A tool for specifying and validating software liability

VERIMAG, Grenoble, France
- Eduardo Mazza
- Marie-Laure Potet

Outline

- Context
- Approach
- Study Case
- Specifications
  - Entities
  - Logs
- Properties
- Responsibility
- Future Work
- Conclusions

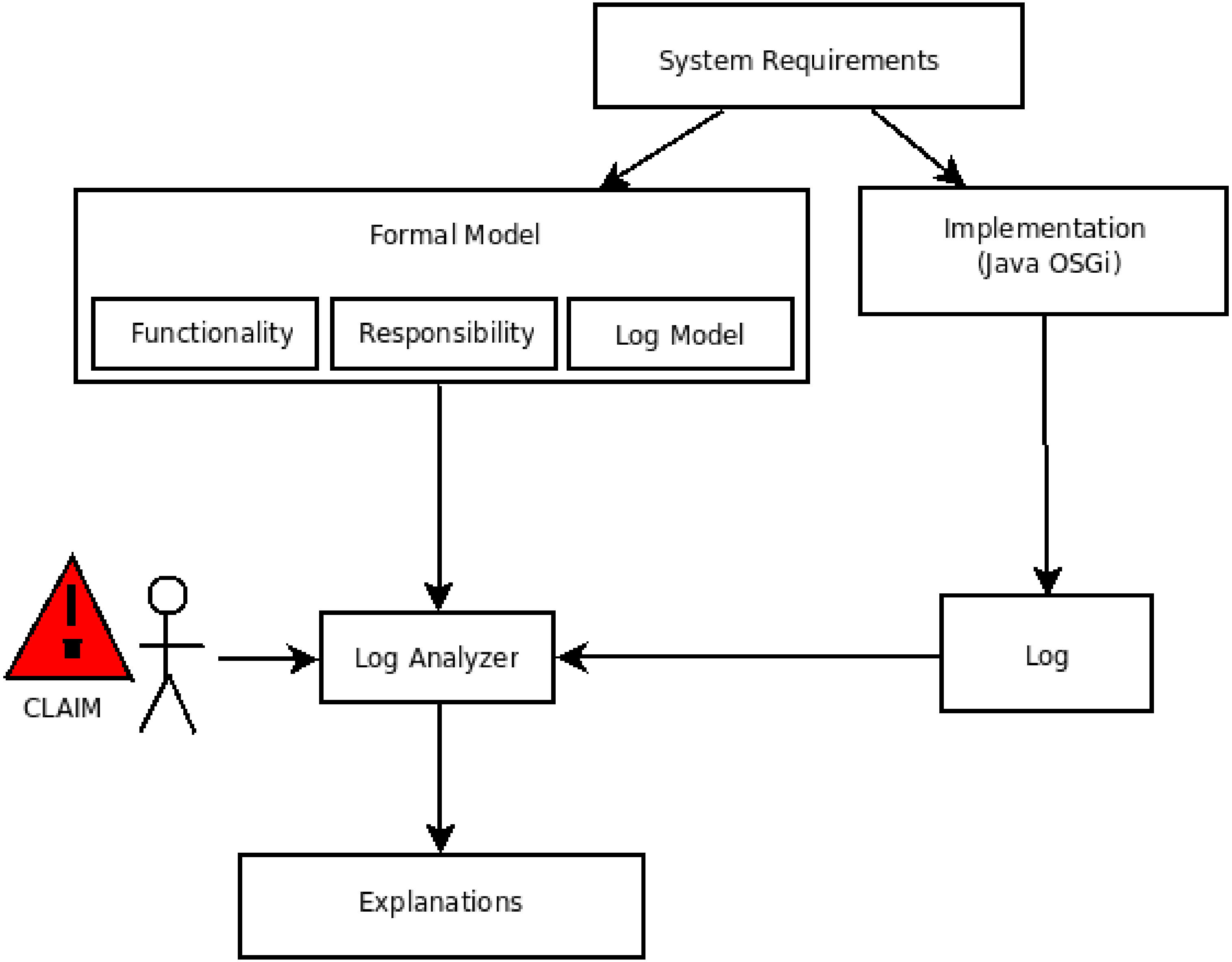
LISE: **L**iability **I**ssues in **S**oftware **E**ngineering

- Context
  - **Multidisciplinary group**
    - Lawyers and Engineers that search to produce a valid solution for legal dispute resolutions based on digital evidences
  - **Liability**
    - With system more complex is important to know who is responsible
      - Example: system that use open-source or third party components
  - **Digital evidences**
    - What can be legally used as digital evidence? How to formalize it?
  - **Contract** made between legal parts
    - The main object of LISE
    - it should contains agreements about **liability** and **digital evidences**

