Liability in Software Engineering Overview of the LISE Approach and Illustration on a Case Study

Daniel Le Métayer, Manuel Maarek, Eduardo Mazza, Marie-Laure Potet, Stéphane Frénot, Valérie Viet Triem Tong, Nicolas Craipeau, Ronan Hardouin, Christophe Alleaune, Valérie-Laure Benabou, et al.

Liability in Software Engineering

Overview of the LISE* Approach and Illustration on a Case Study

Daniel Le Métayer, Manuel Maarek
LICIT
INRIA Grenoble Rhône-Alpes

Eduardo Mazza, Marie-Laure Potet
VERIMAG
University of Grenoble

Stéphane Frénot
AMAZONES
INRIA Grenoble Rhône-Alpes
INSA Lyon

Valérie Viet Triem Tong
SSIR
Supélec Rennes

Nicolas Craipeau
PrINT
University of Caen Basse-Normandie

Ronan Hardouin
DANTE
University of Versailles Saint-Quentin-en-Yvelines

ABSTRACT

LISE is a multidisciplinary project involving lawyers and computer scientists with the aim to put forward a set of methods and tools to (1) define software liability in a precise and unambiguous way and (2) establish such liability in case of incident. This paper provides an overview of the overall approach taken in the project based on a case study. The case study illustrates a situation where, in order to reduce legal uncertainties, the parties to a contract wish to include in the agreement specific clauses to define as precisely as possible the share of liabilities between them for the main types of failures of the system.

Categories and Subject Descriptors

K.5 [**Legal Aspects of Computing**]: General; K.5.m [**Legal Aspects of Computing**]: Miscellaneous—*Contracts*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*

General Terms

Legal Aspects

Keywords

Liability, Contract, Formal Methods, Specification, Defects, Legal Aspects, Evidence

*LISE (Liability Issues in Software Engineering) is a project funded by ANR (Agence Nationale de la Recherche) under the SeSur 2007 programme (ANR-07-SESU-007).
Web site: `http://licit.inrialpes.fr/lise/`.
Emails of the corresponding authors:
`{Daniel.Le-Metayer,Manuel.Maarek}@inrialpes.fr`.
Additional authors are listed in Section 7.

`http://doi.acm.org/10.1145/1806799.1806823`

1. INTRODUCTION

Software contracts usually include strong liability limitations or even exemptions of the providers for damages caused by their products. This situation does not favour the development of high quality software because the software industry does not have sufficient economical incentives to apply stringent development and verification methods. Indeed, experience shows that products tend to be higher quality and more secure when the actors in position to influence their development are also the actors bearing the liability for their defects [2, 5, 27]. The usual argument to justify this lack of liability is the fact that software products are too complex and versatile objects whose expected features (and potential defects) cannot be characterised precisely, and which thus cannot be treated as traditional (tangible) goods. Admittedly, this argument is not without any ground: it is well known that defining in an unambiguous, comprehensive and understandable way the expected behaviour of systems integrating a variety of components is quite a challenge, not to mention the use of such definition as a basis for a liability agreement. Taking up this challenge is precisely the objective of the LISE project: the project studies liability issues both from the legal and the technical points of view with the aim to put forward a formal framework to (1) define liability in a precise and unambiguous way and (2) establish such liability in case of incident.

Obviously, specifying all liabilities in a formal framework is neither possible nor desirable. Usually, the parties wish to express as precisely as possible certain aspects which are of prime importance for them and prefer to state other aspects less precisely (either because it is impossible to foresee at contracting time all the events that may occur or because they do not want to be bound by too precise commitments). Taking this requirement into account, LISE provides a set of tools and methods to be used on a need basis in the contract drafting process (as opposed to a monolithic, "all or nothing" approach).

The objective of this paper is to provide an overview of the overall approach taken in LISE. The presentation is based on a case study: an electronic signature application installed on a mobile phone. Needless to say, we do not intend to illustrate all the aspects of the approach or to describe the general framework through this case study. Our aim is rather to provide some hints on the notion of liability

considered in LISE and the potential use of the results of the project in a concrete situation.

The structure of the paper reflects the chronological ordering of the actions in a real case: Section 2 describes the starting point (IT system subject to the agreement, parties involved, informal agreement between the parties and legal context); Section 3 presents the formal definition of liabilities; Section 4 suggests how the formal definition is used to instrument the system with appropriate logging facilities and to design a log analyser. Finally, Section 5 provides a sketch of related work and Section 6 identifies avenues for further research.

2. STARTING POINT

We consider an electronic signature system allowing an e-commerce company to send a document to be signed by an individual on his mobile phone. The signature of the document is subject to the individual's approval (and authentication) and all communications and signature operations are performed trough his mobile phone. In a real situation, the activation of the signature system would be preceded by a request from the individual or by a negotiation with the e-commerce company, but we do not consider this negotiation phase here.

The mobile phone itself incorporates a smart card (for the verification of the PIN) and a signature application. We assume that the mobile phone provider, the signature application provider and the smart card provider want to execute an agreement to put such a mobile phone signature solution on the market. In order to reduce legal uncertainties, the parties wish to include in the agreement specific provisions to define as precisely as possible the share of liabilities between them for the main types of failures of the system.[1] Their intention is to use these provisions to settle liability issues in an amicable way by the application of well-defined rules. At this stage it may be the case that all the components (software and hardware) are already available and the only remaining task is their integration. It may also be the case that some or all the components still have to be developed. In general, no assumption can thus be made on the fact that software components can be designed or modified in a specific way to make the implementation of liabilities easier. The only assumptions made at this stage are:

- On the technical side: the availability of the functional architecture of the system (interfaces of the components and informal definition of their expected behaviour).
- On the business side: an informal agreement between the parties with respect to the share of liabilities.

The objective of the infrastructure described in this paper is to allow the parties to translate this informal agreement into a contract which is both valid in the legal sense and as precise as possible, in particular w.r.t. technical issues, in order to minimise legal uncertainties. In the remainder of this section, we describe the initial technical and legal situation: the IT system itself (Section 2.1), the actors involved (Section 2.2), the informal agreement between the parties signing the agreement (Section 2.3) and the legal context surrounding the agreement (Section 2.4).

[1]We do not consider infringement or any other liabilities related to intellectual property rights here.

2.1 IT System

At the start of the contractual phase, the IT system is usually defined in an informal way by its architecture: its components, their interfaces, expected behaviours and interactions. In our case study, we assume that the electronic signature system is made of the following components:

- A Server (*Serv*).
- A Signature Application (*SigApp*).
- A Smart Card (*Card*).
- A Mobile Input/Output (*IO*) component which gathers the keyboard and the display of the mobile phone (including their drivers).
- An Operating System (*OpSys*).

