meta-model is at the basis of everything concerning this software and it needs to be strong enough to provide a fully customisable graphical designer with GMF.

The solution I propose is summarised by the Class Diagram in figure 4.1.

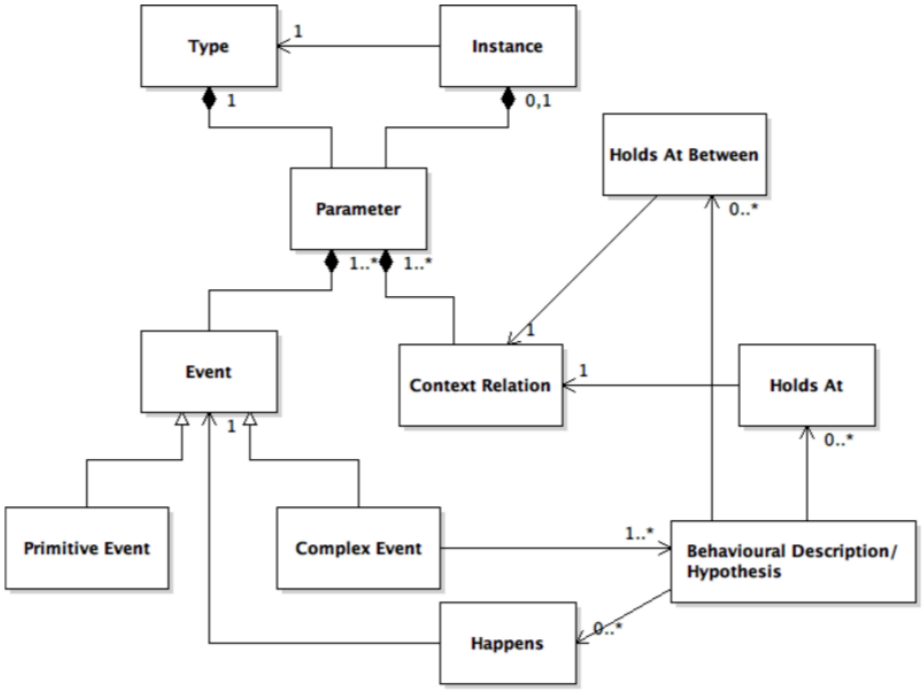


Figure 4.1: kEEPER Meta-Model

The EMF Model Plugin contains all the Java classes of the meta-model and it is composed by three packages: one for all the Java interfaces, a second one for all the implementation classes and a last one that handles the adapters for the entities in the second package. When an object needs to reference a list of other objects of the meta-model, EMF provides a data structure called **EList** which functionalities are similar to the common Java ArrayList.

4.1.1 Type and Instance

The first component of the meta-model surely is the Type-Instance relationship. Both instance and type have a "name" attribute that needs to be set by the final user.

Figure 4.2 shows how Type and Instance classes are related, in fact each instance can have only one type reference. For example, common names for type and instance could be "loc" and "r01" respectively that stand for "location" and the name of a room.

The Type class is extended by two different kind of Type classes: Agent and Observer. It is important to have this split here to manage better the parameters of

the events later.

The Agent is a kind of type that normally performs an action while an Observer type is somehow watching the action being performed.

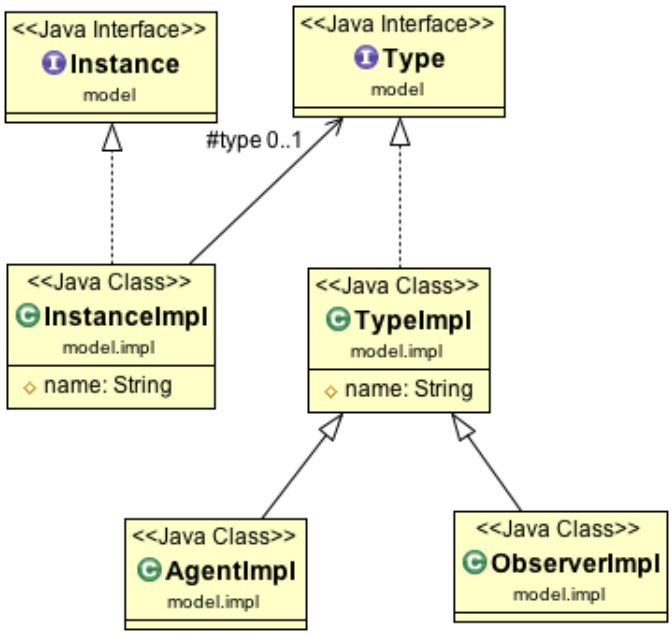


Figure 4.2: Type and Instance Class Diagram

4.1.2 Context Relation

As for the Type and Instance classes, the Context Relation class has its Java interface and its implementation class (see figure 4.3). In this case the attributes in the meta-model that need also to be defined by the user are:

- **Name**: a unique name that identifies the context relation in the diagram.
- **InitialComplexEvent**: the event responsible of the activation of the context relation.
- **EndingComplexEvent**: the event responsible of the deactivation of the context relation.
- **Types**: an EList of the types that represent the parameters of the context relation.

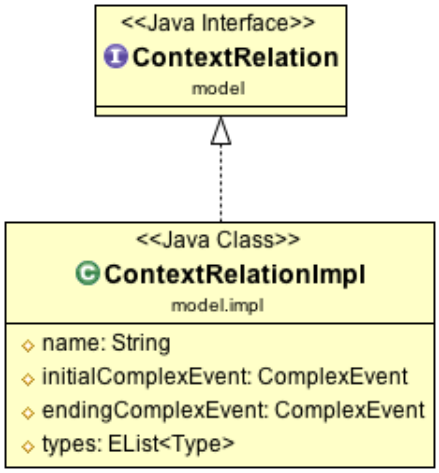


Figure 4.3: Context Relation Class Diagram

4.1.3 Event

As described before, the events can be Primitive or Complex. Primitive events are low-level actions like system calls, while complex events are human actions that originate one or more primitive events. Each event (both primitive and complex) needs some parameters specified by types, like the ones of a context relation. These parameters, in the case of events, are classified into three different categories:

- **Agent**: the agent is who/what performs the action specified by the event.
- **Observer**: the observer is what/who observes the action (camera, NFC reader, etc...) and it can be a parameter of only primitive events.
- **General Type**: all other parameters that have a role in the performance of the action.

Technically, Primitive and Complex Event are two different classes that extend the more general abstract class Event. The abstract class contains an EList of General Type storing the parameters of an event (Primitive or Complex), the Primitive Event class contains a reference to the agent and the observer parameters, while the Complex Event class contains a reference only to the agent parameter.

Furthermore, since each Complex Event class needs at least a Composite Definition that describes its behaviour along the time, there is another EList inside the

Complex Event class that stores all the Composite Definition objects (from now on called Behavioural Description objects).

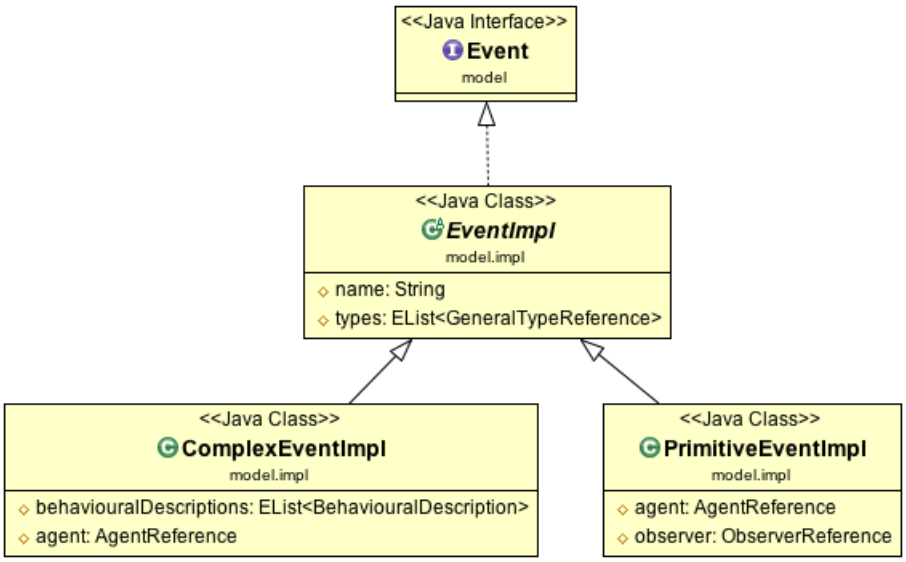


Figure 4.4: Event Class Diagram

4.1.4 Behavioural Description and Hypothesis

Behavioral Description and Hypothesis classes have the same structures because they need the same attributes and references even though they have a different semantic as explained in chapter 2.

Both have a **name** attribute that, for the Behavioural Description class, it needs to be the same as the name of the complex event is describing and, for the Hypothesis, it is a string composed by the letter "h" and a progressive number indicating the i-th hypothesis. For instance: h1, h2, and so on.

Furthermore, they both contain three EList structures to store the three different lists of predicates **Happens**, **Holds At** and **Holds At Between** objects.

The attribute **timeInstants** specifies the time window along which the predicate happens/holds before the complex event.

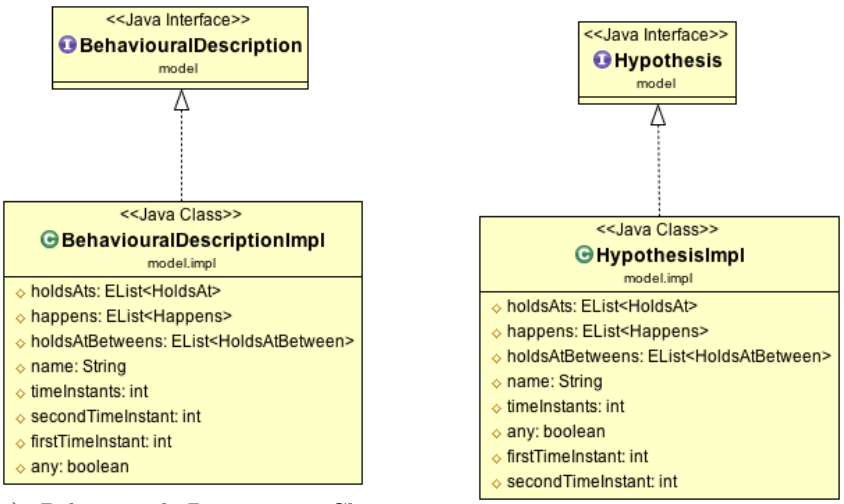
The remaining attributes help specifying the user if he/she wants to, programmatically create other behavioural descriptions with the same list of predicates but with all the possible time combinations. For instance if there are two predicates that happen/hold at two different time instants T1 and T2, then there are two different

possible combinations and the algorithm would create two behavioural descriptions: the first one with the first predicate happening/holding at the time instant T1 and the second predicate happening/holding at the time instant T2, then the second behavioural description with the same time instants but swapped.

In particular the semantic for each of the three remaining attributes is the following:

- **any**: it is a boolean and it activates the algorithm for the creation of all possible behavioural descriptions.
- **firstTimeInstant**: it specifies the starting time instant the "any" algorithm has to consider to create all the possible combinations.
- **secondTimeInstant**: it specifies the ending time instant the "any" algorithm has to consider to create all the possible combinations.

Figures 5a and 5b illustrate the Class Diagram for Behavioural Description and Hypothesis classes.



(a) Behavioural Description Class Diagram

(b) Hypothesis Class Diagram

Figure 4.5: Behavioural Description and Hypothesis Class Diagram

4.1.5 Predicates

The predicates **Happens**, **Holds At**, **Holds At Between** have a correspondent EClass and, consequently a Java Class.

To describe the predicates it is necessary that I first introduce how the dynamic parameters work. The **dynamic parameters** are those ones that are instantiated within a predicate and they are linked to a specific Type class object. It is important to remark here that when an event/context relation is defined, it is associated with specific types that constitute its parameters. In the events, these parameters can be of type agent, observer and general type but in the case of context relations, these parameters can only be of a general type. When a predicate is created, it is necessary to instantiate the dynamic parameters (of the event/context relation) that are linked to a type (agent, observer or general) in order to allow the definition of different instances of parameters of the same type for an event/context relation.

This is the Event-Calculus representation of a Behavioural Description object that explains better what I mean:

```
happens(login(E,C),T,TR):-
        trace(TR),
        emp(E),
        comp(C),
        loc(L),
        emp(E2),
        time(T),
        happens(sys_Login(E,C),T,TR),
        holdsAt(in(E2,L),T,TR),
        holdsAt(isLocatedIn(C,L),T,TR),
        holdsAt(hasPermission(E,C),T,TR).
```

What I want to point out from this example is that the user needs to have the possibility to choose an already used parameter (in another predicate's event/context relation) for the predicate is defining or to define a new one. This is the case of the two first predicates linked to the event "sys_Login" and to the context relation "in": they use two different parameters of the same type Employee: E, E2. That is the reason why I use the concept of dynamic parameters which classes are instantiated at the moment of the creation of a predicate.

The figure 4.6 illustrates the Class Diagram of the predicates and the dynamic parameters.

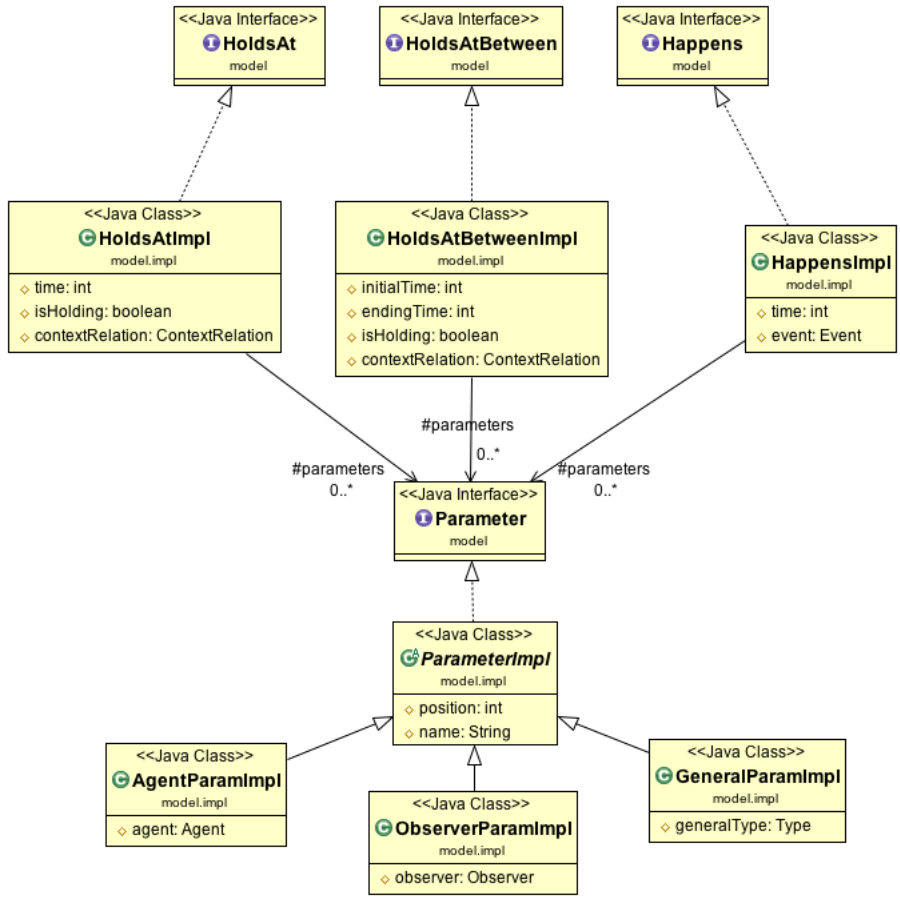


Figure 4.6: Predicates Class Diagram

All the predicates have an EList called "parameters" that store all the dynamic parameters, then there is a reference to the Event/Context Relation object and one/two integers to specify the time instant or time interval when the predicate is true. For the "Holds At" and "Holds At Between" predicates there is also a boolean attribute that specifies whether the predicate is holding at that time instant/interval.

Concerning the **Parameter** class (dynamic parameter), it is extended by the three different kind already discussed and it has a reference to the type Agent/Observer/General. Each dynamic parameter has also a name and an integer indicating the relative position in the event/context relation. For instance in the event "isLocatedIn(C,L)" the parameter C has position 1 and the parameter L has position 2.