Specific objective for VERIMAG

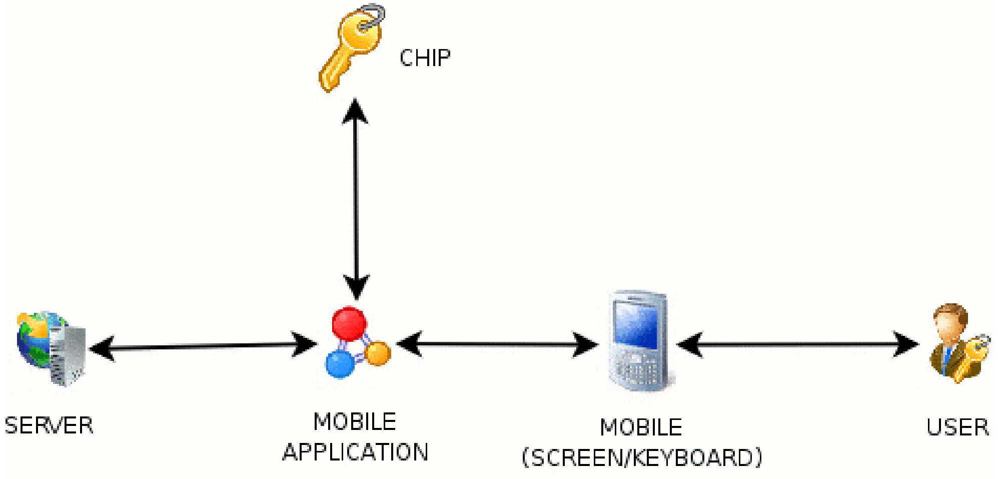
- Propose a language for formally describe the **liability** of legal parts in contracts
- Formal specification of **logs** as digital evidences
- Define a **log analyzer**, to determine the responsibility, based on the log, when an error occurs
- Approach:
  - Use of B to:
    - Define contract elements
    - Define the log analyzer
  - Creation of a tool for verification/validation of liability situations
  - The log can never be corrupted – the information registered corresponds exactly what it happened

Approach


System Requirements
Formal Model
Functionality
Responsibility
Log Model
Implementation
(Java OSGi)
CLAIM
Log Analyzer
Log
Explanations