All the components except *Serv* are embedded in the mobile phone. In this paper, we focus on liabilities related to the mobile phone system and do not consider liabilities related to *Serv* or the communication network.[2] The only functionality of *OpSys* that we consider here is its role of medium for all communications between the mobile phone components (i.e. between *SigApp*, *Card* and *IO*).

The architecture of the system and its information flows are pictured in Figure 1. The protocol starts with the E-Commerce Company (*ECC*) requesting a signature for document *D* (message 1). The document is forwarded by *Serv* and *SigApp*, and presented to Customer (*OWN*) by *IO* (messages 2, 3 and 4). If *OWN* refuses to sign, *ECC* is informed through *IO*, *SigApp* and *Serv* (messages 5-n, 6-n, 7-n and 8-n). If *OWN* agrees, the document and the PIN code entered by *OWN* are forwarded to *Card* by *SigApp* (messages 5-y, 6-y and 7-y). Next, depending on whether *Card* authenticates the PIN code or not, the document and the signature produced by *Card* are sent to *ECC* via *SigApp* and *Serv* (messages 8-y-r, 9-y-r and 10-y-r), or *ECC* is informed via *SigApp* and *Serv* of the authentication failure (messages 8-y-w, 9-y-w and 10-y-w).

The implementation of the embedded components of the signature system, which is further detailed in Section 4.1, is based on OSGi, an interoperable environment for small devices such as home gateways, car embedded systems and mobile phones.[3]

2.2 Actors

We assume that the contract is to be executed by the three parties involved in the manufacture and distribution of the signature solution:

- The Mobile Phone Provider (*MPP*),
- The Signature Application Provider (*SAP*), and
- The Smart Card Provider (*SCP*).

The customer *OWN*, who is the owner of the mobile phone, and the E-Commerce Company *ECC* are supposed to execute different contracts with *MPP* which plays the role of mobile phone operator. We are concerned only with the

[2]These liabilities could be handled in the same way by adding the e-commerce company and telecommunication operator as additional parties.

[3]`http://www.osgi.org/`.

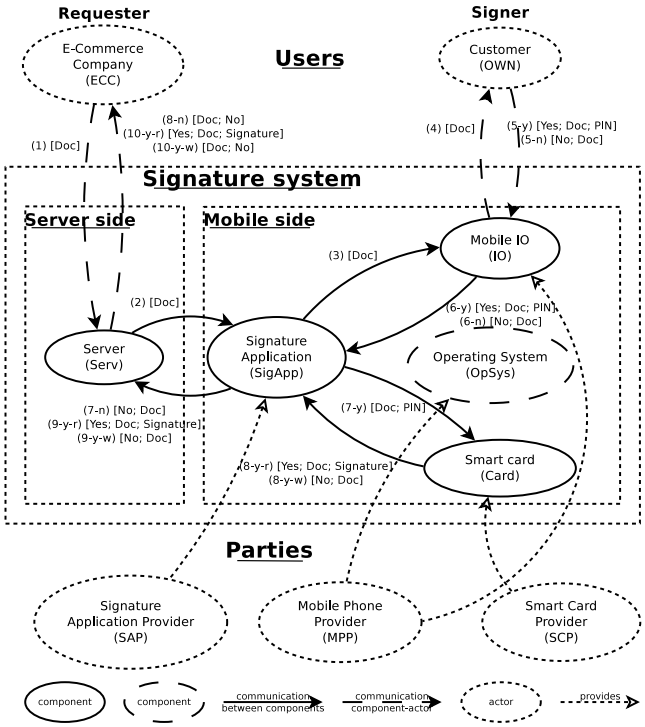

Figure 1: Communications with and within the system, and the providing party for each component.

B2B contract between *MPP*, *SAP* and *SCP* here. We come back in Section 2.4 to the legal consequences of including *OWN* among the parties (specific regulations for consumer protection). In the sequel, we shall use the word "party" for *MPP*, *SAP* and *SCP*, and the word "user" for the end-users of the system (*ECC* and *OWN*).

Each component in the system is provided by one of the parties. In our case, we assume that:

- The *SigApp* component is provided by *SAP*,
- The *Card* component is provided by *SCP*, and
- The *IO* and *OpSys* components are provided by *MPP*.

2.3 Informal Agreement

The parties wish to define as precisely as possible the share of liabilities between themselves in case of a claim from the customer *OWN*.[4] In practice, the customer will typically address his claims to *MPP* because *MPP* is the only party being in direct contact (and contractual relationship) with him (both as a mobile phone provider and operator). *MPP* will have to indemnify the customer if his claim is valid and may in turn be indemnified by one (or several) of the other parties depending on the type of the claim, the available log files and the liability share defined in the agreement.

In the following, we assume that each document to be signed is originally stamped by *ECC* and this stamp θ is (i) unique, (ii) always included in the messages of a given session and (iii) never modified. This stamp θ can be seen as a session number which makes it easier to distinguish messages pertaining to different signature sessions.

As an illustration, we consider two kinds of claims from the customer, called *DiffDoc* and *NotSigned*, concerning the signature of an alleged document D stamped θ:

(a) *DiffDoc*: the plaintiff *OWN* claims that he has been presented a document D' stamped θ different from the alleged document D (stamped θ). In the case of a purchase order, for example, D and D' may differ with respect to the quantity or price of the ordered items.

(b) *NotSigned*: the plaintiff *OWN* claims that he has never been presented any document stamped θ.

We assume that the parties agree on the following informal share of liabilities for these two types of claims:

(a) If *OWN* claims that he has been presented a document D' stamped θ different from the alleged document D (stamped θ), then

  a. *SAP* shall be liable if *SigApp* has forwarded to *OWN* a document (stamped θ) different from the document received from *ECC*.

  b. Otherwise *MPP* shall be liable.

(b) If *OWN* claims that he has never been presented any document stamped θ, then

  a. If the smart card has wrongly validated a PIN for document D stamped θ then *SCP* shall be liable.

  b. Otherwise *MPP* shall be liable.[5]

We do not discuss the value or justifications for this informal agreement here and just take it as an example of a possible share of liabilities. It should be clear that this share of liabilities is the result of a negotiation between the parties, based on a combination of technical as well as business and legal arguments, and it does not have to (and usually cannot) be justified formally. The point is that the formal framework should not impose any undue constraint on the share of liabilities but should provide means for the parties to express their wishes as precisely as possible.