4.1.6 Initially

Another part of the meta-model regards the **Initially** class. It has two attributes:

- **contextRelation**: it stores the Context Relation object that needs to be true at the initial state.
- **instances**: it is an EList of instances that are considered as parameters of the context relation.

Figure 4.7 illustrates the Initially Class Diagram.

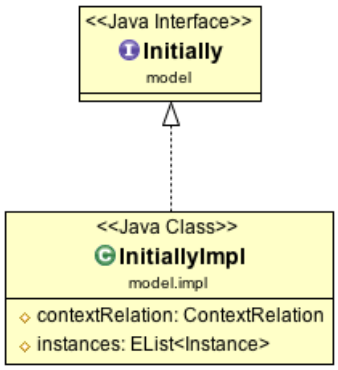


Figure 4.7: Initially Class Diagram

4.2 The Graphical Designer

The graphical designer of the entire software is essentially composed by 6 Eclipse extensions, all of them structured as described in paragraph 3.4.4.

This division into 6 extensions comes from the need of the creation of different diagrams for describing the entire environment. Each graphical designer (extension) models the definition of a diagram that describes only a part of the environment. The division that I propose is the following:

- ***.typeInstance.diagram:** graphical designer for the definition of types and instances.
- ***.contextRelation.diagram:** graphical designer for the definition of context relations.

- ***.event.diagram:** graphical designer for the definition of events.

- ***.behavDesc.diagram:** graphical designer for the definition of behavioural descriptions and its predicates.

- ***.hypothesis.diagram:** graphical designer for the definition of the incident hypotheses and its predicates.

- ***.initial.diagram:** graphical designer for the definition of the initial states.

Each of them collaborates to the total description of the environment. The final user, in order to describe the whole environment needs to create six different diagrams and thanks to GMF, it is possible to use elements in a diagram that were defined previously in another one. This can be the case in the definition of an event when it is mandatory defining some parameters that need to be associated to a Type object that has been defined previously in the Type-Instance diagram.

All the diagrams come with a workspace made up of the **diagram area**, a **palette area** and a **property setting area** at the bottom.

The final user should also start creating the diagrams in the same order I listed the GMF extensions above because the definition of each diagram requires some knowledge that only a diagram up on the list can provide.

GMF, as well as EMF, is a command-stack based platform that organises in a FIFO policy queue all the creation/editing/elimination operations of elements inside the diagram. Thanks to the stack it is possible to handle various situations such as undo and redo operations. GMF uses the abstract class of GEF library "org.eclipse.gef.commands.Command".

Several times, I adopted a programmatically creation of nodes into the diagram to facilitate the user experience avoiding a manual operation. To do so, it was important understanding how the Command class works within GEF and the result is the method shown below.

Listing 4.1: Command for the creation of a node

```
1  public Command createAndExecuteShapeRequestCommand(IElementType type, EditPart
       parent) {
2          CreateViewRequest actionRequest = CreateViewRequestFactory.
               getCreateShapeRequest(type,
3                   PreferencesHint.USE_DEFAULTS);
4          org.eclipse.gef.commands.Command command = parent.getCommand(
               actionRequest);
5          command.execute();
6          return command;
7      }
```

The method takes as input an EditPart object and the element type that refers to an entity of the meta-model. It then creates a request (2) that is then used to create a command (4) before the execution (5).

Now I describe all the listed above graphical designer GMF extensions in detail, one for each paragraph.

4.2.1 Type and Instance GMF extension

The first graphical designer is kind of basic and it represents Type and Instance objects in a concise way. From the right it is possible to drag-and-drop all the elements to the diagram: Agent Type, Observer Type, General Type and Instance. The picture 4.8 illustrates the case of three computers (m1, m2, m3) linked to their correspondent type «comp» and an employee which name is Alice.

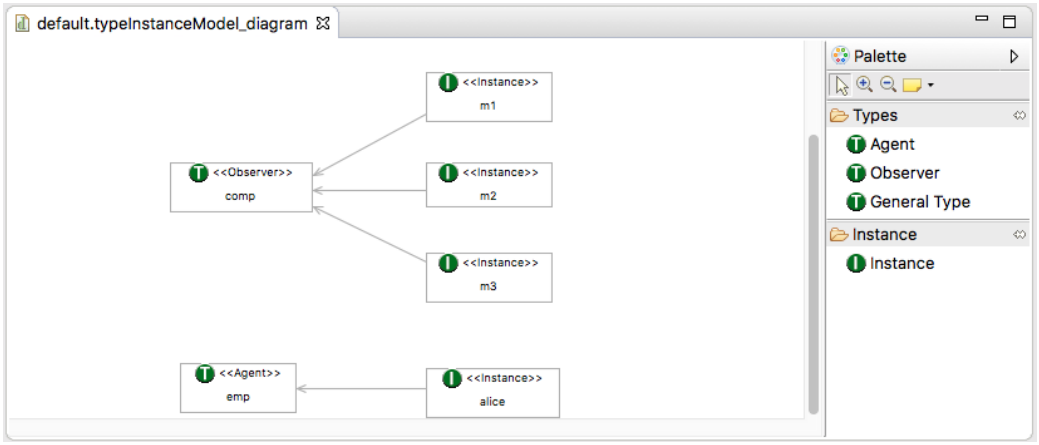


Figure 4.8: Type and Instance Diagram and Palette area

4.2.2 Context Relation GMF extension

The second diagram to be defined is the one related to the Context Relation. Each of them is represented as a unique node with the name attribute and the list of types (parameters of the C.R.) shown. The only element to be dragged-and-dropped from the palette is the Context Relation one and the attributes are modifiable from the Property area after having loaded the resources of the Type-Instance diagram.

As I said before, most of the diagram in order to be created, they need to load the elements from other diagrams. In this case the XMI file belonging to the Type-Instance diagram need to be loaded because each Context Relation object needs a list of parameters that are Type objects. It is possible to accomplish it within an EMF feature that is activated from the context menu by right clicking on the diagram. The example in figure 4.9 shows two nodes representing the two Context Relation objects: “in”, “hasBadge”.

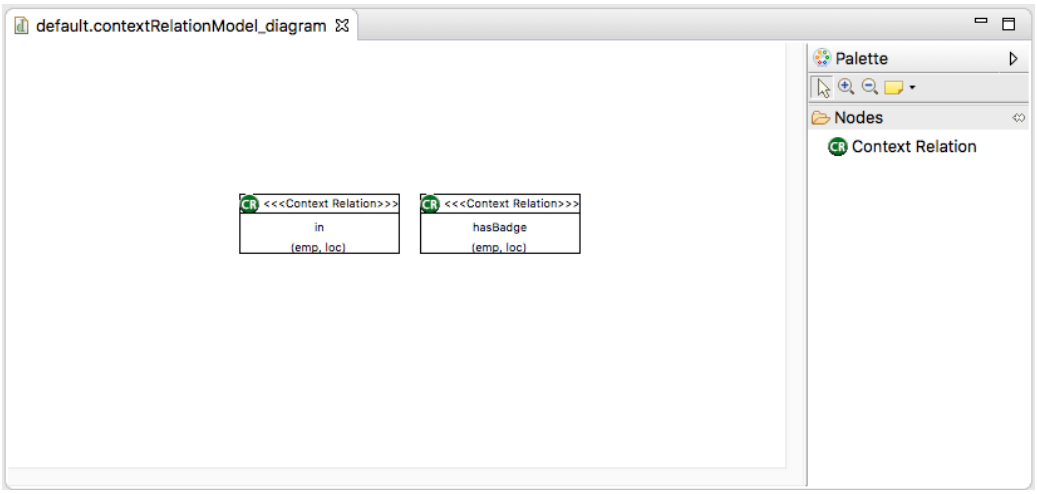


Figure 4.9: Context relation Diagram and Palette area

In the property area (figure 4.10) it is necessary to set a name, a list of parameters and the activating/deactivating Complex Event object which determines when the context relation is true. It is clear that it is possible to select the initial and ending complex event only after having crated the events in the appropriate diagram.

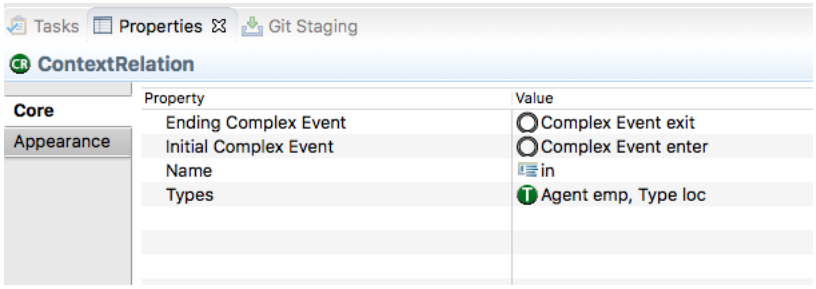


Figure 4.10: Context Relation Property area

4.2.3 Event GMF extension

The events use more complex graphical elements. There is a central node for the definition of a Primitive Event object or a Complex Event object and a link between an event node to each of its parameters.

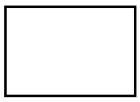
Each parameter, that can be of type Agent, Observer or General has its own graphical representation as the sub-figures 4.11 show.

(a) Primitive Event Node

(b) Complex Event Node

(c) Agent parameter Node

(d) Observer parameter Node



(e) General Type parameter Node

Figure 4.11: Event Graphical Elements

The standard way to instantiate Event objects expects the user to choose first the Primitive Event or Complex Event node and then to drag-and-drop the parameters from the palette and load the Type-Instance external resource by right-clicking again on the diagram. Then the third step is to link the event node to its parameter nodes and once it is done, the last thing to set is the Type object for each parameter in the

Property area. Putting these elements all together, the result is showed in the figure 4.12 which illustrates the definition of 5 Primitive Event objects and 1 Complex Event object.

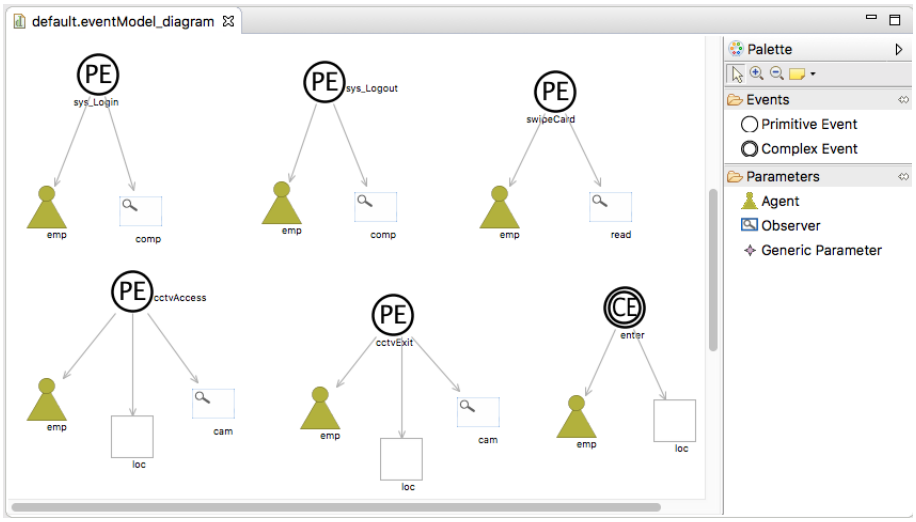


Figure 4.12: Event Diagram and Palette area

However, a better improvement has been done. When the user selects and adds a Primitive Event node to the diagram, the two mandatory parameter nodes (Agent and Observer) automatically appear with the two links connecting the parameters to the event. The user, at this point only has to add other General Type parameters if necessary and select the appropriate Type object from the property area.

The same thing happens if the user drags-and-drops a Complex Event node with the exception that only the Agent node automatically appears connected to the event node because it is the only mandatory one and a complex event does not accept any observer.

The following is the code snippet related to the automatic creation of the Agent node after the user drags-and-drops a Primitive Event node. The same holds for the creation of an observer.

Listing 4.2: Agent creation source code

```
1 private void agentCreation() {
2     // Creating AgentReference
3     Command cmd = editor.createAndExecuteShapeRequestCommand(
```

```
4               event.model.diagram.providers.ModelElementTypes.
                    AgentReference_2016, editor.getDiagramEditPart());
5       editor.getDiagramEditPart().getDiagramEditDomain().getDiagramCommandStack
            ();
6       // Creating and executing the command to set the properties
7       Collection<?> results = DiagramCommandStack.getReturnValues(cmd);
8       Iterator<?> iter = results.iterator();
9       AgentReference newAgent = new AgentReferenceImpl();
10      while (iter.hasNext()) {
11          Object obj = iter.next();
12          if (obj instanceof CreateElementRequestAdapter) {
13              CreateElementRequestAdapter cra = (CreateElementRequestAdapter)
                    obj;
14              newAgent = (AgentReferenceImpl) cra.resolve();
15              // Setting the AgentReference EReference of the Primitive Event
16              SetRequest setRequestAgent = new SetRequest(editor.
                    getEditingDomain(), view.getElement(),
17                      ModelPackage.eINSTANCE.getPrimitiveEvent_Agent(), newAgent
                            );
18              SetValueCommand agentOperation = new SetValueCommand(
                    setRequestAgent);
19              editor.getDiagramEditDomain().getDiagramCommandStack().execute(new
                     ICommandProxy(agentOperation));
20          }
21      }
22      // Refresh the diagram (it allows to render the connection between the
            Event and the Parameter)
23      Display.getDefault().asyncExec(new Runnable() {
24          public void run() {
25              editor.getDiagramEditPart().addNotify();
26          }
27      });
28  }
```

The method calls “createAndExecuteShapeRequestCommand()” that creates the necessary command to perform the action. The command is then returned and it is processed (7) to find out the reference to the new object. An instance of the class “AgentReference” is found and it is stored as a reference in the Primitive Event attribute “agent” within the execution of a new command (19).

Then, the diagram is refreshed and it automatically creates the link between the

two nodes.

4.2.4 Behavioural Description and Hypothesis GMF extension

The user experience for the creation of a Behavioural Description/Hypothesis is totally different. The Behavioural Description/Hypothesis graphical element aims to describe the predicates distributed in a timeline, depending on when they happen/hold.

To create a new element in the diagram the first step to perform is taking the unique element from the palette and move it into the diagram. Then it is necessary to give a name to the behavioural description that must be the same as the complex event that is describing. The other important step is setting the time window of the object. These two operations must be performed in the property area at the bottom of the diagram.

When these two preliminary operations are done, it is necessary to list all the predicates and this is done by double-clicking on the object in the diagram. A guided procedure will start that allows the user to select the predicate, the event/context relation associated to it (from the ones he/she has previously defined) and the dynamic parameters: the user at this point can create new parameters or he/she can choose from a list of already used ones if it is not the first predicate is creating.

The picture 4.13 refers to a Behavioural Description object that describes the Complex Event “enter”. There are 2 “Happens” predicates (swipeCard, cctvAccess), 3 “Holds At” predicates (hasBadge, isAccessControlledBy, isMonitoredBy), 1 “Holds At Between” predicate (in) distributed in 2 different time instants. The semantic of this is that the Complex Event “enter” can happen if all those predicates in the two time instants pictured in the image are true considering the event “enter” happening at the second time instant. The predicate “Holds At Between” associated to the context relation “in” must be false in both time instants in order for the event “enter” to happen. In general, when a context relation associated to the predicates “Holds At” and “Holds At Between” has be false, it appears with a red square/rectangle in the image.

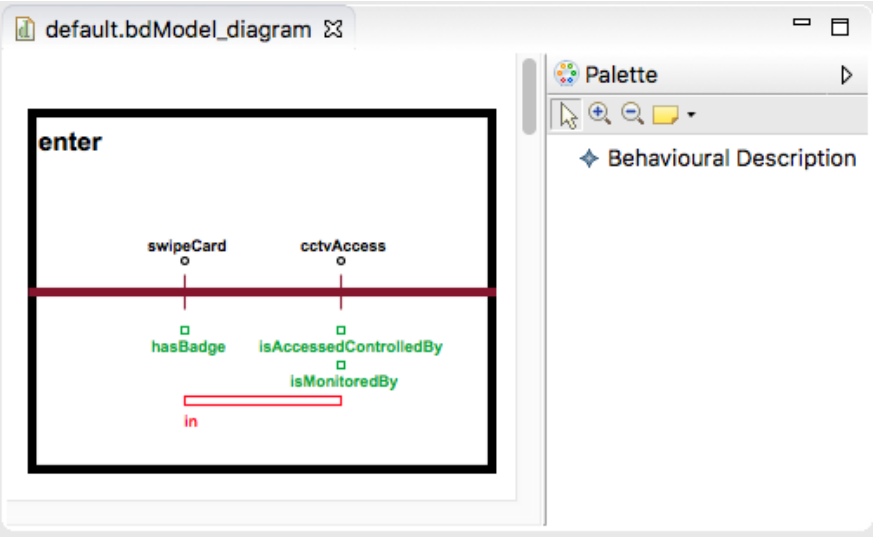


Figure 4.13: Behavioural Description Diagram and Palette area

Creating a timeline like the one on the picture is not possible using the standard GMF elements and that is why I needed to customise the figure that in this case is described by a different class in a different package where all the custom figures are stored.