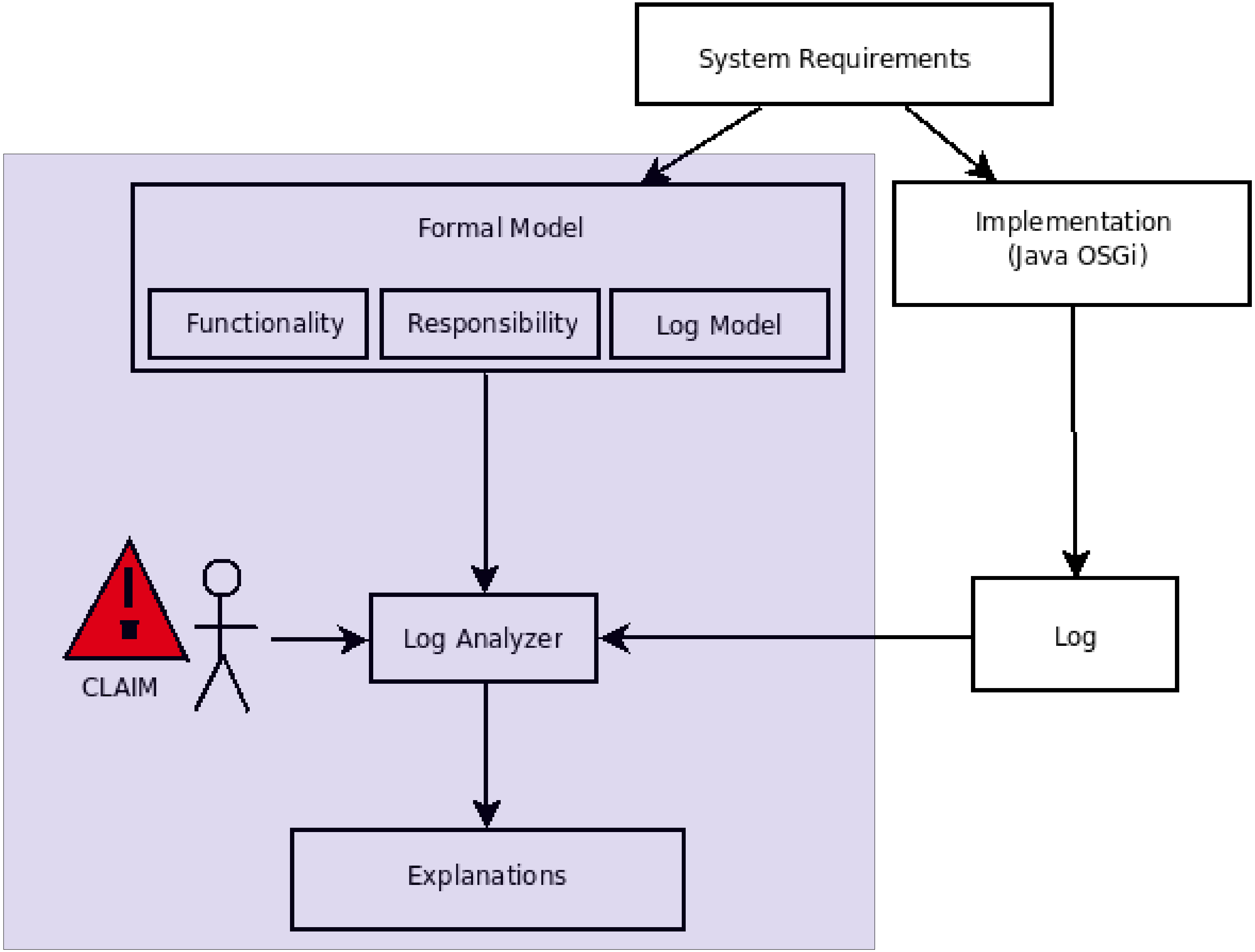
Study case

- Signature system in mobile



- Examples of problems:
  - User alleges that he has never signed any document
  - User alleges that he has signed a document different from the one in server

Approach (use of Event B)