2.4 Legal Context

Even though the intention of the parties is to settle liability issues in an amicable way, according to well-defined rules, it is obvioulsy necessary to take into account the legal context pursuant to computer systems. Any misconception or overlooking of the legal constraints might lead to contractual clauses that could be invalidated in court, thus increasing rather than reducing legal risks. The two main categories of legal constraints to be considered here concern the two main phases of the process: (1) the formal definition of the share of liabilities among the parties and (2) the analysis of the log files to establish these liabilities after the facts. In the following, we examine these two categories of legal constraints in turn.

[4]In this paper we assume that the claims related to the use of the signature solution come from the customer. Claims from other users such as *ECC* would be handled in the same way.

[5]Because *MPP* plays the role of mobile phone operator, he takes the risk in connection with the customer.

2.4.1 Liability Limitations

The first criterion to be taken into account to assess the validity of contractual liability limitations and exemptions is the qualification of the parties: specific protections are provided by law to consumers.[6] We first consider contracts involving only professionals. Several cases of invalidity of liability limitation[7] clauses are defined by law. The first obvious cases where the liability limitation would be considered null and void are when the party claiming the benefit of the clause has committed acts of intentional fault, wilful misrepresentation or gross negligence. Another case is the situation where the limitation would undermine an essential obligation of a party and would thus introduce an unacceptable imbalance in the contract. This situation is more difficult to assess though, and left to the appraisal of the judge who may either accept the limitation, consider it null, or even impose a different liability cap.[8]

As far as consumers are concerned, the law offers a number of protections which severely restrict the applicability of liability limitation clauses. The philosophy of these rules is that the consumer is in a weak position in the contractual relationship and legal guarantees should be provided to maintain some form of balance in the contract. For example, professionals must provide to their consumers "non conformance" and "hidden defects" warranties in French law and "implied warranty" (including "merchantability" and "fitness") in the American Uniform Commercial Code. Any clause which would introduce a significant imbalance at the prejudice of the consumer would be considered unconscionable.

Let us note that we have focused on contractual liability here (liability which is defined in the contract itself): of course, strict liability (when a defect in a product causes personal or property damages) will always apply with respect to third parties (actors who are not parties to the contract). It is still possible though for professionals to define contractual rules specifying their respective share of indemnities due to a victim (third party) by one of the parties.[9]

To conclude this subsection, let us mention other criteria that need to be taken into account to refine the legal analysis [13], in particular: the qualification of the contract itself (product or service agreement), in case of a product agreement, whether it is qualified as a purchase agreement or a license agreement, the nature of the software (dedicated or off-the-shelf software), the behaviour of the actors, etc.

[6] In French law, a party is considered as a consumer if he does not execute the agreement in the context of a professional activity, irrespective of the fact that he may have technical skills in computers.

[7] In the sequel, we use the expression "liability limitation" as a shorthand for "liability limitation or exemption".

[8] The "Faurecia case" illustrates the different interpretations of the notion of "essential contractual obligation". The final decision of the *Cour de renvoi de la Cour de cassation* (November, 26th, 2008) has invalidated the decision of the *Cour de cassation* (February 13th 2007) which had itself declared the liability limitation clause invalid. The *Cour de renvoi* has declared that the limitation of liability was not in contradiction with the essential obligation of the software provider (Oracle) because the customer (Faurecia) could get a reasonable compensation.

[9] In European laws, the victim of a defect caused by a product can sue any of the actors involved in the manufacturing or distribution of the product.

2.4.2 Log Files as Evidence

The first observation concerning the contractual use of log files is that digital evidence is now put on par with traditional written evidence. In addition, as far as legal facts are concerned (as opposed to legal acts, such as contracts), the general rule is that no constraint is imposed on the means that can be used to provide evidence. As far as legal acts are concerned, the rules depend on the amount of the transaction: for example no constraint is put on the means to establish evidence for contracts of value less than one thousand and five hundred Euros in France. The logs to be used in the context of LISE concern the behaviour of software components, which can be qualified as legal facts. Even though they would also be used to establish the existence and content of electronic contracts (as in our case study), we can consider at this stage that their value would be under the threshold imposed by law to require "written evidence" or that the evidence provided by the log files would be accepted as "written evidence" under the aforementioned equivalence principle.

A potential obstacle to the use of log files in court could be the principle according to which "no one can form for himself his own evidence". [10] It seems increasingly admitted however, that this general principle allows exceptions for evidence produced by computers [8]. As an illustration, the printed list of an airline company showing the late arrival of a traveller at the boarding desk was accepted as evidence by the French *Cour de cassation*.[11] Another condition for the validity of log files as evidence is their fairness and legality. For example, a letter or message recorded without the sender or receiver knowing it cannot be used against them.[12] As far as the LISE project is concerned, attention should be paid to the risk of recording personal data in log files: in certain cases, such recording might be judged unfair and make it impossible to use the log as evidence in court.

Generally speaking, to ensure the strength of the log based evidence provisions in the agreement, it is recommended to define precisely all the technical steps for the production of the log files, their storage and the means used to ensure their authenticity and integrity. Last but not least, as in the previous section, the cases where consumers are involved deserve specific attention with respect to evidence: any contractual clause limiting the possibilities of the consumer to defend his case by providing useful evidence is likely to be considered unconscionable in court.

2.4.3 International Law

To conclude this section, let us mention the issue of applicable law. Needless to say, the information technology business is in essence international and, even though we have focused on European laws in a first stage, more attention will be paid in the future to broaden the scope of the legal study and understand in which respect differences in laws and jurisdictions should be taken into account in the design of the LISE framework. For example, liability limitation clauses are more likely to be considered as valid by American courts which put greater emphasis on contractual freedom [17].

[10] *Nul ne peut se constituer de preuve à soi-même* in French.

[11] *Cass. civ. 1re, July 13th. 2004: Bull. civ. 2004, I, n° 207.*

[12] Similarly for phone conversation recordings.

3. FORMAL SPECIFICATION OF LIABILITIES

The share of liabilities between the parties was expressed in Section 2.3 in a traditional, informal way. Texts in natural language, even in simple "legal language", often conceal ambiguities and misleading representations. The situation is even worse when such statements refer to mechanisms which are as complex as software. Such ambiguities are sources of legal uncertainties for the parties executing the contract. The use of formal (mathematical) methods has long been studied and put into practice in the computer science community to define the specification of software systems (their expected behaviour) and to prove their correctness or to detect errors in their implementations. For various reasons however (both technical and economical), it remains difficult to apply formal methods at a large scale to prove the correctness of a complete system.

In contrast with previous work on formal methods, our goal here is not to apply them to the verification of the system itself (the mobile phone solution in our case study) but to define liabilities in case of malfunction and to build an analysis tool to establish these liabilities from the log files of the system.