The class (BehaviouralDescriptionFigure.java/HypothesisFigure.java) uses the library org.eclipse.draw2d and in particular it extends the class org.eclipse.draw2d.Shape that has the tools that helped me to obtain that result.

The creation of predicates and dynamic parameters is completely done programmatically and it involves the use of commands and requests of GMF. The code snippet below creates an "Happens" predicate and it handles the graphical windows that guide the user choosing the predicate and dynamic parameters and all the process involving the commands.

Listing 4.3: Happens predicate creation source code

```
1 private Happens happensSelected() {
2     try {
3         // Parsing event file
4         LoadEvents loadEvents = new LoadEvents(editFilesPath + "/default.
              eventModel");
5         Display display = PlatformUI.getWorkbench().getDisplay();
6         Shell shell = new Shell(display);
```

```
7           Event ev = null;
8           // Creating second dialog to show the list of the available events
9           CustomListDialog showEventsDialog = new CustomListDialog(null, new
                LabelProvider());
10          String[] eventsNameArray = new String[loadEvents.getEnvironment().
                getEvents().size()];
11          for (int i = 0; i < loadEvents.getEnvironment().getEvents().size(); i
                ++) {
12              eventsNameArray[i] = loadEvents.getEnvironment().getEvents().get(i
                    ).getName();
13          }
14          showEventsDialog.setElements(eventsNameArray);
15          showEventsDialog.setMultipleSelection(false);
16          showEventsDialog.setTitle("Select an event");
17          // User pressed cancel
18          if (showEventsDialog.open() != Window.OK) {
19              return null;
20          }
21          String eventSelected = (String) showEventsDialog.getResult()[0];
22          int index = 0;
23          do {
24              if (eventSelected.equals(loadEvents.getEnvironment().getEvents().
                    get(index).getName())) {
25                  ev = loadEvents.getEnvironment().getEvents().get(index);
26              }
27              index++;
28          } while ((index < loadEvents.getEnvironment().getEvents().size()) &&
                ev == null);
29          // Creating third dialog where the user inputs the time instant where
                to place the event
30          int timeSelection = createSingleTimeInstantsDialog();
31          if (timeSelection == -1)
32              return null;
33          // Creating Happens
34          Command cmd = editor.createAndExecuteShapeRequestCommand(
35              behavDesc.model.diagram.providers.ModelElementTypes.Happens_2002,
                    editor.getDiagramEditPart());
36          editor.getDiagramEditPart().getDiagramEditDomain().
                getDiagramCommandStack();
37          // Creating and executing the command to set the properties
38          Collection<?> results = DiagramCommandStack.getReturnValues(cmd);
39          Iterator<?> iter = results.iterator();
40          Happens newHappens = new HappensImpl();
```

```
41          while (iter.hasNext()) {
42              Object obj = iter.next();
43              if (obj instanceof CreateElementRequestAdapter) {
44                  CreateElementRequestAdapter cra = (CreateElementRequestAdapter
                        ) obj;
45                  newHappens = (HappensImpl) cra.resolve();
46                  // Setting the happens EReference of the Behavioural
                        Description
47                  SetRequest setRequestHappens = new SetRequest(editor.
                        getEditingDomain(), view.getElement(),
48                          ModelPackage.eINSTANCE.
                                getBehaviouralDescription_Happens(), newHappens);
49                  SetValueCommand behavDescOperation = new SetValueCommand(
                        setRequestHappens);
50                  editor.getDiagramEditDomain().getDiagramCommandStack()
51                          .execute(new ICommandProxy(behavDescOperation));
52                  // Setting the property of Happens
53                  SetRequest setRequestTimeInstant = new SetRequest(editor.
                        getEditingDomain(), newHappens,
54                          ModelPackage.eINSTANCE.getHappens_Time(),
                                timeSelection);
55                  SetValueCommand propertyOperation = new SetValueCommand(
                        setRequestTimeInstant);
56                  editor.getDiagramEditDomain().getDiagramCommandStack()
57                      .execute(new ICommandProxy(propertyOperation));
58              }
59          }
60          // Looking for the event the user decided to associate with the new '
                happens' predicate and setting the property
61          for (int i = 0; i < loadEvents.getEnvironment().getEvents().size(); i
                ++) {
62              if (eventSelected.equals(loadEvents.getEnvironment().getEvents().
                    get(i).getName())) {
63                  SetRequest setRequestEvent = new SetRequest(editor.
                        getEditingDomain(), newHappens,
64                          ModelPackage.eINSTANCE.getHappens_Event(), loadEvents.
                                getEnvironment().getEvents().get(i));
65                  SetValueCommand operation = new SetValueCommand(
                        setRequestEvent);
66                  editor.getDiagramEditDomain().getDiagramCommandStack().execute
                        (new ICommandProxy(operation));
67              }
68          }
```

```
69          DynamicParametersDialog dpd = new DynamicParametersDialog(null, bd,
                newHappens, editor, editFilesPath,
70                  diagramFilePath);
71          if (dpd.open() != Window.OK) {
72              DestroyElementRequest destroyRequest = new DestroyElementRequest(
                    editor.getEditingDomain(), newHappens,
73                      false);
74              DestroyElementCommand destroy = new DestroyElementCommand(
                    destroyRequest);
75              editor.getDiagramEditDomain().getDiagramCommandStack().execute(new
                     ICommandProxy(destroy));
76              return null;
77          }
78          return newHappens;
79          } catch (IOException e) {
80              e.printStackTrace();
81          }
82          return null;
83  }
```

From the line 4 to the line 32 I use the library "org.eclipse.ui.dialogs" from Eclipse API to let the user decides the predicate and the choice of the dynamic parameters with some customised dialog windows. Then the usual method "createAndExecute-ShapeRequestCommand()" generates and executes the command to create a new instance of the predicate; finally the rest of the code sets the attributes of the new predicate. At the end of this process a refresh on the diagram is performed and the new element is rendered in the behavioural description node.

4.2.5 Initially GMF extension

The Initially element is described with a simple rectangle containing the name of the entity at the top and a list of instances below. Figure 4.14 shows 2 Initially nodes.

The user first needs to give a name writing it in the property area and he/she has to choose the Context Relation object again from the property area after having loaded the external resource corresponding to the Context Relation diagram. When it is done, the last step is choosing the instances that have the same type of the

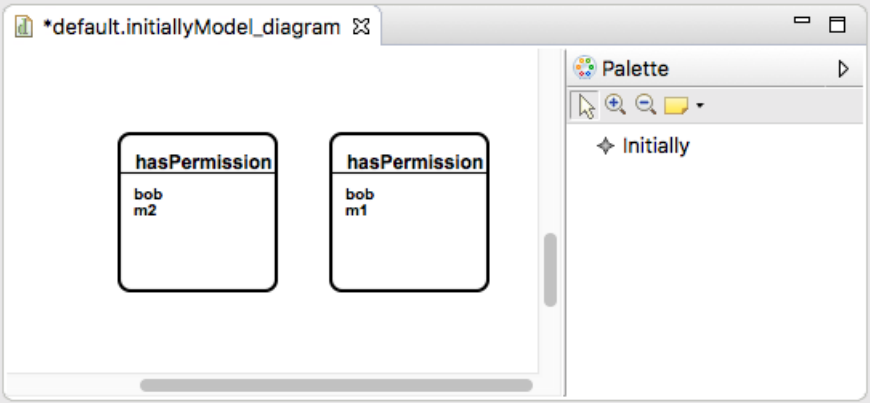


Figure 4.14: Initially Diagram and Palette area

parameters of the Context Relation object selected before. This can be done by double-clicking on the node in the diagram: again here an Eclipse dialog will show up allowing the user to select only the instances that have the correct type.

4.3 Encoding extension

The encoding plugin is the last to be described. Its job is encoding the information from the XMI files that describe an instance of the meta-model created thanks to the graphical designer. As I also specified in the first chapter, kEEPER needs an Event-Calculus description of the environment and the incident hypotheses so the objective of this extension is providing a .txt file for the description of the environment in E.C. and one .txt file per each incident hypothesis again written following the standard of the Event-Calculus logical language.

It is possible to run this Eclipse plugin from the context menu by right-clicking on the folder containing the diagram files in the Project Explorer of Eclipse and selecting the "Gen Encoding" option. The algorithm will first validate some important aspects of the instance the user created checking whether all the elements in the diagrams have been set correctly and warning the user if not. If the validation is successful the algorithm will proceed loading and parsing the files creating the new .txt files ready to be taken as input from kEEPER that afterwards will generate the specifications.

The figure 4.15 shows the context menu where to run the encoding algorithm from.

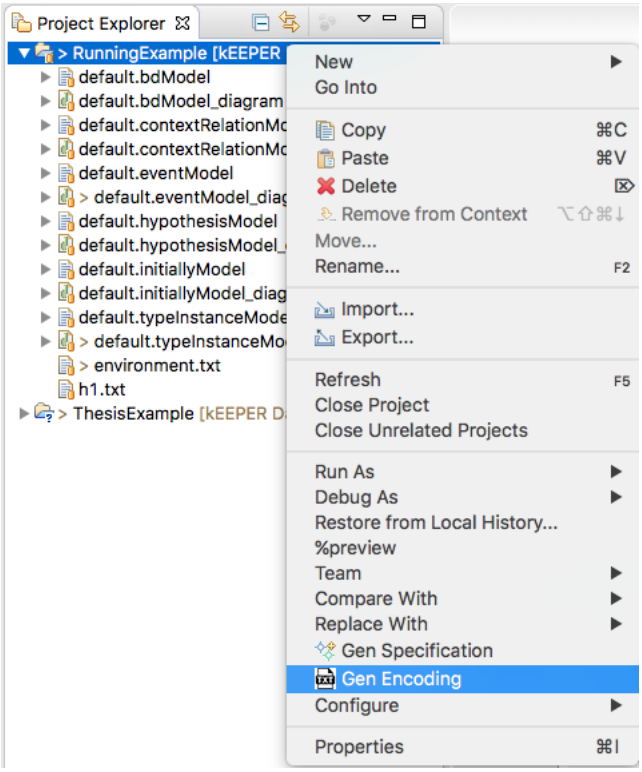


Figure 4.15: Gen Encoding context menu function

Chapter 5

Evaluation

I divide this chapter into two main sections. In the first part I describe the three case studies used to evaluate the graphical designer. A part of the first example has been used in this thesis to motivate the need of a systematic approach that kEEPER provides. The other two scenarios, as well as the motivating example, are inspired by two existing digital forensic corpora. The second part is a discussion about what I have done and which are the limitations of my work.

5.1 Case studies

The tool kEEPER has been tested with three different case studies to evaluate its effectiveness and in particular to understand whether the synthesised preservation specification prescribes to preserve the **relevant events** and the **minimal amount of events**. Relevance and minimality are the two metrics of evaluation of the tool: the former is assessed verifying whether the specification generated prescribes to preserve events that were relevant to satisfy the speculative hypotheses while the latter is assessed verifying that the approach used prescribes to preserve fewer events than the ones that can be inferred directly from the data-set that the investigator would use if the tool did not exist. The three case studies describe different possible environments where incidents can occur and I used the three of them to evaluate my graphical designer understanding if it was possible to completely describe the envir-

onment and the incident hypotheses. It turns out that the graphical designer provides all the necessary instruments to easily represent the environment and the incident hypotheses correctly. In particular, I represented graphically the environment and the hypotheses of the exfiltration scenario and the harassment scenario in 2.5 and 2 hours respectively against the 3.5 and 2 working days required. Screenshots of the diagrams are provided for the first two examples. The Event-Calculus translation of the environment and the incident hypotheses of the first two scenarios can be found in the appendixes A and B.

5.1.1 Motivating Example

This example describes an incident of an environment within an enterprise and involves a total of 7 types and 10 instances.

The types and the instances of the environment are depicted in the picture 5.1 that shows their graphical representation. They are two employees *bob* and *alice*, a location *r01*, a camera *cctv1*, a reader *nfc1*, a file *doc*, a storage device *usb1* and three computers *m1*, *m2*, *m3*.

The context relations are 9 in total and they specify particular status of the environment such as the location of a computer or which device is monitoring a particular room. All the context relations are shown in figure 5.2.

The primitive events are 9 and they are shown in figure 5.3 while the complex events are 7 and they are shown in figure 5.4. The primitive events describe some atomic actions (system calls) from mounting a device in a computer or copying a sensitive file into a device to the login or logout of an employee. The complex events describe the more complex actions performed by an employee like entering or going out of a location.

The events pre-conditions are described by the 8 behavioural descriptions shown in figure 5.5. In particular, the events “enter” and “exit” need 3 behavioural descriptions each because of the different pre-conditions expressed by the predicates. For example the first two behavioural descriptions in the figure 5.5 have the same predicates but holding in different time instants.

Finally the description of the environment is concluded with the definition of its initial state. The initial state of this example is expressed by 10 Initially objects that are represented in the diagram as figure 5.6 shows. As it is possible to infer from the picture, *bob* has permission to access the computers *m1* and *m2* and *alice* has permission to access the computer *m3* and *m1*. Both *alice* and *bob* have the badge to enter the room *r01* which access is monitored by the camera *cctv1* and controlled by the nfc device *nfc1*. The sensitive document *doc* is stored in the computer *m1* that is located in the room *r01*.

The unique incident hypothesis of this scenario is pictured in figure 5.7 that describes the copy of a sensitive document while an external device is mounted in a computer.