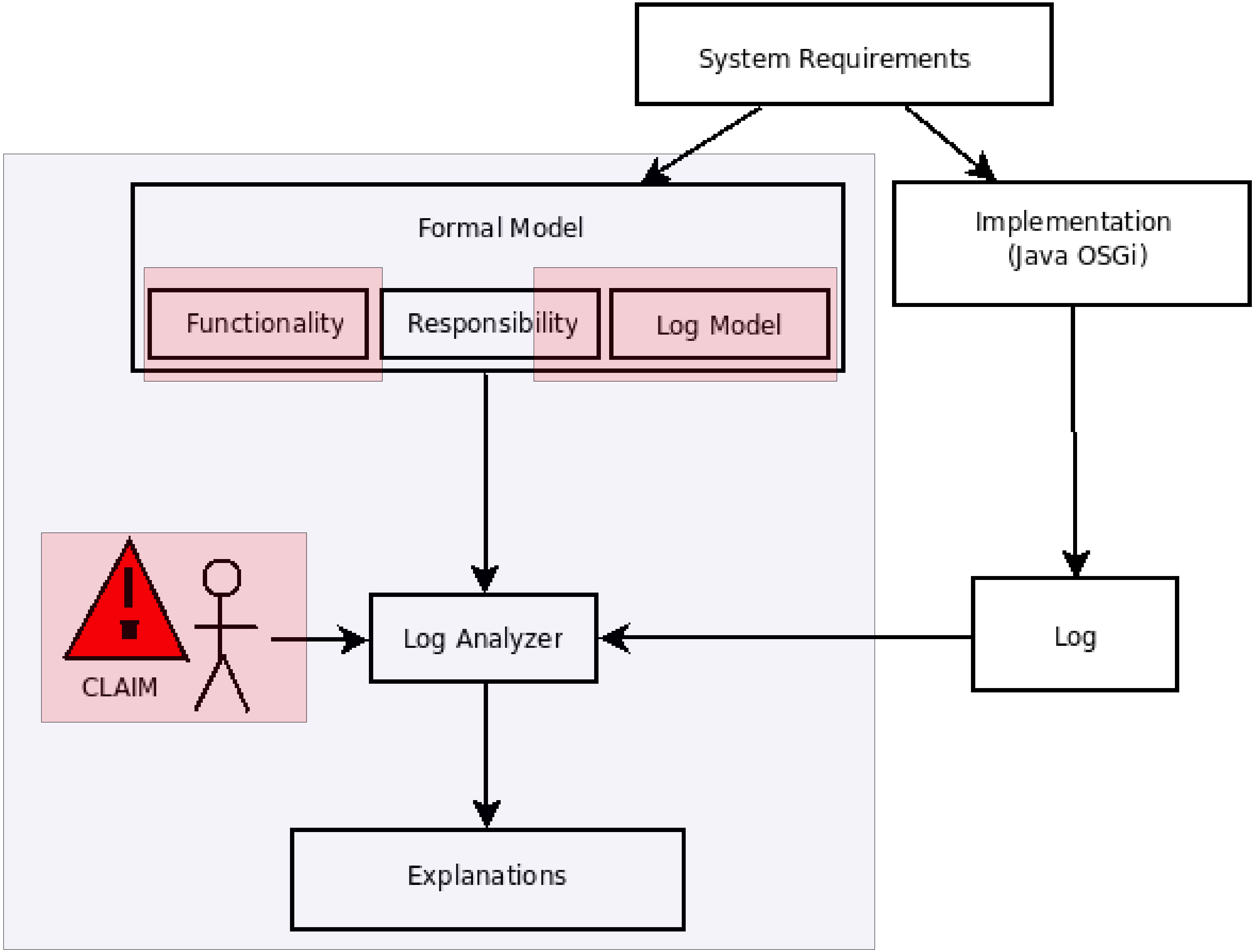
B to help define the formal model

- Formal model
  - Precise log definition and correct/incorrect behavior
  - Validation by animation
- Properties verification
  - Log accuracy with behavior
  - Responsibility function “completeness”
    - Log contain the minimum information to define responsiblity