It should be clear however, as stated in Section 1, that our goal is not to provide a monolithic framework in which all liabilities would have to be expressed. The method proposed in the LISE project can be used at the discretion of the parties involved and as much as necessary to express the liabilities concerning the features or potential failures deemed to represent the highest sources of risks for them.

In this section, we present successively the parameters which are used to establish liabilities (Sections 3.1 and 3.2) before introducing the liability function in Section 3.3. Let us note that, in order to make the mathematical definitions and the reasoning simpler, the notions used in this section represent an abstraction of the real system. The link between this abstract view and the real system is described in Section 4.

3.1 Trace Model

Following the informal description in Section 2, the sets of components, parties and users are defined as follows:

$$\begin{aligned} \mathit{Components} &= \{\mathit{Serv}, \mathit{SigApp}, \mathit{Card}, \mathit{IO}, \mathit{OpSys}\} \\ \mathit{Parties} &= \{\mathit{MPP}, \mathit{SAP}, \mathit{SCP}\} \\ \mathit{Users} &= \{\mathit{OWN}, \mathit{ECC}\} \end{aligned}$$

Θ is the set of stamps and $\mathcal{C}$ the set of communicating entities (components and users). $\mathcal{O}$ and $\mathcal{M}$ denote respectively the set of communication operations and message contents. The distinction between send and receive events allows us to capture communication errors.[13]

$$\mathcal{C} = \mathit{Components} \cup \mathit{Users} \qquad \mathcal{O} = \{\text{Send}, \text{Receive}\} \qquad \mathcal{M} = \mathit{List}\begin{pmatrix} \mathit{Documents} \cup \\ \{\text{Yes}, \text{No}\} \cup \\ \mathit{PINCodes} \cup \\ \mathit{Signatures} \end{pmatrix}$$

We assume that signature sessions in traces are complete and the type (document, response, PIN code, signature) of each element composing a message is implicitly associated with the element itself in order to avoid any ambiguity.

[13]This feature is not illustrated in this paper.

We denote by *Traces* the set of all traces, a trace T being defined as as a function associating a stamp with a list of items. Each item is defined by the communication operation (Send or Receive), the sender, the receiver and the content of the message:

$$T : \Theta \rightarrow \mathit{List}(\mathcal{O} \times \mathcal{C} \times \mathcal{C} \times \mathcal{M})$$

A first comment on the above definition is the fact that we use a functional type (from stamps to lists of items) to represent traces. This choice makes the manipulation of traces easier in the sequel because we are always interested in the items corresponding to a given session. Other representations could have been chosen as well, such as lists of items including the stamp information.

Note that we use the term "trace" here and keep the word "log" to denote the actual information recorded by the system. The correspondance between traces and logs is presented in Section 4.

3.2 Trace Properties

We present successively the two types of trace properties used in this paper: error properties and claim properties.

3.2.1 Error Properties

The most important parameter to determine the allocation of liabilities is the nature of the errors which can be detected in the log files of the system. Ideally, the framework should be general enough to reflect the wishes of the parties and to make it possible to explore the combinations of errors in a systematic way. One possible way to realize this exploration is to start with a specification of the key properties to be satisfied by the system and derive the cases which can lead to the negation of these properties.

Our goal being to analyse log files, we characterise the expected properties of the system directly in terms of traces (which are abstractions of logs). For example, the fact that *SigApp* should send to *IO* the document D received from *Serv* (and only this document) can be expressed as follows:[14]

$$\begin{gathered} \forall D \in \mathit{Documents}, \\ (\text{Receive}, \mathit{Serv}, \mathit{SigApp}, [D]) \in T(\theta) \;\Leftrightarrow \\ (\text{Send}, \mathit{SigApp}, \mathit{IO}, [D]) \in T(\theta) \end{gathered}$$

Note that all properties are implicitly parametrised by a trace T and a stamp θ. In the sequel these parameters are left implicit for the sake of readability.

In the scenario considered here, the systematic study[15] of the cases of violation of this property leads to the following errors:

$$\begin{gathered} \mathit{SigApp\text{-}Diff} \equiv \\ \exists D, D' \in \mathit{Documents}, D \neq D' \wedge \\ (\text{Receive}, \mathit{Serv}, \mathit{SigApp}, [D]) \in T(\theta) \wedge \\ (\text{Send}, \mathit{SigApp}, \mathit{IO}, [D']) \in T(\theta) \end{gathered}$$

[14]Note that we do not consider the ordering of Send and Receive trace items for the sake of conciseness. This ordering is not necessary to express the liabilities presented in Section 2.

[15]Space considerations prevent us from presenting this systematic derivation here. It relies on a decomposition of the negation of the properties into disjunctive normal form and selective application of additional decomposition transformations for "non existence" properties.

$$\textit{SigApp-Not} \equiv$$
$$\exists D \in \textit{Documents},$$
$$(\text{Receive}, \textit{Serv}, \textit{SigApp}, [D]) \in T(\theta) \ \wedge$$
$$\forall D' \in \textit{Documents},$$
$$(\text{Send}, \textit{SigApp}, \textit{IO}, [D']) \notin T(\theta)$$

$$\textit{SigApp-Un} \equiv$$
$$\exists D \in \textit{Documents},$$
$$(\text{Send}, \textit{SigApp}, \textit{IO}, [D]) \in T(\theta) \wedge$$
$$\forall D' \in \textit{Documents},$$
$$(\text{Receive}, \textit{Serv}, \textit{SigApp}, [D']) \notin T(\theta)$$

The terms of this disjunction correspond to three typical types of errors:

1. The first term defines a case where a message is sent with content different from expected.

2. The second term is a case of expected message which is not sent.

3. The third term is a case where an unexpected message is sent.

Similarly, the negation of the property that *Card* returns a signature only when it has received *OWN*'s PIN code P_{OWN} leads to several errors of the three aforementioned types, from which we assume that only two are deemed relevant by the parties. The first one, *Card-WrongVal* describes a case where an approval and a signature are sent by *Card* even though it has not received a right PIN code:

$$\textit{Card-WrongVal} \equiv$$
$$\exists D \in \textit{Documents}, \exists S \in \textit{Signatures},$$
$$(\text{Send}, \textit{Card}, \textit{SigApp}, [\text{Yes}; D; S]) \in T(\theta) \wedge$$
$$(\text{Receive}, \textit{SigApp}, \textit{Card}, [D; P_{OWN}]) \notin T(\theta)$$

The second one, *Card-WrongInval*, defines a case where *Card* refuses to sign a document even though it has received the correct PIN code P_{OWN}:

$$\textit{Card-WrongInval} \equiv$$
$$\exists D \in \textit{Documents},$$
$$(\text{Send}, \textit{Card}, \textit{SigApp}, [\text{No}; D]) \in T(\theta) \wedge$$
$$(\text{Receive}, \textit{SigApp}, \textit{Card}, [D; P_{OWN}]) \in T(\theta)$$

Needless to say, errors can also be defined directly based on the parties' understanding of the potential sources of failure of the system and their desire to handle specific cases. The derivation method suggested here can be used when the parties wish to take a more systematic approach to minimise the risk of missing relevant errors.