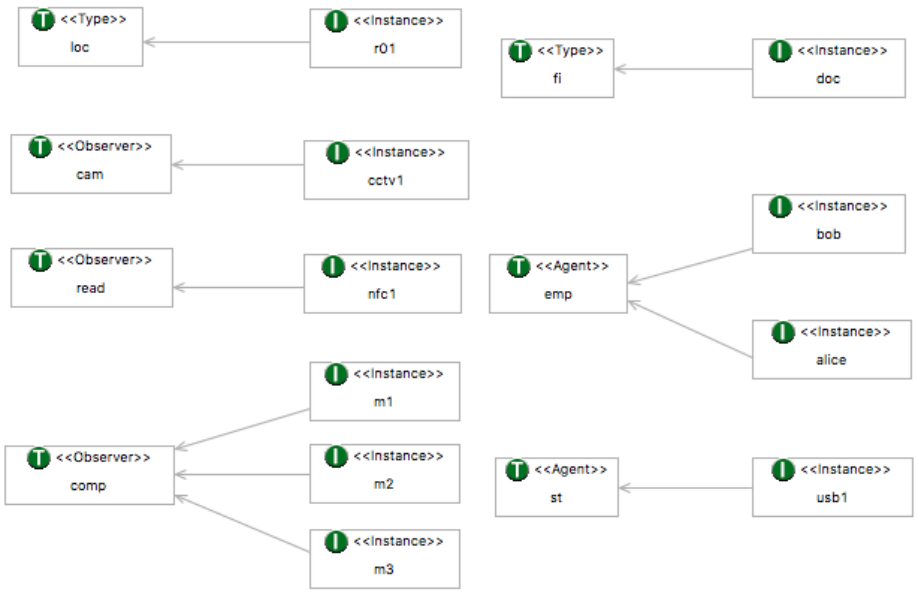


Figure 5.1: Type and Instance Diagram for the Motivating Example

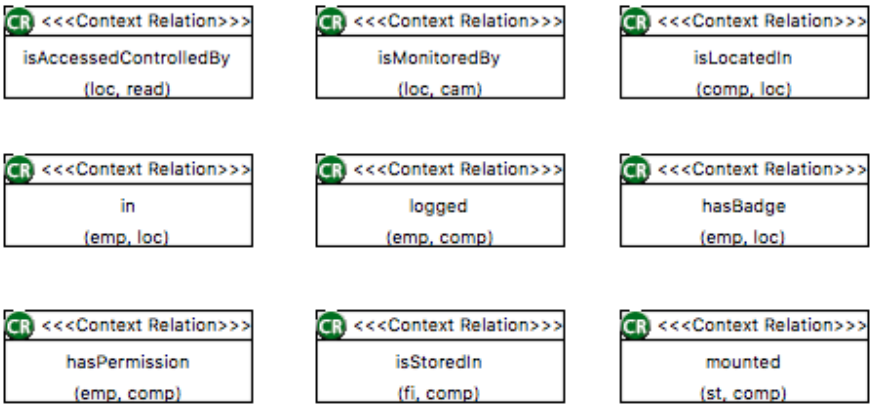


Figure 5.2: Context Relation Diagram for the Motivating Example

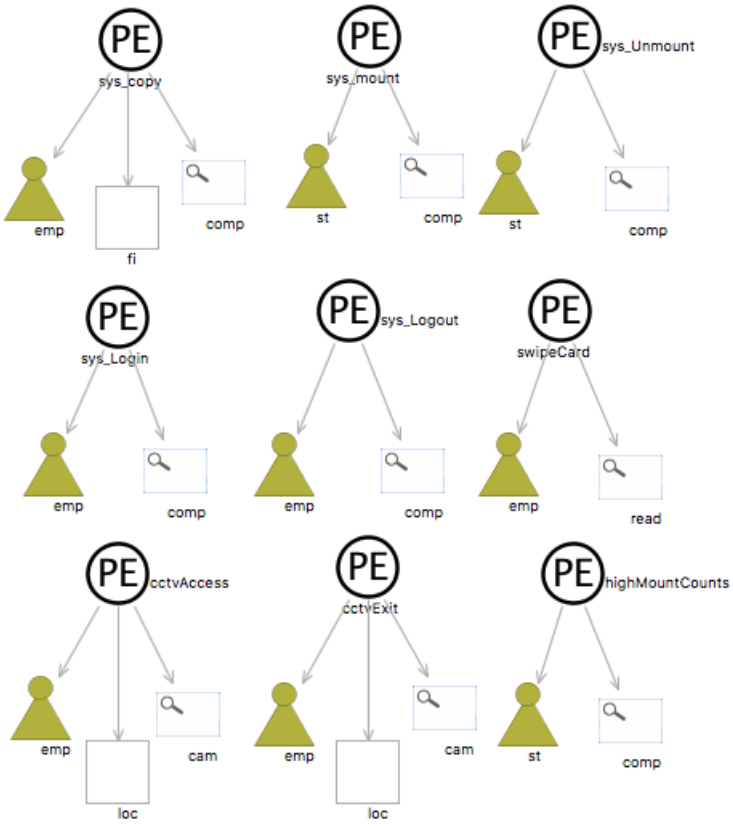


Figure 5.3: Primitive Event Diagram for the Motivating Example

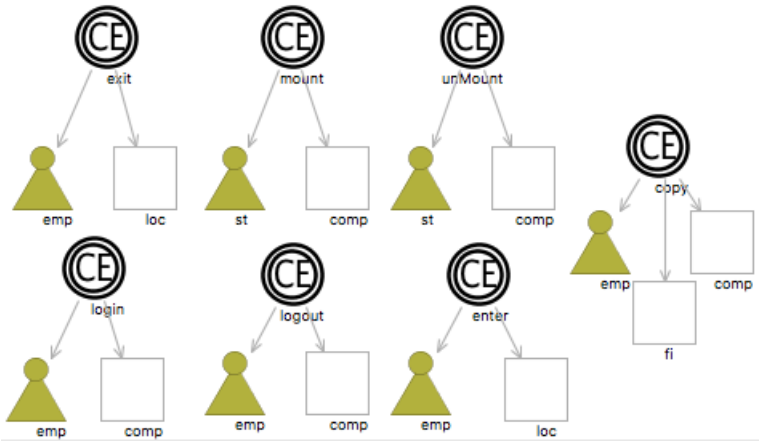


Figure 5.4: Complex Event Diagram for the Motivating Example

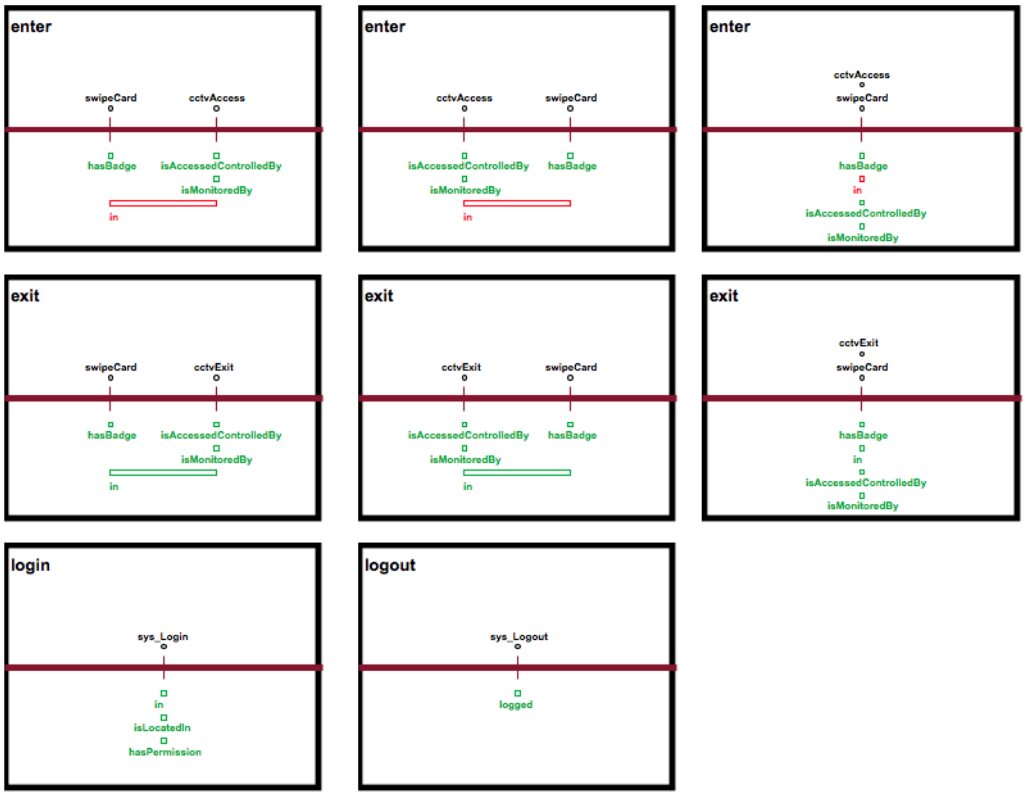


Figure 5.5: Behavioural Description Diagram for the Motivating Example

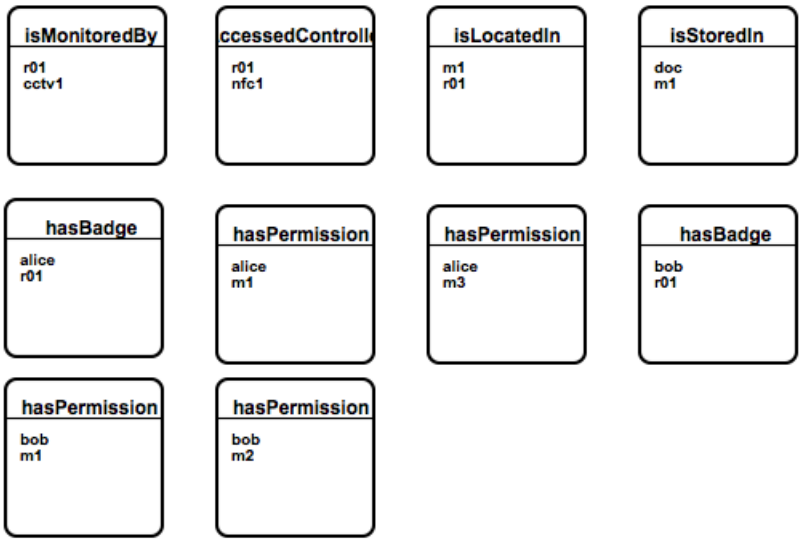


Figure 5.6: Initially Diagram for the Motivating Example

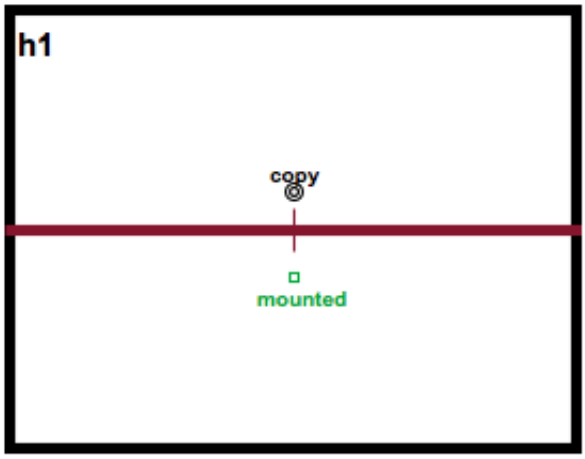


Figure 5.7: Hypothesis Diagram for the Motivating Example

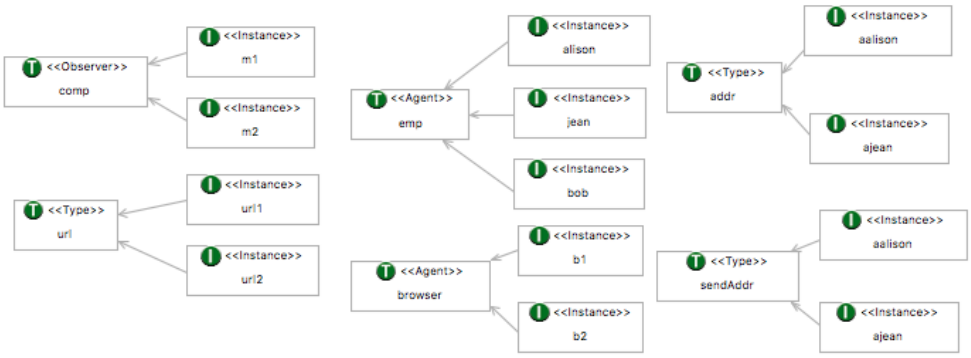
5.1.2 Exfiltration Scenario

This scenario involves the exfiltration of corporate documents from the laptop of a senior executive. The company (M57.Biz) is a small start-up organisation and it was the victim of the leak of sensitive information. These information regarded names and salaries of the employees and they have been posted in the "comments" section of the website of a competitor.

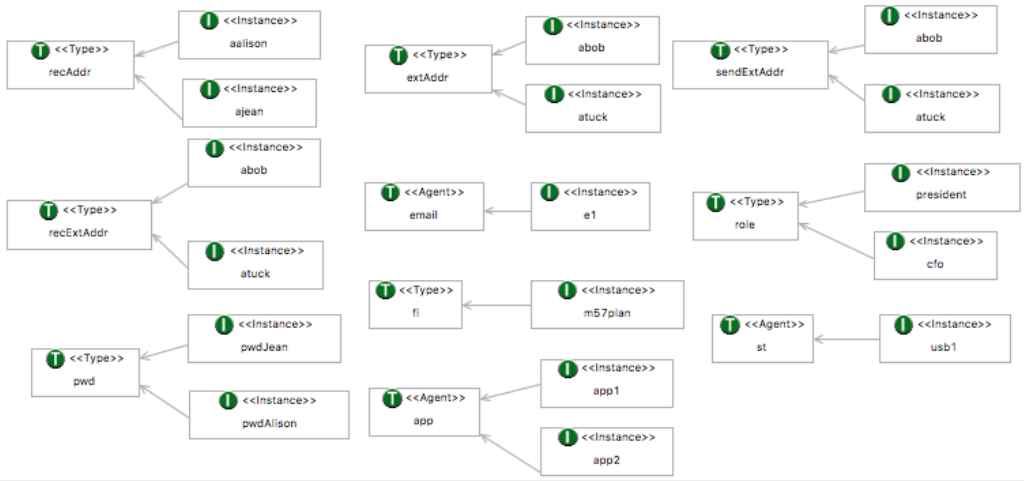
Types and instances describe computers, applications and the various email addresses that have a role in this scenario. They are shown in the sub-figures 5.8. The context relations aim to describe who can access to the computers, who is in possess of email addresses and which applications are classified as malware. They are 13 in total and they are shown in figure 5.9. The description of the scenario models the sending and receiving of emails from/to internal and external email addresses where internal addresses are to be intended of being corporate addresses and external are email accounts external to the organisation. These actions are described by the events together with the system calls to the operating system that handle the mounting of external devices, copying of files and installation of applications. The complex events describe these actions as performed by the employees of the organisation while the primitive events describe the accesses to the hard drive to send/receive the emails. Primitive and complex events together are 32 in total and they are shown in the sub-figures 5.10. The behavioural descriptions specify the pre-conditions for the installation of applications, mounting of a device, login and logout of employees and what need to happen before receiving or sending an email. They are 20 in total and the correspondent diagram is illustrated in figure 5.11. The initial state of the environment models the permissions of the employees to access the computers. For instance, *alison* can access the computer *m2* using her password *pwdAlison*. Furthermore, the initial state describes that the application *app1* is a malware and that the sensitive document is *m57plan* that is stored in the computer *m1*. There are 11 total Initially nodes in the diagram that is shown in figure 5.12.

In conclusion, 6 incident hypotheses regarding the sending and receiving of emails of the sensitive document *m57plan* and the copying of the same file are described in

figure 5.13.