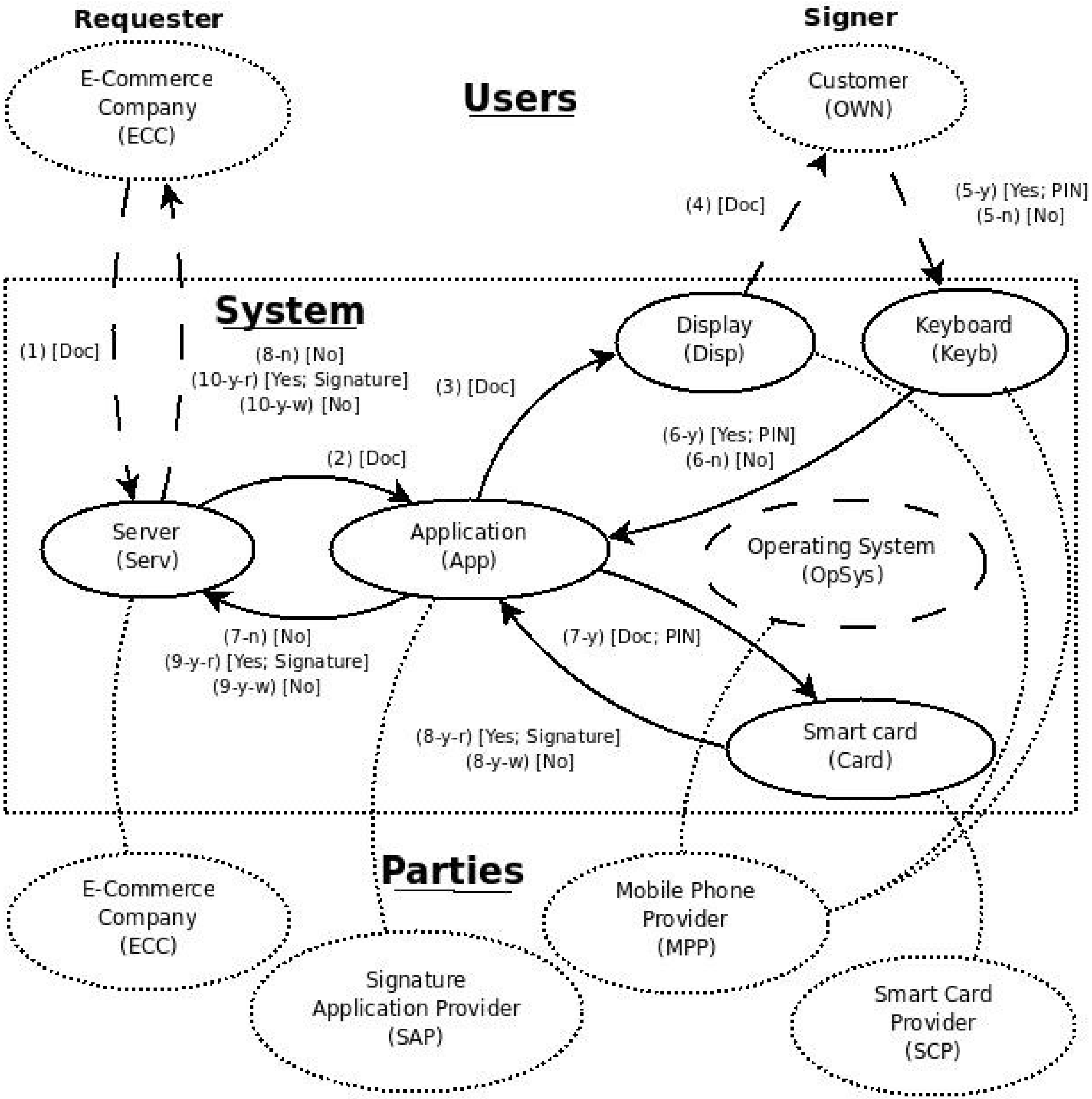
B for specifying the log analyzer

- Log analyzer: a trusted component for legal parties
  - Formal specification
  - Proved properties
  - Take as input:
    - Claim
    - Logs
- Responsibility explanation (analyzer output)
  - Who is responsible?
  - Why is it responsible?

Approach (today)



Study case schema



Entities

- Entities
  - System components (set COMP)
    - {Server, App, Card, ...)
  - Users (set USER)
    - {Costumer, ECC}
  - Legal parties (set PARTY)
    - {MPP, SAP, ...}
- Model as constants
- Liability function
  - liability: ACTOR ⇸ PARTY
  - ACTOR = COMP U USER

Logs

- Abstract log:
  - Sequence of messages with the order that they were send/received
  - One log for each actor (ACTOR)
  - Distributed log model

alog : ACTOR → seq(OP x ACTOR x ACTION x seq(PARAMETER))

  - OP = {Send, Receive}
  - ACTION = {SendDocument, …}
  - PARAMETER: represents values transmitted

Log Integrity Properties

- Additional information for log
  - AC: ACTOR → ACTION
    - What are the possible actions for each actor
- Some properties that can be verified:
  - Verifying actions execution:

    $(\text{Send}, sa, ac, pa) \in alog(ss) \Rightarrow (sa, ac) \in AC$

    $(\text{Receive}, sa, ac, pa) \in alog(ss) \Rightarrow (ss, ac) \in AC)$

  - Verifying communication errors

    $(\text{Receive}, sa, ac, pa) \in alog(ss) \Rightarrow (\text{Send}, ss, ac, pa) \in alog(sa)$

Log Functionality Properties

- We can define all possibles logs that specify the regular system executions for each actor

  Correct : (ACTOR x LOG) → BOOL

  - Function that takes as input actor and associated log and gives as output a boolean that indicates if the log belongs or not to the correct executions

- The correct behaviors are used defining the responsibility function

Log Functionality Properties

- Regular behavior can be stated as abstract log properties
  - “Every time the user receives a document it should have later a message that says if the user sign or not the document”

    (op, ss, ShowDocument, pa) ∈ alog(Display)

    ⇒ (op, ss, SendReponse, pa) ∈ alog(User)
  - “Before send the document to sign the same document should be seen by the mobile user”

    (op, ss, Sign, pa) ∈ alog(Card)

    ⇒ (op, ss, ShowDocument, pa) ∈ alog(User)

Claims

- Basis for legal disputes
  - How can we represent them using the model and avoiding ambiguity?
  - Terminology
    - The plaintiff alleges that suffered damage because of actions (or lack of actions) by a defendant
- Claim are designed for different situations (using natural language)
  - “User complains that never signed the document” (NotSigned)

$\exists$ doc, sig (

(Receive, App, Response, [doc, sig]) $\in$ alog(Server) ^

$\neg$((Receive, Display, Show, [doc]) $\in$ alog(User)

)

Liability

- Link between elements:
  - Log
  - Claim
  - Parties
- Written in the contract between the parts using natural language
  - Formalization using the log properties
- **IF** Claim = NotSigned **THEN**

```
    IF NOT Correct(App, alog(App)) THEN
        Resp = SAP
    ELSE IF NOT Correct(Card, alog(Card)) THEN
        Resp = SCP
    ELSE IF NOT Correct(Mobile, alog(Mobile)) THEN
        Resp = MPP
```

Future work

- Animation for liability situations
- Language to express properties that are easier to write and read
  - Temporal logic elements
- Log completeness for liability verification
- Analyzer specification

Conclusions

- How can formal methods be used in legal disputes
- Attempt to create properties that help to validate digital evidences (logs)
- What are the kind of properties that can be used for claims?