Last but not least, the language used to express properties for this case study is relatively simple as it does not account for the ordering of items in traces. In general, richer logics may be needed, for example to express temporal properties. The choice of the language of properties does not have any impact on the overall process but it may make some of the technical steps, such as the log analysis (Section 4), more or less difficult.

3.2.2 Claim Properties

Claim properties represent the "grounds for claims" of the users: they correspond to failures of the system as experienced by the users. In practice, such failures should cause damages to the user for them to give rise to liabilities but damages are left out the formal model. Claims can thus be expressed, like errors, as properties on traces. We consider two claim properties here, *DiffDoc* and *NotSigned*, which define the grounds for the claims introduced in Section 2.3:[16]

$$\textit{DiffDoc} \equiv$$
$$\exists D, D' \in \textit{Documents}, \exists S \in \textit{Signatures}, D \neq D' \wedge$$
$$(\text{Send}, \textit{SigApp}, \textit{Serv}, [\text{Yes}; D; S]) \in T(\theta) \wedge$$
$$(\text{Receive}, \textit{SigApp}, \textit{IO}, [D']) \in T(\theta)$$

$$\textit{NotSigned} \equiv$$
$$\exists D \in \textit{Documents}, \exists S \in \textit{Signatures},$$
$$(\text{Send}, \textit{SigApp}, \textit{Serv}, [\text{Yes}; D; S]) \in T(\theta) \wedge$$
$$\forall D' \in \textit{Documents}, (\text{Receive}, \textit{SigApp}, \textit{IO}, [D']) \notin T(\theta)$$

The first definition (*DiffDoc*) defines a claim corresponding to a case where *OWN* has been presented a document D' with stamp θ (as indicated by (Receive, *SigApp*, *IO*, $[D']$) different from the document D sent by the signature application to the server (message $[\text{Yes}; D; S]$). The second definition (*NotSigned*) defines a claim corresponding to a case where the signature application has sent to the server a message $[\text{Yes}; D; S]$ indicating that *OWN* has signed a document stamped θ when *OWN* has never been presented any document stamped θ.

3.3 Liability Function

The formal specification of liabilities can be defined as a function mapping a claim, a trace and a stamp onto a set of parties:[17]

$$\textit{Liability} : \textit{Claims} \times \textit{Traces} \times \Theta \to \mathbb{P}(\textit{Parties})$$

We use an intermediate function to define *Liability*: the *Check-Properties* function which returns the subset of properties (errors and claims) holding for a session identified by a trace and a stamp:

$$\begin{aligned} \textit{Check-Properties}: \ & \textit{Traces} \times \Theta \to \mathbb{P}(\textit{Properties}) \\ \textit{Errors} = \ & \left\{ \begin{array}{c} \textit{SigApp-Diff}, \textit{SigApp-Un}, \textit{SigApp-Not}, \\ \textit{Card-WrongVal}, \textit{Card-WrongInval} \end{array} \right\} \\ \textit{Claims} = \ & \{\textit{DiffDoc}, \textit{NotSigned}\} \\ \textit{Properties} = \ & \textit{Errors} \cup \textit{Claims} \end{aligned}$$

where *Properties* is the set of relevant trace properties, which, in our case study, includes the sets *Errors* defined in Section 3.2.1 and *Claims* defined in Section 3.2.2. Obviously, other error and claim properties could be useful to define other shares of liabilities. The actual implementation of *Check-Properties* (Section 4) is based on the definitions of errors and claims presented in Section 3.2.1 and Section 3.2.2.

Finally, the following function captures the share of lia-

[16] Note that, just as error properties, claim properties are implicitly parametrised by a trace T and a stamp θ.

[17] $\mathbb{P}(S)$ denotes the powerset of S.

bilities introduced in Section 2.3:

$$
\begin{array}{l}
\mathit{Liability}(C,T,\theta) = \\
\quad \mathit{If}\ C = \mathit{DiffDoc}\ \mathit{then} \\
\quad\quad \mathit{If}\ \mathit{DiffDoc} \in \mathit{Check\text{-}Properties}(T,\theta) \\
\quad\quad \mathit{Then}\quad \mathit{If}\ \mathit{SigApp\text{-}Diff} \in \mathit{Check\text{-}Properties}(T,\theta) \\
\quad\quad\quad\quad\quad \mathit{Then}\ \{\mathit{SAP}\} \\
\quad\quad\quad\quad\quad \mathit{Else}\ \{\mathit{MPP}\} \\
\quad\quad \mathit{Else}\ \emptyset \\
\quad \mathit{If}\ C = \mathit{NotSigned}\ \mathit{then} \\
\quad\quad \mathit{If}\ \mathit{NotSigned} \in \mathit{Check\text{-}Properties}(T,\theta) \\
\quad\quad \mathit{Then}\quad \mathit{If}\quad \mathit{Card\text{-}WrongVal} \in \mathit{Check\text{-}Properties}(T,\theta) \\
\quad\quad\quad\quad\quad \mathit{Then}\ \{\mathit{SCP}\} \\
\quad\quad\quad\quad\quad \mathit{Else}\ \{\mathit{MPP}\} \\
\quad\quad \mathit{Else}\ \emptyset
\end{array}
$$

The two cases in *Liability* correspond to the two types of claims considered in Section 2.3. For each type of claim, the goal of the first test is to check the validity of the claim raised by OWN. If OWN raises a claim which is not confirmed by the trace then the result of *Liability* is the empty set because no party has to be made liable for an unjustified claim. If OWN claims to have been presented a document D' different from the alleged document D and this claim is confirmed by the trace ($\mathit{DiffDoc} \in \mathit{Check\text{-}Properties}(T,\theta)$) then SAP is liable if *SigApp* has forwarded to OWN a document (stamped θ) different from the document received from ECC ($\mathit{SigApp\text{-}Diff} \in \mathit{Check\text{-}Properties}(T,\theta)$); otherwise MPP is liable. Similarly, if OWN's claim is that he has never been presented any document stamped θ and this claim is confirmed by the trace ($\mathit{NotSigned} \in \mathit{Check\text{-}Properties}(T,\theta)$) then SCP is liable if the smart card has wrongly validated a PIN in session θ ($\mathit{Card\text{-}WrongVal} \in \mathit{Check\text{-}Properties}(T,\theta)$); otherwise MPP is liable.