(a) Type and Instance Diagram for the Exfiltration Scenario



(b) Type and Instance Diagram for the Exfiltration Scenario

Figure 5.8: Type and Instance Diagram for the Exfiltration Scenario

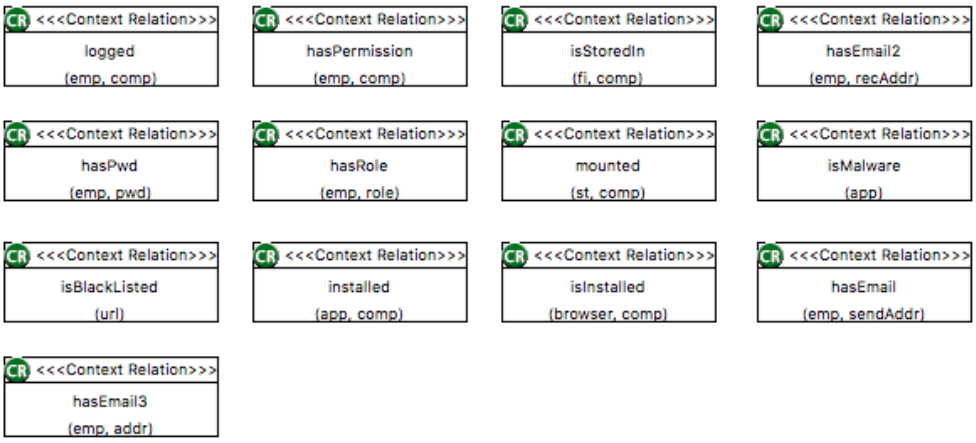
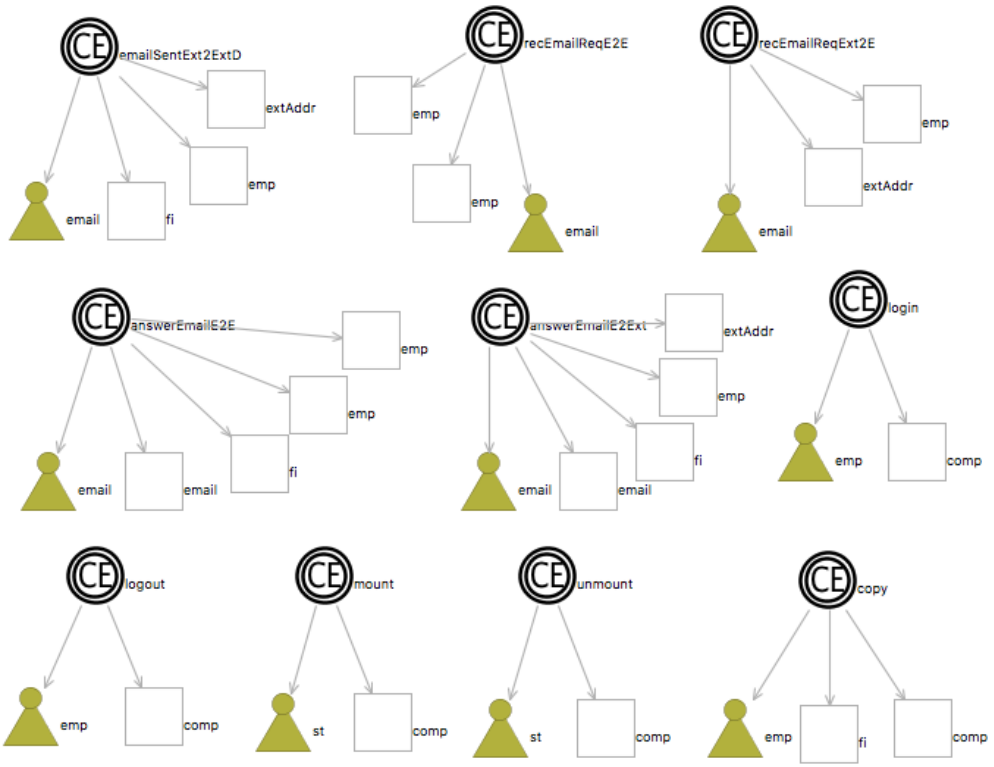
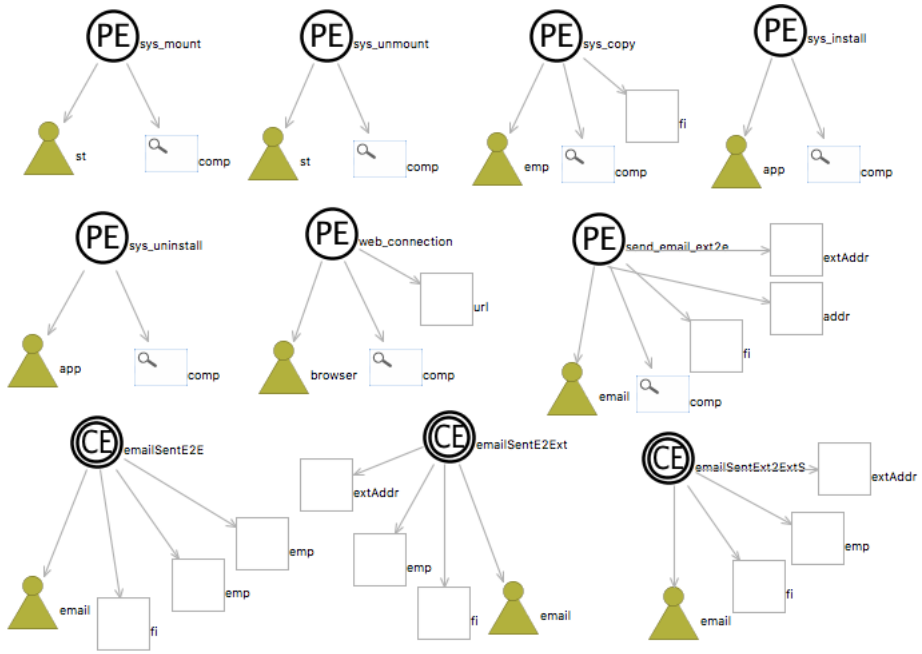


Figure 5.9: Context Relation Diagram for the Exfiltration Scenario

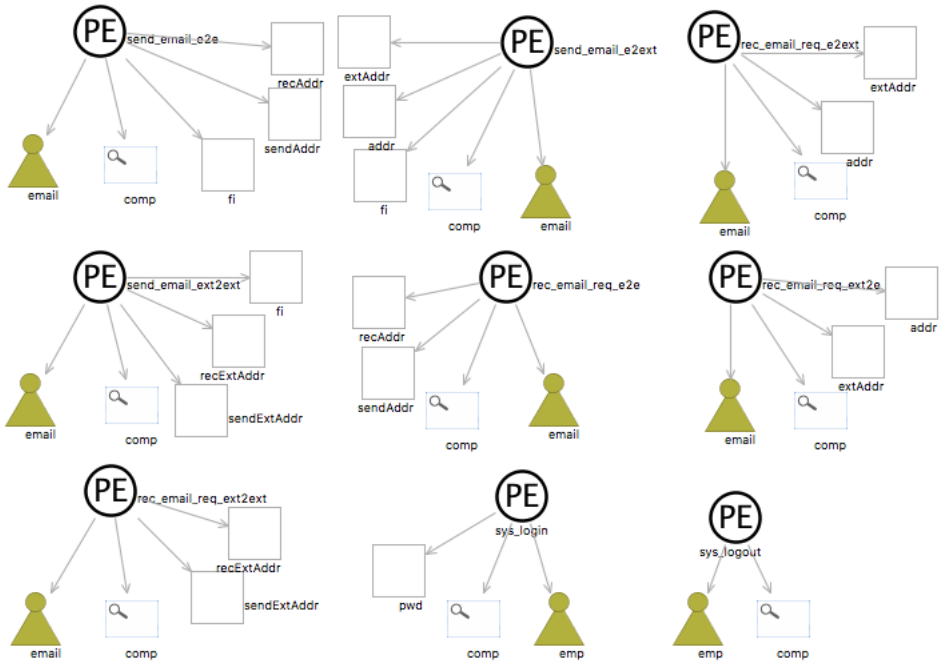


(a) Event Diagram for the Exfiltration Scenario

Figure 5.10: Event Diagram for the Exfiltration Scenario

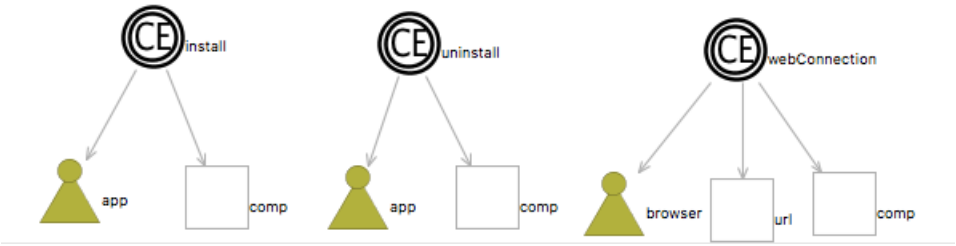


(b) Event Diagram for the Exfiltration Scenario



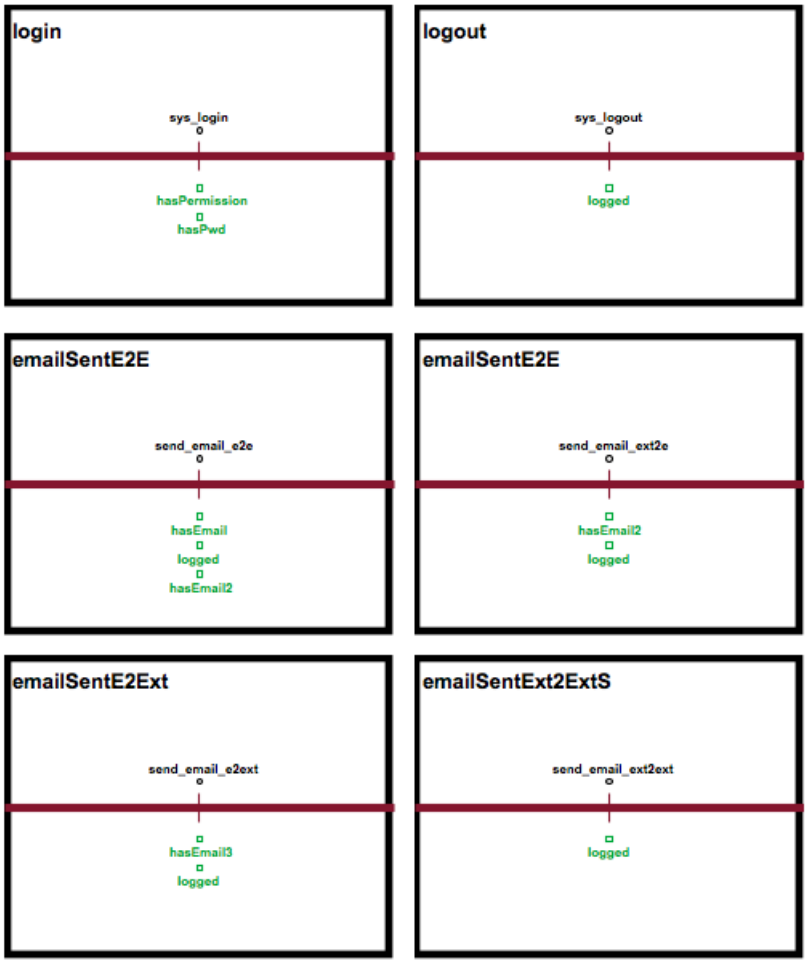
(c) Event Diagram for the Exfiltration Scenario