4. LOG ARCHITECTURE AND ANALYSER

The formal liability framework presented in the previous section is useful in itself, because it makes it possible to define liabilities in a very precise way. Its role can be enhanced however, if the actual system can be supported by facilities to record the required log files (so that experts can be sure to find all useful information after the facts) and if these log files can be analysed automatically based on the liability specifications. In this section, we sketch successively the log infrastructure and analyser for our case study.

4.1 Log Architecture

The formal setting proposed in Section 3 defines traces as lists of items corresponding to individual message exchanges between components. This choice is motivated by its generality and the possibility to model a variety of concrete implementations. As mentioned in Section 2.1, the implementation of our case study is based on OSGi, a Java based environment for the design of applications made of dynamically loadable collections of classes called bundles. Our OSGi implementation consists of three bundles implementing the following interfaces:

```
public interface SigAppIfc {
    public String submitDocToUser(String doc);
}

public interface CardIfc {
    public String sign(String doc, int pin);
}

public interface IOIfc {
    public String askToUser(String doc);
}
```

Listing 1: OSGi interfaces of the system

Each bundle provides one service (through a Java public method) which corresponds to a pair of request/answer messages in Figure 1. For example, the method `submitDocToUser` corresponds to the pair of messages 2 and 7-n, 9-y-r or 9-y-w, depending on the outcome of the transaction. The parameter of the method corresponds to the value Doc passed in message 2 and the string returned as a result is used to encode the different types of answers 7-n, 9-y-r and 9-y-w. Similarly, the method `sign` corresponds to the pair of messages 7-y and 8-y-r or 8-y-w, and the method `askToUser` to the pair 3 and 6-y or 6-n.

Table 1 shows the correspondence between the messages of Figure 1 (which are equivalent to pairs of trace items defined in Section 3) and the OSGi method calls and returns.

We have designed and developed an extension to the OSGi framework, that we call LogOS (Log Over Services), which records every service use. For each access to a service implementation, the framework generates on the fly a proxy that intercepts the call, logs the query of the call and forwards it to the implementation. LogOS is built on top of Felix,[18] an open-source implementation of the OSGi specifications. The source code of LogOS, the LogOS patch for Felix, and the implementation of the components presented in this document are available on the INRIA forge.[19]

4.2 Analyser

The *Liability* function was expressed in terms of traces in Section 3. Traces were defined in Subsection 3.1 as functions mapping stamps to sequences of items, thus at a more abstract level than the logs effectively recorded by LogOS. The goal of the analyser is to implement the *Liability* function specification of Section 3.3 based on the effective LogOS log file format and the correspondence between traces and log files. The four main phases of the analysis are the following:

1. The first phase is the identification of the part of the log files which is relevant for the analysis. This part obviously depends on the claim. In our case study, the stamp θ can be used to identify the relevant signature session and thus to extract the useful parts of the log. This phase of the analyser is made simpler here because of the assumptions of stamps uniqueness and integrity. Identifying a given session can be a more complex task in general.

2. The second phase consists in transforming the relevant parts of the logs into a trace. Table 1 describes this transformation stage for our case study. For example, let us consider a LogOS log file item recording the call by the `SigApp` bundle to the `askToUser` service of the `IO` bundle for a document *doc*1 and stamp 480031:[20]

```
480031;2009-05-05;08:34:12;SigApp;IO;IOImpl;
    IOIfc;askToUser;doc1
    ;2009-05-05;08:34:23;"1234"
```

[18] `http://felix.apache.org/`

[19] `http://gforge.inria.fr/frs/?group_id=2014`

[20] Note that LogOS log file items include more information than traces, such as the times and dates.

Generic trace items: Components (sender → receiver)	Generic trace items: Message type exchanged	Message number according to Figure 1	Corresponding method (call or return)
Serv → *SigApp*	[Doc]	2	call to `submitDocToUser`
SigApp → *Serv*	[No; Doc]	7-n	return from `submitDocToUser`
SigApp → *Serv*	[Yes; Doc; Signature]	9-y-r	return from `submitDocToUser`
SigApp → *Serv*	[No; Doc]	9-y-w	return from `submitDocToUser`
SigApp → *IO*	[Doc]	3	call to `askToUser`
IO → *SigApp*	[Yes; Doc; PIN]	6-y	return from `askToUser`
IO → *SigApp*	[No; Doc]	6-n	return from `askToUser`
SigApp → *Card*	[Doc; PIN]	7-y	call to `sign`
Card → *SigApp*	[Yes; Doc; Signature]	8-y-r	return from `sign`
Card → *SigApp*	[No; Doc]	8-y-w	return from `sign`

Table 1: Relation between trace items, messages and methods

Following Table 1, this log item can be transformed into the following trace items for stamp 480031:[21]

$$\left[\begin{array}{l}(\text{Send}, \mathit{SigApp}, \mathit{IO}, [\mathit{doc1}]);\\ (\text{Receive}, \mathit{SigApp}, \mathit{IO}, [\mathit{doc1}])\\ (\text{Send}, \mathit{IO}, \mathit{SigApp}, [\mathit{Yes}; \mathit{doc1}; 1234]);\\ (\text{Receive}, \mathit{IO}, \mathit{SigApp}, [\mathit{Yes}; \mathit{doc1}; 1234])\end{array}\right]$$

3. The third phase is the application of *Check-Properties* to check the validity of the claim and find the errors occurring in the trace. Again, this phase is made easier by the fact that the properties needed to express the share of liabilities considered here are fairly simple. All these properties (*Errors* in Section 3.2.1 and *Claims* in Section 3.2.2) are expressed as the presence or absence of certain items in the trace, which can be implemented by systematic searches in the trace constructed from the log files.[22]

4. The fourth phase is the exploitation of the result of the *Check-Properties* function to compute *Liability*. This phase is straightforward since it amounts to the implementation of the simple tests specified in the definition of *Liability* in Section 3.3.

The simple four phases structure sketched here obviously needs to be optimized in order to avoid the search for all properties (claims and errors) in the trace: only the claim raised by *OWN* and the errors useful for this claim[23] need to be searched in the trace. *Check-Properties* is thus not implemented as such, but split into specific functions searching for specific properties.