Figure 5.10: Event Diagram for the Exfiltration Scenario

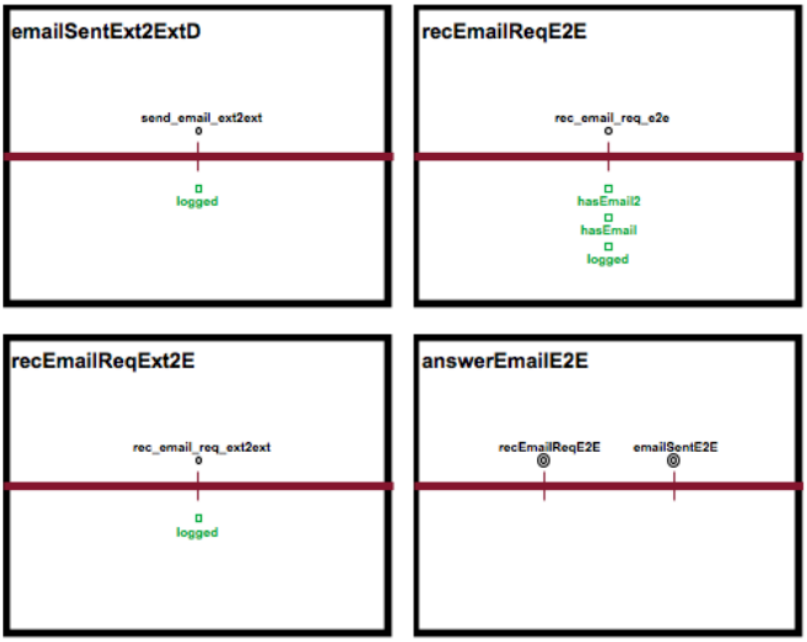


(d) Event Diagram for the Exfiltration Scenario

Figure 5.10: Event Diagram for the Exfiltration Scenario

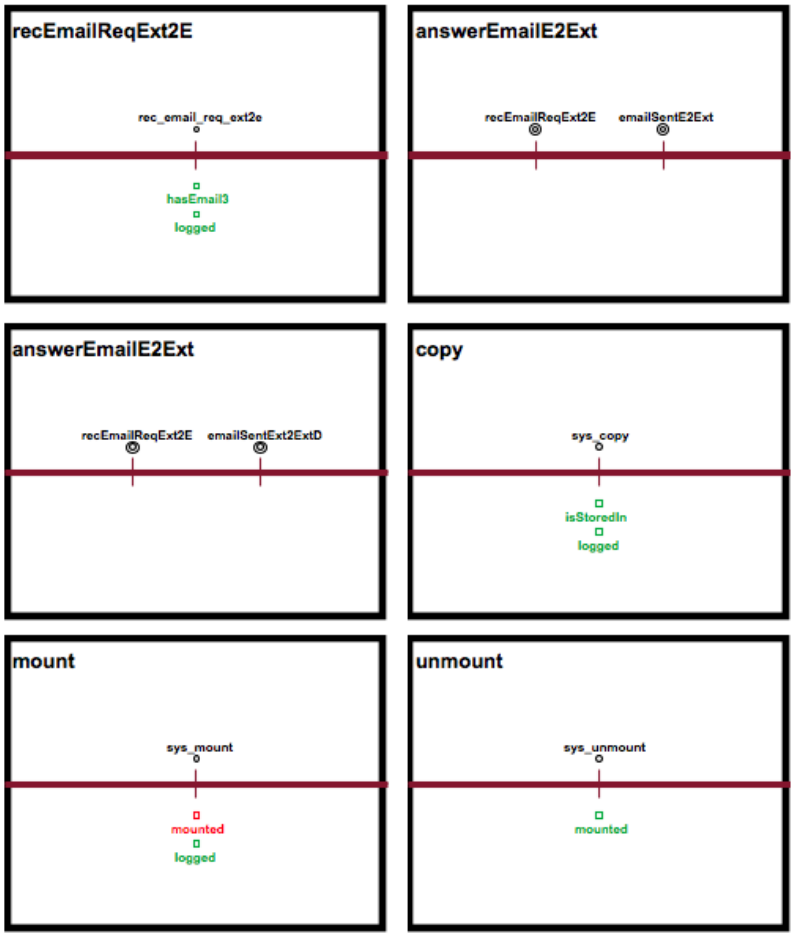


(a) Behavioural Description Diagram for the Exfiltration Scenario

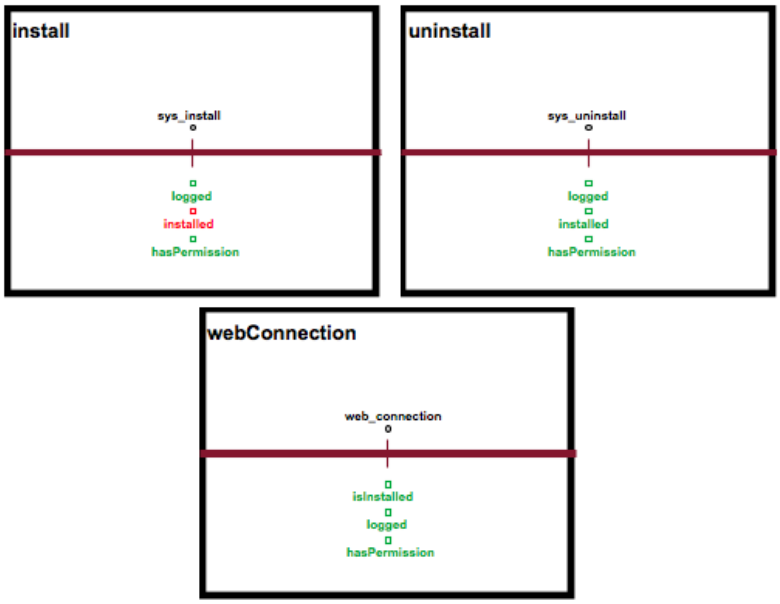


(b) Behavioural Description Diagram for the Exfiltration Scenario

Figure 5.11: Behavioural Description Diagram for the Exfiltration Scenario



(c) Behavioural Description Diagram for the Exfiltration Scenario



(d) Behavioural Description Diagram for the Exfiltration Scenario

Figure 5.11: Behavioural Description Diagram for the Exfiltration Scenario

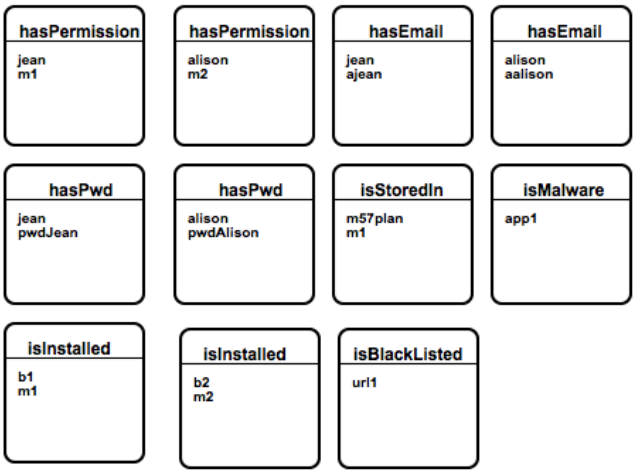


Figure 5.12: Initially Diagram for the Exfiltration Scenario

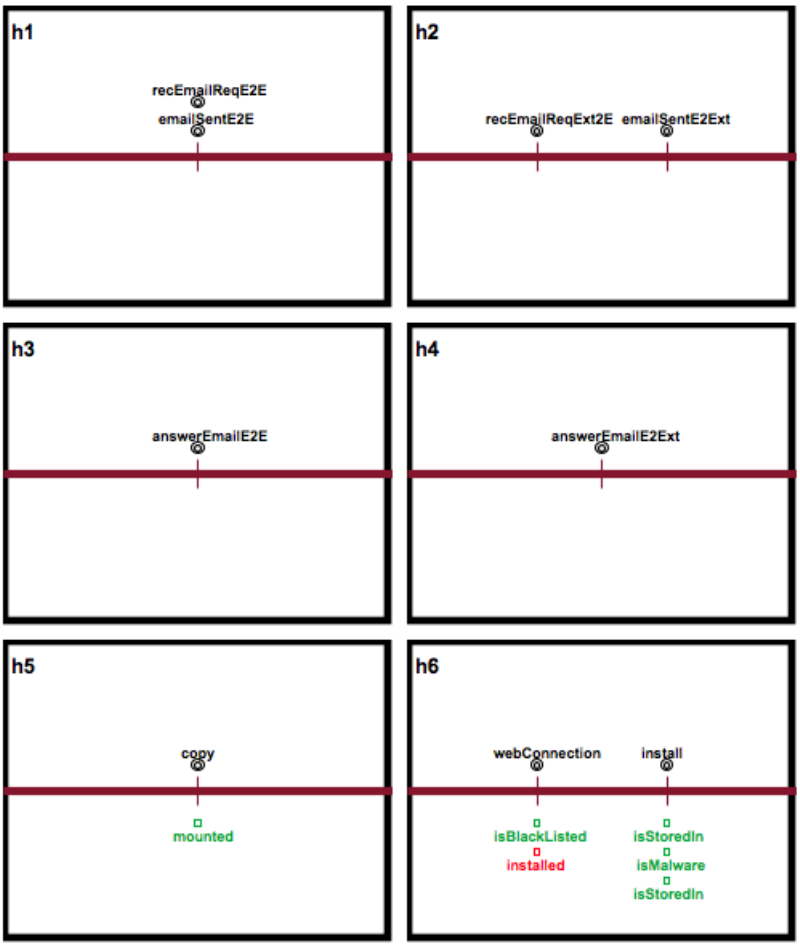


Figure 5.13: Hypothesis Diagram for the Exfiltration Scenario

5.1.3 Harassment Scenario

This scenario involves the Nitroba University and takes place in Summer 2008. Lily Tuckrige, a teacher in the Chemistry Department of the university has been receiving harassing emails and she suspects that one student of hers is sending them all. The environment involves the teacher, her students and a system administrator that manages the local network of the university. The system administrator is provided with the header of the emails received by the teacher. The header of the emails show the sender IP address and thanks to this information, the system administrator understands that those messages are coming from the university dormitory. Based on these information it is possible to formulate some incident hypotheses such as the sending of emails to an academic email address from both internal or external addresses. The environment and the incident hypotheses have been successfully described with diagrams and encoded into Event-Calculus like the other two scenarios.

5.2 Discussion

The development of the graphical designer took exactly 5 months of work which I did at the University College of Dublin. At my arrival at the beginning of March I started to study the paper of Prof. Pasquale [4] that has been the main reference of my work because it describes the approach of kEEPER. Then I made the first version of the meta-model and I started to study EMF that thankfully was quite well–documented with tutorials and examples. This first step towards the development of the meta-model took a month and a half after which I already had the generated Java source code that allowed me to start creating some instances of the meta-model. At this stage I understood which were the modifications to make on the meta-model before proceeding to develop the graphical designer with GMF. I began working with the Graphical Modeling Framework at the end of April but the task was not easy as learning EMF for the lack of documentation. GMF unfortunately is not well-documented at the moment and the only way to understand it deeply was browsing the source code and looking for information on the internet in forums

and blogs. There is also a small tutorial online which is useful to learn the basic functionalities of the framework but if you want to fully customise your graphical designer the only way to do that is having patience and a bit of time. The framework is based on GEF, so learning how to use the Graphical Editing Framework was the first step I performed. It took me a while to know enough about GMF but at the end of May there was a first version of the graphical designer ready.

Not much of the generated code of EMF has been changed to accomplish my purposes. I cannot say the same for GMF: a lot of adjustments have been made on the source code. This task was time consuming because of the lack of documentation and because some graphical elements had to be created from scratch since GMF does not support a full customisation. For instance, I had to use custom figures that imply writing code by hand and I also had to make the same changes in different classes because the automatic generation of the code did not provide the customisation I wanted. The graphical designer was ready at the beginning of August and now kEEPER can benefit from it.

On my personal opinion EMF and GMF are two very powerful frameworks that allow the developer to make graphical designers for specific domain applications saving some time. In particular, it is clear that building meta-models with EMF saves a lot of time. Regarding GMF, it should have an official documentation and it should have some improvements on the possible customisations of the generated source code but I think that writing the code from scratch would have been more time consuming.

However I achieved the initial goals of my thesis which were solving the previous limitations of usability, correctness and reusability of kEEPER. At the end of the project I can also say that there are currently two main limitations that could be overcome in future works.

When the user has to define the Behavioural Description objects and he/she wants to define several ones, each one representing a possible combinations in terms of time instants of the predicates, the graphical designer requires him/her to add to the diagram a number of nodes equals to the number of those possible combinations.

This is quite a long task for the user that needs to create and set all the properties of the nodes manually but it is easy to overcome with an algorithm that would allow the user to create and set only one node in the diagram. Currently the graphical representation of such node exists but the algorithm needs to be implemented.

At the moment, when the user wants to create the Behavioural Description diagram and the Initially diagram, and in particular, when he/she sets the references to the objects of other diagrams, a guided procedure shows up in a dialog window and the user needs to follow it. It would be way better if the user had the possibility to perform such actions directly from the property area of each diagram without the use of any dialog. This is a challenging improvements because it requires a deep knowledge of EMF since it is not possible to generate the required source code and the developer needs to manually change it after the automatic generation.

Chapter 6

State of the Art

In the literature there are no implemented approaches to build forensic-ready systems. For this reason my State of the Art cannot be based on comparing graphical designers for engineering forensic-ready systems. I divide this chapter into two sections. In the first part I describe an approach that aims to define a specific language format to describe digital evidence files useful for the digital forensic investigation. Then, I compare the approach to mine describing the common features.

In the second part I describe other forensic readiness approaches that are present in the literature.

6.1 Derric Domain-Specific Language

Derric is a domain specific language (DSL) for declaratively specifying data structures. In digital forensics, since a large quantities of data from different sources has to be analysed, there is the need of declaratively specifying a unique data format that can be used to model different digital evidences. This allows a separation of tasks between data description and data analysis.

A Derric description contains information about a particular file that constitutes the digital evidence. It normally provides some details about the length of a specific section of the digital evidence file, the version of the file or it indicates what to do in case of one or more fragments of the file got deleted or destroyed. Evidences

that have different sources and that are stored in different file formats (for instance PNG and JPEG files) can now have the same description formats. This is important because, most of the time, the files that constitute digital evidence are damaged and fragmented. Therefore, given the description of the files in Derric, it is possible to recollect the relevant data for the investigation thanks to the guides provided by this description.

The recollection, referred in the paper as the "Matching" process, is done by implemented Java classes that also reassemble those files that have been fragmented. The Java classes are generated automatically starting from the Derric specifications.

The combination of both the use of the domain specific-language Derric and the code generator can be referred to as Model-Driven Engineering approach that is the same approach that I use to build my graphical designer. In particular, I use this approach twice when I build the meta-model to describe environment and hypotheses and when I build the graphical designer to describe the graphical elements that I use in the diagrams.

Both Derric and my work fit into the workflow of a digital forensics investigator. Therefore the two approaches are comparable: kEEPER graphical designer aims to describe the environment and the incident hypotheses that may occur within the environment in a standard way, while Derric aims to describe the digital evidence that comes from different digital sources in a standard way too. kEEPER starts from the definition of the environment and incident hypotheses done with the graphical designer and these data is then processed to prescribe the preservation specifications; the definition of digital evidence with Derric is also then processed for data recollection.

The table 7.1 summarises the differences between my work and the above described DSL Derric.

Both the graphical designer I developed and Derric have a role in the digital investigation: my work is a graphical designer for kEEPER that aims to prescribe preservation specifications and Derric aims to recollect data from different digital sources. They are both based on the Model-Drive Engineering approach: my work is

	Fit into the investigation process	Model-Driven Engineering	What does it describe?	Aims
My work	X	DSLs + Code Generators	Environment + Hypotheses	Preservation Specifications
Derric	X	DSL	Digital Evidence	Data Recollection

Table 6.1: Comparisons between Derric and my work

based on the use of two different domain-specific languages to describe environment and incident hypotheses, one for the definition of the meta-model and another one for the graphical designer, Derric instead is a domain-specific language to describe digital evidence.

6.2 Forensic Readiness approaches

A lot of different approaches have been described by researchers to achieve forensic readiness along the lines of implementing policies and processes, aligning systems wth forensics objectives and the training of employees [22].

Policies indicate what needs to be done to achieve forensic readiness in an organisation. In the paper of Barske et al. [23] it is stated that, to ensure digital forensic readiness, there is the need of putting in place a minimum of 9 different policies. These policies describe how the information systems of the organisation have to be monitored, how and what data needs to be preserved including regulations on the amount of time that these data will be retained, how and in which circumstances the digital evidence can be used in the investigation and when it is important to start an investigation.

Processes are also quite important: Tan in his paper [24] says that the processes of collecting the logs (what is logged and how) is crucial and they need to be defined. For instance, an investigation of 80 hours can be done in half hour if the timestamp of a particular action is logged correctly. Tan also defines the processes of forensic acquisition (for instance which disk has to be used to store the data) and evidence

handling (for instance who has access to the evidence).

Grobler and Louwrens, suggest instead the use of a digital evidence management system that organise all the digital evidence in such a way that the data retained needs to be accessible and all the meta-data (authors, date) need to be stored together with the evidence [25].

The employees of an organisation need to be trained so that they understand the role that they would have in case of an incident happens [6]. Rowlingson says that all the workers need to note dates, times and signatures of all the tasks being performed in each working day. He adds that the reports only need to be seen by the necessary people that are only the involved ones in that particular task.

An important approach that has been proposed by researchers in the past few years is known as "forensic-by-design" that consists in integrating forensic tools at design time. For instance Mink et al. [26] discuss the possibility of integrating forensic-ready principles in a next generation aircraft system. They conclude that, such principles should be integrated at design time. Another research on a cyber-physical cloud system has been conducted stating that such a system would benefit if built using a conceptual forensic-by-design framework discussed in the paper [27]. The framework takes into consideration important aspects regarding the resources an organisation has to use to face potential threats and risks and the local laws and regulations on evidence requirements that might change between different places.

Conclusion

In this thesis I described the graphical designer that has been developed to support the development of forensic-ready systems. In particular, the graphical designer facilitates the description of the environment and the hypotheses of potential incidents. These are taken as input by the kEEPER tool to automatically generate preservation specifications. These specifications prescribe a designated software component, the Forensic Readiness Controller, to preserve data received from the digital sources available in the environment, because they might be relevant to investigate potential incidents. I used the approach of Model-Driven Engineering (MDE) that allowed my software to be built on a meta-model. The graphical designer has been developed using two frameworks that are part of the Eclipse projects: EMF to create the meta-model and GMF to associate a graphical representation to the meta-model. The use of EMF and GMF allowed me to overcome some of the current limitations of the kEEPER tool. The usability has been improved thanks to the graphical designer that allows the user to create an instance of the meta-model within diagrams. The presence of the meta-model ensures correctness of the description of the environment and the incident hypotheses. Eventually, it is possible to adapt the entire environment description in the future to different circumstances just by changing the meta-model a bit and it is easy to extend the functionalities of the graphical designer because it is made up of different plugins.

To evaluate my approach I have used the graphical designer to model the environment and hypotheses related to two incident scenarios described in existing digital corpora available publicly. The graphical designer offered a general interface allowing to model different incident scenarios related to different environmental domains.

Future work will be aimed to further improve the "look-and-feel" of the graphical designer by, for example, using custom figures to represent the elements defined in the environment and the incident hypotheses.

I will also support the automated generation of all possible orderings of events and context relations in a composite definition. This will avoid the user to specify all possible orderings of events and context relations that can trigger a composite event.

Finally, I will work on the execution of a rigorous user study to assess experimentally the usability of the graphical designer and identify limitations emerging from the users' interaction.

Bibliography

[1] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[2] Tim Grance, Karen Kent, and Brian Kim. Computer security incident handling guide. *NIST Special Publication*, 800:61, 2004.

[3] Robert M Lee, Michael J Assante, and Tim Conway. German steel mill cyber attack. *Industrial Control Systems*, 30, 2014.

[4] Dalal Alrajeh, Liliana Pasquale, and Bashar Nuseibeh. On evidence preservation requirements for forensic-ready systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 559–569. ACM, 2017.

[5] Dauda Sule. Importance of forensic readiness. 1, 2014.

[6] Robert Rowlingson. A ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3):1–28, 2004.

[7] Simson L Garfinkel. Digital forensics research: The next 10 years. *digital investigation*, 7:S64–S73, 2010.

[8] Barbara Endicott-Popovsky, Deborah A Frincke, and Carol A Taylor. A theoretical framework for organizational network forensic readiness. *JCP*, 2(3):1–11, 2007.

[9] Sopra Group/ISID. Forensics readiness guidelines. *NICS*, 2011.

[10] Brian Carrier, Eugene H Spafford, et al. Getting physical with the digital investigation process. *International Journal of digital evidence*, 2(2):1–20, 2003.

[11] Zhi-Yan Xu, Yan-Pu Zhang, and Yu-Qiang Chen. The design and implementation of the mobile facility communication software based on plug-in. In *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, pages 821–827. Springer, 2013.

[12] Nitroba university harassment scenario, 2017.

[13] M57-jean, 2017.

[14] C. Falk F.P Buchholz. Design and implementation of zeitline: a forensic timeline editor. 2005. In Proc. of the Digital Forensics Research Workshop.

[15] Kamil Reddy and Hein S Venter. The architecture of a digital forensic readiness management system. *Computers & security*, 32:73–89, 2013.

[16] Clay Shields, Ophir Frieder, and Mark Maloof. A system for the proactive, continuous, and efficient collection of digital forensic evidence. *digital investigation*, 8:S3–S13, 2011.

[17] Liliana Pasquale, Sorren Hanvey, Mark Mcgloin, and Bashar Nuseibeh. Adaptive evidence collection in the cloud using attack scenarios. *Computers & Security*, 59:236–254, 2016.

[18] Murray Shanahan. The event calculus explained. In *Artificial intelligence today*, pages 409–430. Springer, 1999.

[19] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[20] Michel Wermelinger and Yijun Yu. Analyzing the evolution of eclipse plugins. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 133–136. ACM, 2008.

[21] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[22] George Grispos, Jesus Garcia-Galan, Liliana Pasquale, and Bashar Nuseibeh. Are you ready? towards the engineering of forensic-ready systems. *arXiv preprint arXiv:1705.03250*, 2017.

[23] David Barske, Adrie Stander, and Jason Jordaan. A digital forensic readiness framework for south african sme's. In *Information Security for South Africa (ISSA), 2010*, pages 1–6. IEEE, 2010.

[24] John Tan. Forensic readiness. *Cambridge, MA:@ Stake*, pages 1–23, 2001.

[25] Cornelia P Grobler and CP Louwrens. Digital forensic readiness as a component of information security best practice. In *IFIP International Information Security Conference*, pages 13–24. Springer, 2007.

[26] Dustin Mink, Alec Yasinsac, Kim-Kwang Raymond Choo, and William Glisson. Next generation aircraft architecture and digital forensic. 2016.

[27] Nurul Hidayah Ab Rahman, William Bradley Glisson, Yanjiang Yang, and Kim-Kwang Raymond Choo. Forensic-by-design framework for cyber-physical cloud systems. *IEEE Cloud Computing*, 3(1):50–59, 2016.