5. RELATED WORK

The significance of liability, warranty and accountability and their potential impact on software quality have already been emphasized by computer scientists as well as lawyers ([2, 5, 27, 28]). However we are not aware of previous work on the application of formal methods to the definition of software liability. Earlier work on the specification of contracts mostly deal with obligations in a general sense ([9, 11, 24]), with specific types of contracts such as commercial contracts or privacy rules ([2, 16, 22]) or with the responsibility of agents in multi-agent systems ([12]) but do not address liabilities related to software errors. It should be clear however that several connected areas share part of our objectives and provide useful hints and results:

- Software dependability [4, 18, 23] is also concerned with failure analysis (using, for example, fault trees or FMECA analysis processes) but focuses on fault prevention, tolerance and removal rather than on the specification of liabilities.

- Model based diagnosis ([7, 19, 23, 35]) provides techniques for fault analysis and diagnosability based on observability properties. Diagnosis can be carried out either off-line or on-line[24] with different cost and time constraints. Faults are generally represented as single events rather than as logical properties in our approach. Again, the objective of model based diagnosis is to detect faults and analyze them in order to take appropriate measures rather than to determine the liable parties after a failure has occurred and damages have been caused.

- Intrusion detection ([14]) systems also aim at detecting unexpected behaviours but they are targeted towards security attacks rather than faults. They are generally classified into two categories: the *anomaly detection* and the *misuse detection* approaches, the first one being based on a model of the correct behaviour of the system and the second one on typical attack patterns. As stated in the next section, our framework can accomodate both negative properties (as shown in this paper) and positive properties (correct behaviours). In contrast with intrusion detection however, we do not have any "real time" constraint here and accuracy is far more significant than efficiency for liability analysis.

- Forensics ([3, 21, 26]) and digital evidence ([10, 32, 33, 34]) share with LISE the objective to analyse digital information in a legal setting. However the contributions in these areas are generally targeted towards security attacks or computer crime investigations rather than the identification of liable parties in a software

[21] As stated above, the duplication of send and receive events is useful in general to detect communication errors. This feature is not illustrated here though, because communication errors have not been used to define the share of liabilities.

[22] Note that, in contrast with the use of formal modals for program verification, we only need to check these properties on a given trace here, rather than for all possible execution traces, and this trace is obviously finite.

[23] For example, only the error *SigApp-Diff* is useful to compute *Liability* for the claim *DiffDoc*.

[24] "Off-board"or "on-board" for embedded systems.

contract. Technically speaking, a significant impact is the fact that, in our setting, the search in the logs is driven by pre-defined properties (errors and claims).

- Service Level Agreements also define contractual provisions but generally focus on Quality of Service rather than functional requirements and do not put emphasis on formal specifications. A notable exception is the SLAng language [30] which is endowed with a formal semantics and can be used to specify a variety of services such as Application Service Provision, Internet Service Provision or Storage Service Provision. In addition, the monitorability and monitoring of SLAng services have been considered both from the formal and practical point of view [31, 25].

Needless to say, each of the above areas are useful sources of inspiration for the LISE project, but we believe that none of them, because of their different objectives, provides the answer to the key problem addressed in this paper, namely the formal specification and instrumentation of liability.

6. CONCLUSION

First, we should stress that the set of methods and tools provided by the LISE framework can be used in an incremental way, depending on the wishes of the parties, the economic stakes and the timing constraints for drafting the contract:

1. The first level is a systematic (but informal) definition of liabilities in the style of Section 2.3.
2. The second level is the formal definition of liabilities as presented in Section 3.3. This formal definition itself can be more or less detailed and encompass only a part of the liability rules defined informally. In addition, it does not require a complete specification of the software but only the properties relevant for the targeted liability rules.
3. The third level is the implementation of a log infrastructure (as shown in Section 4.1) or the enhancement of existing logging facilities to ensure that all the information required to establish liabilities will be available if a claim is raised. Another option is to check that it will be possible to extract the required information from regular files if needed.
4. The fourth level is the implementation of a log analyser (as shown in Section 4.2) to assist human experts in the otherwise tedious and error-prone log inspection.
5. A fifth level, not presented here, would be the verification of the correctness of the log analyser with respect to the formal definition of liabilities (considering the correspondence between log files and traces). This level would bring an additional guarantee about the validity of the results produced by the system.

Each of these levels contributes to reducing further the uncertainties with respect to liabilities and the parties can decide to choose the level commensurate with the risks involved with potential failures of the system.

The notions of trace and property have been presented in a somewhat simplified way in this paper. It may be the case that not all the relevant information is included in the log files of the system. For example, in our case study, the fact that the customer *OWN* has declared the theft of his mobile phone or has signed an acknowledgement receipt for a product sent by the E-Commerce Company can be useful information to analyse the situation (depending on the liability rules decided by the parties). Traces can thus be more than abstract versions of the log files and include other types of actions from all the actors involved. Also, we have defined *Properties* as $Errors \cup Claims$ here. In general, it can be useful to use other types of properties to define liabilities (for example the fact that *OWN* has answered "yes" to the signature request sent by *SigApp* here). These properties can be included into *Properties* without any impact on the rest of the process.

As far as the methodology is concerned, we are working on an iterative process for the elaboration of the formal specification of liabilities involving interactions with the parties to discover oversights or missing errors and to take into account logging constraints. Logging constraints are typically related to implementation issues (e.g. performance penalties or log distribution), security requirements or privacy issues ([1]). For example, in our case study, logging critical data such as PIN codes outside the smart card would not be acceptable, thus requiring an iteration step to define an equivalent (or approximated) definition of liability that could be reflected in the logs. In addition it is necessary to take into account observability issues: previous work on model based diagnosis and diagnosability ([7, 19, 23, 35]) will prove useful to this respect.

In general, liabilities will be expressed by a combination of informal and formal means, both of which being integrated in the legal agreement. We are working on a framework allowing the parties to feed their contract with clauses automatically generated from the formal specifications of liability with seamless integration with the rest of the contract. The final objective is to allow contract drafters to manipulate statements either in natural language or in formal language while maintaining the links between the two parts and the consistency of the whole document. This extension is based on our previous work on the links between mathematical texts and natural language explanations ([15]).

In terms of implementation of the log infrastructure, one distinguishing feature of the case study considered here is the possibility to manage the log in a centralised way. Obviously, this may not be the case in many situations, which will add the extra difficulty to define correspondences (or causality relationships) between items in different logs ([6, 35]). Last but not least, we are currently working on two other key issues related to log files which have not been discussed here: their optimisation in terms of storage (compaction, retention delay, etc.) using an index-based factorization method and techniques to ensure their authenticity and integrity ([1, 20, 29]) including trusted serialization of log items.

7. ADDITIONAL AUTHORS

Christophe Alleaume (PrINT — University of Caen Basse-Normandie), Valérie-Laure Benabou (DANTE — University of Versailles Saint-Quentin-en-Yvelines), Denis Beras (AMAZONES — INRIA Grenoble Rhône-Alpes, INSA Lyon), Christophe Bidan (SSIR — Supélec Rennes), Gregor Goessler (POPART — INRIA Grenoble Rhône-Alpes), Julien Le Clainche (LICIT — INRIA Grenoble Rhône-Alpes), Ludovic Mé (SSIR — Supélec Rennes), and Sylvain Steer (DANTE — University of Versailles Saint-Quentin-en-Yvelines).

8. REFERENCES

[1] R. Accorsi. On the relationship of privacy and secure remote logging in dynamic systems. In *SEC*, volume 201 of *IFIP*, pages 329–339. Springer, 2006.

[2] R. Anderson and T. Moore. Information security economics — and beyond. Information Security Summit (IS2), 2009.

[3] A. R. Arasteh, M. Debbabi, A. Sakha, and M. Saleh. Analyzing multiple logs for forensic evidence. *Digital Investigation*, 4:82–91, 2007.

[4] A. Avizienis, J.-C. Laprie and B. Randell. Fundamental concepts of computer system dependability. In *IARP/IEEE-RAS Workshop on robot dependability: technological challenges of dependable robots in human environments*, 2001.

[5] D. M. Berry. Abstract appliances and software: The importance of the buyer's warranty and the developer's liability in promoting the use of systematic quality assurance and formal methods. CiteSeerX, `http://www.scientificcommons.org/42749418`, 2007.

[6] D. Biswas, T. Gazagnaire, and B. Genest. Small logs for transactional services: Distinction is much more accurate than (positive) discrimination. In *High Assurance Systems Engineering Symposium (HASE)*, 2008.

[7] L. Brandan-Briones, A. Lazovik, and P. Dague. Optimal observability for diagnosability. In *International Workshop on Principles of Diagnosis*, 2008.

[8] N. Craipeau. Digital evidence – La preuve électronique. TR, Deliverable LISE D1.3, July 2009.

[9] A. D. H. Farrell, M. J. Sergot, M. Sallé, and C. Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems (IJCIS)*, 14(2-3):99–129, 2005.

[10] P. Gladyshev and A. Enbacka. Rigorous development of automated inconsistency checks for digital evidence using the B method. *International Journal of Digital Evidence (IJDE)*, 6(2):1–21, 2007.

[11] G. Governatori, Z. Milosevic, and S. W. Sadiq. Compliance checking between business processes and business contracts. In *EDOC*, pages 221–232. IEEE Computer Society, 2006.

[12] D. Grossi, L. Royakkers, and F. Dignum. Organizational structure and responsibility. *Artificial Intelligence and Law*, 15:223–249, 2007.

[13] R. Hardouin. Liability in software contracts – Le sens des responsabilitiés en matière de contrats informatiques. TR, Deliverable LISE D1.1, July 2009.

[14] A. K. Jones and R. S. Sielken. Computer system intrusion detection: a survey. TR, University of Virginia Computer Science Department, 1999.

[15] F. Kamareddine, M. Maarek, and J. B. Wells. Flexible encoding of mathematics on the computer. In *MKM*, volume 3119 of *LNCS*, pages 160–174, 2004.

[16] D. Le Métayer. A formal privacy management framework. In *FAST*, volume 5491 of *LNCS*, pages 162–176, 2009.

[17] S. Lipovetsky. Les clauses limitatives de responsabilité et de garantie dans les contrats informatiques. Approche comparative France/États-Unis. Quelles limitations. *Expertises des systèmes d'information*, n° 237, pages 143–148, May 2000.

[18] B. Littlewood and L. Strigini. Software reliability and dependability: a roadmap. In *ICSE - Future of SE Track*, pages 175–188, ACM, 2000.

[19] Y. Papadopoulos. Model-based system monitoring and diagnosis of failures using statecharts and fault trees. *Reliability Engineering and System Safety*, 81:325–341, 2003.

[20] P. Parrend and S. Frénot. Security benchmarks of OSGi platforms: toward hardened OSGi. *Software - Practice and Experience (SPE)*, 39(5):471–499, 2009.

[21] S. P. Peisert, S. Karin, M. Bishop, and K. Marzullo. Principles-driven forensic analysis. In *Workshop on New security paradigms (NSPW)*, pages 85–93, 2005.

[22] S. L. Peyton Jones and J.-M. Eber. How to write a financial contract. In *The Fun of Programming*, Cornerstones of Computing, chapter 6. 2003.

[23] C. Picardi, R. Bray, F. Cascio, L. Console., P. Dague, O. Dressler, D. Millet, B. Rhefus., P. Struss and C. Vallée. IDD: integrating diagnosis in the design of automotive systems. In *European Conference on Artificial Intelligence (ECAI)*, pages 628–632, 2002.

[24] C. Prisacariu and G. Schneider. A formal language for electronic contracts. In *FMOODS*, volume 4468 of *LNCS*, pages 174–189. Springer, 2007.

[25] F. Raimondi, J. Skene, and W. Emmerich. Efficient online monitoring of web-service SLAs. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 170–180, 2008.

[26] S. Rekhis and N. Boudriga. A temporal logic-based model for forensic investigation in networked system security. *Computer Network Security*, 3685:325–338, 2005.

[27] D. J. Ryan. Two views on security and software liability. Let the legal system decide. *IEEE Security and Privacy*, January-February, 2003.

[28] F. B. Schneider. Accountability for perfection. *IEEE Security and Privacy*, March-April, 2009.

[29] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.

[30] J. Skene, D. D. Lamanna, and W. Emmerich. Precise service level agreements. In *ICSE*, pages 179–188. IEEE, 2004.

[31] J. Skene, A. Skene, J. Crampton, and W. Emmerich. The monitorability of service-level agreements for application-service provision. In *International Workshop on Software and Performance (WOSP)*, pages 3–14. ACM, 2007.

[32] M. Solon and P. Harper. Preparing evidence for court. *Digital Investigation*, 1:279–283, 2004.

[33] P. Stephenson. Modeling of post-incident root cause analysis. *Digital Evidence*, 2(2), 2003.

[34] R. E. K. Stirewalt, L. K. Dillon, and E. Kraemer. The inference validity problem in legal discovery. In *ICSE Companion*, pages 303–306. IEEE, 2009.

[35] S. Yang, L. Hélouët, and T. Gazagnaire. Logic-based diagnosis for distributed systems. In *Perspectives in Concurrency Theory: A Festschrift for P. S. Thiagarajan*. CRC Press, 2009.