Appendix A

Motivating Example

Listing A.1: Event-Calculus description of the environment of the Motivating Example

```
1  % ****************** Environment Description ******************
2
3  % ********* Context Description *********
4
5  % ***** Types and Instances *****
6
7  % Room r01 is a location
8  loc(r01).
9
10 % cctv1 is a cctv camera
11 cam(cctv1).
12
13 % nfc1 is a nfs reader
14 read(nfc1).
15
16 % m1, m2 and m3 are computers
17 comp(m1;m2;m3).
18
19 % doc is a file
20 fi(doc).
21
22 % alice and bob are employees
23 emp(alice;bob).
24
25 % usb1 is an external storage device
```

```
26 st(usb1).
27
28 % ***** Context Relations *****
29
30 % A location is access-controlled by a nfc reader
31
32 fluent(isAccessControlledBy(L,R)):-
33     loc(L), read(R).
34
35 % A location is surveilled by a camera
36
37 fluent(isMonitoredBy(L,C)):-
38     loc(L), cam(C).
39
40 %A desktop is in a specific location
41
42 fluent(isLocatedIn(C,L)):-
43     comp(C), loc(L).
44
45 % An employee is in a location
46
47 fluent(in(E,L)):-
48     emp(E), loc(L).
49
50 % An employee is logged to a machine
51
52 fluent(logged(E,C)):-
53     emp(E), comp(C).
54
55
56 % An employee has a badge to access to a specific location
57
58 fluent(hasBadge(E,L)):-
59     emp(E), loc(L).
60
61 % An employee has the permission to log to a machine
62
63 fluent(hasPermission(E,C)):-
64     emp(E), comp(C).
65
66 % A file is stored on a machine
67
68 fluent(isStoredIn(F,C)):-
```

```
69      fi(F), comp(C).
70
71  % A storage device is mounted on a computer by an employee
72
73  fluent(mounted(S,C)):-
74      st(S), comp(C).
75
76
77  % ********* Behaviour Description *********
78
79  % ***** Primitive Events *****
80
81  event(V):-
82      pe(V).
83
84  % A storage device is mounted several times on a computer
85
86  pe(highMountCounts(S, C)):-
87      st(S), comp(C).
88
89  % A system log indicates that a file stored on a computer is copied by an
        employee
90
91  pe(sys_copy(E, F, C)):-
92      emp(E), fi(F), comp(C).
93
94  % A system log indicates that a storage device is mounted on a computer
95
96  pe(sys_Mount(S, C)):-
97      st(S), comp(C).
98
99  % A system log indicates that a storage device is unmounted from a computer
100
101 pe(sys_Unmount(S, C)):-
102     st(S), comp(C).
103
104 % A system log indicates that an employee performed the login to a computer
105
106 pe(sys_Login(E, C)):-
107     emp(E), comp(C).
108
109 % A system log indicates that an employee performed the logout from a computer
110
```

```
111 pe(sys_Logout(E, C)):-
112     emp(E), comp(C).
113
114 % A nfc reader is reading a card tag associated with an employee
115
116 pe(swipeCard(E, R)):-
117     emp(E), read(R).
118
119 % A camera recording indicates an employee accessing a location
120
121 pe(cctvAccess(E, L, C)):-
122     emp(E), loc(L), cam(C).
123
124 % A camera recording indicates an employee exiting a location
125
126 pe(cctvExit(E, L, C)):-
127     emp(E), loc(L), cam(C).
128
129
130 % ***** Complex Events *****
131
132 event(V):-
133     ce(V).
134
135 %An employee enters in a location
136
137 ce(enter(E,L)):-
138     emp(E), loc(L).
139
140 %An employee exits from a location
141
142 ce(exit(E,L)):-
143     emp(E), loc(L).
144
145
146 %An employee mounts a storage device on a computer
147
148 ce(mount(S,C)):-
149     st(S), comp(C).
150
151 %An employee unmounts a storage device from a computer
152
153 ce(unMount(S,C)):-
```

```
154       st(S), comp(C).
155
156  %An employee logins to a computer
157
158  ce(login(E,C)):-
159       emp(E), comp(C).
160
161  %An employee logouts from a computer
162
163  ce(logout(E,C)):-
164       emp(E), comp(C).
165
166  %An employee copies a file on a computer
167
168  ce(copy(E,F,C)):-
169       emp(E), fi(F), comp(C).
170
171
172  % ***** Composite Definitions *****
173
174  % An employee enters a location if, while she is outside the location,
175  % she swipes her card on the nfc reader controlling access to the location and
          subsequently
176  % a cctv records her access to the location.
177
178  happens(enter(E,L),T2,TR):-
179       trace(TR),
180       emp(E),
181       loc(L),
182       cam(C),
183       read(R),
184       time(T1),
185       time(T2),
186       T1<T2,
187       holdsAt(hasBadge(E,L),T1,TR),
188       happens(swipeCard(E,R),T1,TR),
189       holdsAt(isAccessControlledBy(L,R),T2,TR),
190       happens(cctvAccess(E,L,C),T2,TR),
191       holdsAt(isMonitoredBy(L,C),T2,TR),
192       neg_holdsAt_between(T1, in(E,L),T2,TR).
193
194
195  % An employee enters a location if, while she is outside the location,
```

```
196 % a cctv records her access to the location and subsequently
197 % she swipes her card on the nfc reader controlling access to the location.
198
199 happens(enter(E,L),T2,TR):-
200       trace(TR),
201       emp(E),
202       loc(L),
203       cam(C),
204       read(R),
205       time(T1),
206       time(T2),
207       T1<T2,
208       holdsAt(hasBadge(E,L),T2,TR),
209       happens(swipeCard(E,R),T2,TR),
210       holdsAt(isAccessControlledBy(L,R),T1,TR),
211       happens(cctvAccess(E,L,C),T1,TR),
212       holdsAt(isMonitoredBy(L,C),T1,TR),
213       neg_holdsAt_between(T1, in(E,L),T2,TR).
214
215
216 % An employee enters a location if, while she is outside the location,
217 % a cctv records her access to the location and at the same time
218 % she swipes her card on the nfc reader controlling access to the location.
219
220 happens(enter(E,L),T2,TR):-
221       trace(TR),
222       emp(E),
223       loc(L),
224       cam(C),
225       read(R),
226       time(T1),
227       time(T2),
228       T1=T2,
229       holdsAt(hasBadge(E,L),T2,TR),
230       not holdsAt(in(E,L),T2,TR),
231       happens(swipeCard(E,R),T2,TR),
232       holdsAt(isAccessControlledBy(L,R),T1,TR),
233       happens(cctvAccess(E,L,C),T1,TR),
234       holdsAt(isMonitoredBy(L,C),T1,TR).
235
236
237 % An employee exits a location if, while she is inside the location,
```

```
238 % she swipes her card on the nfc reader controlling access to the location and
        subsequently
239 % a cctv records her access to the location.
240
241 happens(exit(E,L),T2,TR):-
242       trace(TR),
243       emp(E),
244       loc(L),
245       time(T1),
246       time(T2),
247       read(R),
248       cam(C),
249       T1<T2,
250       holdsAt(hasBadge(E,L),T1,TR),
251       happens(swipeCard(E,R),T1,TR),
252       holdsAt(isAccessControlledBy(L,R),T2,TR),
253       happens(cctvExit(E,L,C),T2,TR),
254       holdsAt(isMonitoredBy(L,C),T2,TR),
255       holdsAt_between(T1, in(E,L),T2,TR).
256
257
258 % An employee exits a location if, while she is inside the location,
259 % a cctv records her exit from the location and subsequently
260 % she swipes her card on the nfc reader controlling access to the location.
261
262 happens(exit(E,L),T2,TR):-
263       trace(TR),
264       emp(E),
265       loc(L),
266       time(T1),
267       time(T2),
268       read(R),
269       cam(C),
270       T1<T2,
271       holdsAt(hasBadge(E,L),T2,TR),
272       happens(swipeCard(E,R),T2,TR),
273       holdsAt(isAccessControlledBy(L,R),T1,TR),
274       happens(cctvExit(E,L,C),T1,TR),
275       holdsAt(isMonitoredBy(L,C),T1,TR),
276       holdsAt_between(T1, in(E,L),T2,TR).
277
278
279 % An employee exits a location if, while she is inside the location,
```

```
280 % a cctv records her exit from the location and at the same time
281 % she swipes her card on the nfc reader controlling access to the location.
282
283 happens(exit(E,L),T2,TR):-
284     trace(TR),
285     emp(E),
286     loc(L),
287     time(T1),
288     time(T2),
289     read(R),
290     cam(C),
291     T1==T2,
292     holdsAt(hasBadge(E,L),T2,TR),
293     holdsAt(in(E,L),T2,TR),
294     happens(swipeCard(E,R),T2,TR),
295     holdsAt(isAccessControlledBy(L,R),T1,TR),
296     happens(cctvExit(E,L,C),T1,TR),
297     holdsAt(isMonitoredBy(L,C),T1,TR).
298
299 % An employee logins to a computer if the system log of the computer indicates
        that
300 % the employee has performed the login and she is authorised to access the
      computer,
301 % and there is at least an employee in the location where the computer is
      placed.
302
303 happens(login(E,C),T,TR):-
304     trace(TR),
305     emp(E),
306     comp(C),
307     loc(L),
308     time(T),
309     happens(sys_Login(E,C),T,TR),
310     emp(E2),
311     holdsAt(in(E2,L),T,TR),
312     holdsAt(isLocatedIn(C,L),T,TR),
313     holdsAt(hasPermission(E,C),T,TR).
314
315 % An employee logouts from a computer if she is logged on the computer and
316 % the system log of the computer indicates that
317 % the employee has performed the logout
318
319 happens(logout(E,C),T,TR):-
```

```
320       trace(TR),
321       emp(E),
322       comp(C),
323       time(T),
324       happens(sys_Logout(E,C),T,TR),
325       holdsAt(logged(E,C),T,TR).
326
327  % A storage device is mounted on a computer if it is currently not mounted,
328  % the system log of the computer indicates that the device is mounted
329  % and there is an employee logged on the computer
330
331  happens(mount(S,C),T,TR):-
332       trace(TR),
333       time(T), comp(C), st(S), emp(E),
334       happens(sys_Mount(S,C),T,TR),
335       not holdsAt(mounted(S,C),T,TR),
336       holdsAt(logged(E,C),T,TR).
337
338  % A storage device is unmounted from a computer if it is currently mounted,
339  % and the system log of the computer indicates that the device is unmounted
340
341  happens(unMount(S,C),T,TR):-
342       trace(TR),
343       time(T), comp(C), st(S),
344       happens(sys_Unmount(S,C),T,TR),
345       holdsAt(mounted(S,C),T,TR).
346
347  % A file is copied on a computer by an employee if
348  % the system log of the computer indicates that the file was copied,
349  % the file is stored on the computer,
350  % and the employee is logged to the computer.
351
352  happens(copy(E,F,C),T,TR):-
353       trace(TR),
354       time(T), fi(F), comp(C), emp(E),
355       happens(sys_copy(E,F,C),T,TR),
356       holdsAt(isStoredIn(F,C),T,TR),
357       holdsAt(logged(E,C),T,TR).
358
359
360  % +++ Context Relation Definitions +++
361
362  % If an employee enters in a location, s/he is the location
```

```
363
364 initiates(enter(E,L),in(E,L), T):-
365     emp(E), loc(L), time(T).
366
367
368 %If an employee mounts a storage on a computer, the storage is mounted on the
        computer by an employee
369
370 initiates(mount(S,C),mounted(S,C), T):-
371     st(S), comp(C), time (T).
372
373 %If an employee logins to a computer, she is logged to the computer
374
375 initiates(login(E,C),logged(E,C), T):-
376     emp(E), comp(C), time(T).
377
378 %If an employee exits a location, she is no longer inside the location
379
380 terminates(exit(E,L),in(E,L), T):-
381     emp(E), loc(L), time(T).
382
383 %If a storage device is unmounted from a computer, it is no longer mounted on
        the computer
384
385 terminates(unMount(S,C),mounted(S,C), T):-
386     st(S), comp(C), time(T).
387
388 % If an employee logouts from a computer, she is no longer logged to the
        computer
389
390 terminates(logout(E,C),logged(E,C), T):-
391     emp(E), comp(C), time(T).
392
393 % Initial state
394
395 initially(isMonitoredBy(r01, cctv1)).
396 initially(isAccessControlledBy(r01, nfc1)).
397 initially(isLocatedIn(m1, r01)).
398 initially(isStoredIn(doc, m1)).
399 initially(hasBadge(alice, r01)).
400 initially(hasPermission(alice, m1)).
401 initially(hasPermission(alice, m3)).
402 initially(hasBadge(bob, r01)).
```

```
403 initially(hasPermission(bob, m1)).
404 initially(hasPermission(bob, m2)).
```

Appendix B

Exfiltration Scenario

Listing B.1: Event-Calculus description of the environment of the Exfiltration Scenario

```
% ****************** Environment Description ******************

% ********* Context Description *********

% ***** Types and Instances *****

% m1 and m2 are computers
comp(m1;m2).

%employees
emp(alison;jean;bob).

% urls
url(url1;url2).

% browsers
browser(b1;b2).

% employees' emails
addr(aalison;ajean).

% sender and recipient emails
sendAddr(aalison;ajean).
recAddr(aalison;ajean).

% external email addresses
```

```
27 extAddr(abob;atuck).
28
29 % external sender and recipient email addresses
30 sendExtAddr(abob;atuck).
31 recExtAddr(abob;atuck).
32
33 % email body
34 email(e1).
35
36 % employees' roles
37 role(president;cfo).
38
39 % passwords for employees electronic identities
40 pwd(pwdJean;pwdAlison).
41
42 % confidential file
43 fi(m57plan).
44
45 % external storage device
46 st(usb1).
47
48 % applications
49 app(app1;app2).
50
51
52 % ***** Context Relations *****
53
54 % An employee is logged on a computer
55
56 fluent(logged(E,C)):-
57     emp(E), comp(C).
58
59 %A employee has the permission to access to a computer
60
61 fluent(hasPermission(E,C)):-
62     emp(E), comp(C).
63
64
65 %A file is stored on a computer
66
67 fluent(isStoredIn(F,C)):-
68     fi(F), comp(C).
69
```

```
70 %An employee has an email address
71
72 fluent(hasEmail(P,E)):-
73     emp(P), addr(E).
74
75 fluent(hasEmail(P,E)):-
76     emp(P), sendAddr(E).
77
78 fluent(hasEmail(P,E)):-
79     emp(P), recAddr(E).
80
81 %An employee has a password associated with his/her email address
82 fluent(hasPwd(E,P)):-
83     emp(E), pwd(P).
84
85 %An employee has a role
86 fluent(hasRole(E,R)):-
87     emp(E), role(R).
88
89 %A storage device is mounted on a computer
90 fluent(mounted(S,C)):-
91     st(S), comp(C).
92
93
94 % An application is a malware
95 fluent(isMalware(A)):-
96     app(A).
97
98 % A url is blacklisted
99 fluent(isBlacklisted(U)):-
100     url(U).
101
102 % An application is installed on a computer
103 fluent(installed(A,C)):-
104     app(A), comp(C).
105
106 % A browser is installed on a computer
107 fluent(isInstalled(B,C)):-
108     browser(B), comp(C).
109
110
111 % ********* Behaviour Description *********
112
```

```
113 % ***** Primitive Events *****
114
115 event(V):-
116     pe(V).
117
118 % From a computer hard drive it is possible to infer that an email E with an
119 % attachment A is sent from an internal corporate address S to an internal
120 % corporate address R
121 pe(send_email_e2e(E,A,S,R,C)):-
122     email(E), fi(A), sendAddr(S), recAddr(R), comp(C).
123
124 % From a computer hard drive it is possible to infer that an email E with an
125 % attachment A is sent from an internal corporate address S to an external
126 % address R
127 pe(send_email_e2ext(E,A,S,R,C)):-
128     email(E), fi(A), addr(S), extAddr(R), comp(C).
129
130 % From a computer hard drive it is possible to infer that an email E with an
131 % attachment A is sent from an external address S to an internal corporate
132 % address R
133 pe(send_email_ext2e(E,A,S,R,C)):-
134     email(E), fi(A), extAddr(S), addr(R), comp(C).
135
136 % From a computer hard drive it is possible to infer that an email E with an
137 % attachment A is sent from an external address S to an external address R
138 pe(send_email_ext2ext(E,A,S,R,C)):-
139     email(E), fi(A), sendExtAddr(S), recExtAddr(R), comp(C).
140
141 % From a computer hard drive it is possible to infer that an email E is
        received
142 % by an internal corporate address S from an internal corporate address R
143 pe(rec_email_req_e2e(E,S,R,C)):-
144     email(E), sendAddr(S), recAddr(R), comp(C).
145
146 % From a computer hard drive it is possible to infer that an email E is
        received
147 % by an internal corporate address S from an external address R
148 pe(rec_email_req_e2ext(E,S,R,C)):-
149     email(E), addr(S), extAddr(R), comp(C).
150
151 % From a computer hard drive it is possible to infer that an email E is
        received
152 % by an external address S from an internal corporate address R
```

```
153 pe(rec_email_req_ext2e(E,S,R,C)):-
154     email(E), extAddr(S), addr(R), comp(C).
155
156 % From a computer hard drive it is possible to infer that an email E is
        received
157 % by an external address S from an external address R
158 pe(rec_email_req_ext2ext(E,S,R,C)):-
159     email(E), sendExtAddr(S), recExtAddr(R), comp(C).
160
161
162 % From the system log of computer C it is possible to infer that an employee E
163 % performed the login using her password P
164 pe(sys_login(E,P, C)):-
165     emp(E), pwd(P), comp(C).
166
167 % From the system log of computer C it is possible to infer that an employee E
168 % did the logout
169 pe(sys_logout(E, C)):-
170     emp(E), comp(C).
171
172
173 % From the system log of computer C it is possible to infer that a storage
        device S
174 % was mounted
175 pe(sys_mount(S,C)):-
176     st(S), comp(C).
177
178 % From the system log of computer C it is possible to infer that a storage
        device S
179 % was unmounted
180 pe(sys_unmount(S,C)):-
181     st(S), comp(C).
182
183 % From the system log of computer C it is possible to infer that employee E
        copied
184 % file F
185 pe(sys_copy(E, F, C)):-
186     emp(E), fi(F), comp(C).
187
188 % From the hard drive of computer C it is possible to infer that application A
         was installed
189 pe(sys_install(A,C)):-
190     app(A), comp(C).
```

```
191
192 % From the hard drive of computer C it is possible to infer that application A
        was uninstalled
193 pe(sys_uninstall(A,C)):-
194     app(A), comp(C).
195
196 % From the web history of computer C it is possible to infer that application
        a connection to a blacklisted
197 % url from browser B was performed
198 pe(web_connection(B,U,C)):-
199     browser(B), url(U), comp(C).
200
201 % ***** Complex Events *****
202
203 event(V):-
204     ce(V).
205
206 % An email with attachment A is sent by an employee P1 to employee P2
207 ce(emailSentE2E(E,A,P1,P2)):-
208     email(E), fi(A), emp(P1), emp(P2).
209
210 % An email with attachment A is sent by an employee P to an external address R
211 ce(emailSentE2Ext(E,A,P,R)):-
212     email(E), fi(A), emp(P), extAddr(R).
213
214 % An email with attachment A is sent by an employee P to an external address R
215 % (Sender and recipient addresses are the same)
216 ce(emailSentExt2ExtS(E,A,P,R)):-
217     email(E), fi(A), emp(P), extAddr(R).
218
219 % An email with attachment A is sent by an employee P to an external address R
220 % (Sender and recipient addresses are the different)
221 ce(emailSentExt2ExtD(E,A,P,R)):-
222     email(E), fi(A), emp(P), extAddr(R).
223
224 % An email is received by an employee P1 from another employee P2
225 ce(recEmailReqE2E(E,P1,P2)):-
226     email(E), emp(P1), emp(P2).
227
228 % An email is received by an employee P from an external address S
229 ce(recEmailReqExt2E(E,S,P)):-
230     email(E), extAddr(S), emp(P).
231
```

```
232
233 % An email previously received by employee P1 from P2 is answered by also
        including
234 % an attachment A
235 ce(answerEmailE2E(E1,E2,A,P1,P2)):-
236       email(E1), email(E2), fi(A), emp(P1), emp(P2).
237
238 % An email previously received by employee P from an external address S is
        answered by also including
239 % an attachment A
240 ce(answerEmailE2Ext(E1,E2,A,P,S)):-
241       email(E1), email(E2), fi(A), emp(P), extAddr(S).
242
243
244 % An employee logins to a computer
245 ce(login(E,C)):-
246       emp(E), comp(C).
247
248 % An employee logouts from a computer
249 ce(logout(E,C)):-
250       emp(E), comp(C).
251
252 % A device is mounted on a computer
253 ce(mount(S,C)):-
254       st(S), comp(C).
255
256 % A device is unmounted from a computer
257 ce(unmount(S,C)):-
258       st(S), comp(C).
259
260 % An employee copies a file on a computer
261 ce(copy(E,F,C)):-
262       emp(E), fi(F), comp(C).
263
264 % An application is installed on a computer
265 ce(install(A,C)):-
266       app(A),comp(C).
267
268 % An application is uninstalled from a computer
269 ce(uninstall(A,C)):-
270       app(A),comp(C).
271
```

```
272 % A web connection to url U is performed through a browser B installed on a
        computer
273 ce(webConnection(B,U,C)):-
274     browser(B), url(U), comp(C).
275
276
277 % ***** Composite Definitions *****
278
279 % An employee who is entitled to access to computer C
280 % performs the login
281
282 happens(login(E,C),T,TR):-
283     emp(E),
284     comp(C),
285     time(T),
286     trace(TR),
287     emp(E2),
288     happens(sys_login(E,P,C),T,TR),
289     holdsAt(hasPermission(E,C),T,TR),
290     holdsAt(hasPwd(E,P),T,TR).
291
292 % An employee who is entitled to access to computer C
293 % and is currently logged to C performs the logout
294
295 happens(logout(E,C),T,TR):-
296     emp(E),
297     comp(C),
298     time(T),
299     trace(TR),
300     happens(sys_logout(E,C),T,TR),
301     holdsAt(logged(E,C),T,TR).
302
303 % An email with attachment A is sent by an employee P1, who is currently
        logged on computer C,
304 % to an employee P2 by using an internal corporate address
305
306 happens(emailSentE2E(E,A,P1,P2),T,TR):-
307     trace(TR),
308     email(E),
309     fi(A),
310     sendAddr(S),
311     recAddr(R),
312     time(T),
```

```
313       comp(C),
314       emp(P1), emp(P2),
315       happens(send_email_e2e(E,A,S,R,C),T,TR),
316       holdsAt(hasEmail(P1,S),T,TR),
317       holdsAt(logged(P1,C),T,TR),
318       holdsAt(hasEmail(P2,R),T,TR).
319
320  % An email with attachment A is sent by an employee P1, who is currently
       logged on computer C,
321  % to an employee P2 by using an external address
322
323  happens(emailSentE2E(E,A,P1,P2),T,TR):-
324       trace(TR),
325       email(E),
326       fi(A),
327       extAddr(S),
328       addr(R),
329       time(T),
330       comp(C),
331       emp(P1), emp(P2),
332       happens(send_email_ext2e(E,A,S,R,C),T,TR),
333       holdsAt(hasEmail(P2,R),T,TR),
334       holdsAt(logged(P1,C),T,TR).
335
336  % An email with attachment A is sent to an external email address R by an
       employee
337  % P who is  using his/her internal corporate address and is currently logged
       on computer C
338
339  happens(emailSentE2Ext(E,A,P,R),T,TR):-
340       trace(TR),
341       email(E),
342       fi(A),
343       addr(S),
344       extAddr(R),
345       time(T),
346       comp(C),
347       emp(P),
348       happens(send_email_e2ext(E,A,S,R,C),T,TR),
349       holdsAt(hasEmail(P,S),T,TR),
350       holdsAt(logged(P,C),T,TR).
351
```

```
352 % An email with attachment A is sent to an external email address by an
        employee
353 % who is  using an external email account and is currently logged on computer
        C
354 % (Sender and recipient addresses are the same)
355
356 happens(emailSentExt2ExtS(E,A,P,R),T,TR):-
357       trace(TR),
358       email(E),
359       fi(A),
360       sendExtAddr(S),
361       recExtAddr(R),
362       S == R,
363       time(T),
364       comp(C),
365       emp(P),
366       happens(send_email_ext2ext(E,A,S,R,C),T,TR),
367       holdsAt(logged(P,C),T,TR).
368
369
370 % An email with attachment A is sent to an external email address by an
        employee
371 % who is  using an external email account and is currently logged on computer
        C
372 % (Sender and recipient addresses are different)
373
374 happens(emailSentExt2ExtD(E,A,P,R),T,TR):-
375       email(E),
376       fi(A),
377       sendExtAddr(S),
378       recExtAddr(R),
379       S != R,
380       time(T),
381       comp(C),
382       emp(P),
383       trace(TR),
384       happens(send_email_ext2ext(E,A,S,R,C),T,TR),
385       holdsAt(logged(P,C),T,TR).
386
387 %An email is received by an employee P1 from another employee P2
388 happens(recEmailReqE2E(E,P1,P2),T,TR):-
389       trace(TR),
390       email(E),
```

```
391       emp(P1),
392       emp(P2),
393       comp(C),
394       time(T),
395       sendAddr(S),
396       recAddr(R),
397       happens(rec_email_req_e2e(E,S,R,C),T,TR),
398       holdsAt(hasEmail(P2,R),T,TR),
399       holdsAt(hasEmail(P1,S),T,TR),
400       holdsAt(logged(P2,C),T,TR).
401
402  % An email is received by an employee, who is currently logged to a computer,
       from another employee.
403  % The email was sent to an external email account
404
405  happens(recEmailReqE2E(E,P1,P2),T,TR):-
406       trace(TR),
407       email(E),
408       emp(P1),
409       emp(P2),
410       comp(C),
411       time(T),
412       addr(S),
413       extAddr(R),
414       happens(rec_email_req_e2ext(E,S,R,C),T,TR),
415       holdsAt(hasEmail(P1,S),T,TR),
416       holdsAt(logged(P2,C),T,TR).
417
418  %An email is received by an employee, who is currently logged to a computer,
       from an external email address
419
420  happens(recEmailReqExt2E(E,S,P),T,TR):-
421       trace(TR),
422       email(E),
423       emp(P),
424       time(T),
425       comp(C),
426       extAddr(S),
427       addr(R),
428       happens(rec_email_req_ext2e(E,S,R,C),T,TR),
429       holdsAt(hasEmail(P,R),T,TR),
430       holdsAt(logged(P,C),T,TR).
431
```

```
% An email is received by an employee P, who is currently logged to a computer
    , from an external email address
% to an external email account
% (Email address of the sender and recipient are different)

happens(recEmailReqExt2E(E,S,P),T,TR):-
    trace(TR),
    email(E),
    emp(P),
    time(T),
    comp(C),
    sendExtAddr(S),
    recExtAddr(R),
    S != R,
    happens(rec_email_req_ext2ext(E,S,R,C),T,TR),
    holdsAt(logged(P,C),T,TR).


% An email previously received by employee P1 from P2 is answered by also
    including
% an attachment A

happens(answerEmailE2E(E1,E2,A,P1,P2),T2,TR):-
    trace(TR),
    email(E1),
    email(E2),
    P1 != P2,
    fi(A),
    emp(P1),
    emp(P2),
    time(T1),
    time(T2),
    T2 > T1,
    happens(recEmailReqE2E(E1,P2,P1),T1,TR),
    happens(emailSentE2E(E2,A,P1,P2),T2,TR).


% An email previously received by employee P from an external address S is
    answered by also including
% an attachment A and using an internal corporate address

happens(answerEmailE2Ext(E1,E2,A,P,S),T2,TR):-
    trace(TR),
```

```
472       email(E1),
473       email(E2),
474       fi(A),
475       emp(P),
476       extAddr(S),
477       time(T2),
478       time(T1),
479       T2 > T1,
480       happens(emailSentE2Ext(E2,A,P1,S),T2,TR),
481       happens(recEmailReqExt2E(E1,S,P),T1,TR).
482
483  % An email previously received by employee P from an external address S is
       answered by also including
484  % an attachment A using an external address
485
486  happens(answerEmailE2Ext(E1,E2,A,P,S),T2,TR):-
487       trace(TR),
488       email(E1),
489       email(E2),
490       fi(A),
491       emp(P),
492       extAddr(S),
493       time(T2),
494       time(T1),
495       T2 > T1,
496       happens(emailSentExt2ExtD(E2,A,P1,S),T2,TR),
497       happens(recEmailReqExt2E(E1,A,S,P),T1,TR).
498
499
500  % File F stored in computer C is copied by an employee who is logged to C
501
502  happens(copy(E,F,C),T,TR):-
503       time(T), fi(F), comp(C), emp(E),
504       trace(TR),
505       happens(sys_copy(E,F,C),T,TR),
506       holdsAt(isStoredIn(F,C),T,TR),
507       holdsAt(logged(E,C),T,TR).
508
509  % An external storage device is mounted on a computer while an employee is
       logged to the computer
510
511  happens(mount(S,C),T, TR):-
512       time(T), comp(C), st(S), emp(E),
```

```
513     trace(TR),
514     happens(sys_mount(S,C),T,TR),
515     not holdsAt(mounted(S,C),T,TR),
516     holdsAt(logged(E,C),T,TR).
517
518 % An external storage device who is currently mounted on a computer is
        unmounted
519
520 happens(unmount(S,C),T,TR):-
521     time(T), comp(C), st(S),
522     trace(TR),
523     happens(sys_unmount(S,C),T,TR),
524     holdsAt(mounted(S,C),T,TR).
525
526 % An application A is installed on computer C while an employee who is
        entitled to access
527 % the computer is logged
528
529 happens(install(A,C),T,TR):-
530     app(A),
531     comp(C),
532     time(T),
533     emp(P),
534     trace(TR),
535     happens(sys_install(A,C),T,TR),
536     holdsAt(logged(P,C),T,TR),
537     not holdsAt(installed(A,C),T,TR),
538     holdsAt(hasPermission(P,C),T,TR).
539
540 % An application A that is currently installed on computer C is uninstalled
        while an employee who is entitled to access
541 % the computer is logged
542 happens(uninstall(A,C),T,TR):-
543     trace(TR),
544     app(A),
545     comp(C),
546     time(T),
547     emp(P),
548     happens(sys_uninstall(A,C),T,TR),
549     holdsAt(logged(P,C),T,TR),
550     holdsAt(installed(A,C),T,TR),
551     holdsAt(hasPermission(P,C),T,TR).
552
```

```
553 % A web connection to url U is performed from a browser installed on a
        computer
554 % while an employee entitled to access the computer is logged
555
556 happens(webConnection(B,U,C),T,TR):-
557     trace(TR),
558     browser(B),
559     url(U),
560     comp(C),
561     time(T),
562     happens(web_connection(B,U,C),T,TR),
563     holdsAt(isInstalled(B,C),T,TR),
564     holdsAt(logged(E,C),T,TR),
565     holdsAt(hasPermission(E,C),T,TR).
566
567
568 % +++ Context Relation Definitions +++
569
570 % If an employee performs the login to a computer, then she is logged to the
        computer
571
572 initiates(login(E,C),logged(E,C), T):-
573     emp(E), comp(C), time(T).
574
575 % If an employee performs the logout from a computer, then she is no longer
        logged to the computer
576
577 terminates(logout(E,C),logged(E,C), T):-
578     emp(E), comp(C), time(T).
579
580 % If a storage device is being mounted on a computer, then it is considered as
       mounted on the computer
581
582 initiates(mount(S,C), mounted(S,C), T):-
583     st(S), comp(C), time(T).
584
585 % If a storage device is being unmounted from a computer, then it is no longer
       considered mounted on the computer
586
587 terminates(unmount(S,C), mounted(S,C), T):-
588     st(S), comp(C), time(T).
589
```

```
590 % If an application is being installed on a computer, then it is considered as
        installed on the computer
591
592 initiates(install(A,C), installed(A,C), T):-
593     app(A), comp(C), time(T).
594
595 % If an application is being uninstalled on a computer, then it is no longer
        installed on the computer
596
597 terminates(uninstall(A,C), installed(A,C), T):-
598     app(A), comp(C), time(T).
599
600 %;******** Initial State *********
601
602 %Alison and Jean have permission to login to M2 and M1, respectively.
603 initially(hasPermission(jean,m1)).
604 initially(hasPermission(alison,m2)).
605
606 initially(hasEmail(jean,ajean)).
607 initially(hasEmail(alison,aalison)).
608
609 initially(hasPwd(jean, pwdJean)).
610 initially(hasPwd(alison, pwdAlison)).
611
612 initially(isStoredIn(m57plan,m1)).
613
614 initially(isMalware(app1)).
615
616 initially(isInstalled(b1,m1)).
617 initially(isInstalled(b2,m2)).
618
619 initially(isBlacklisted(url1)).
620 %
      ********************************************************************************
```