Apport des méthodes formelles pour l'exploitation de logs informatiques dans un contexte contractuel

Lacramioara Astefanoaei[1], Gregor Gössler[1], Daniel Le Métayer[1],
Eduardo Mazza[1,2], Marie Laure Potet[2], Valerie Viet Triem Tong[3]
[1] INRIA Grenoble – Rhône-Alpes,
[2] Vérimag Gières
[3] SUPELEC Rennes
{Lacramioara.Astefanoaei, Gregor.Goessler, Daniel.Le-Métayer}@inrialpes.fr,
{Marie-Laure.Potet, Eduardo.Mazza}@imag.fr
valerie.viettriemtong@supelec.fr

Résumé. Dans cet article, nous présentons la démarche adoptée dans le projet LISE pour spécifier de manière formelle les responsabilités des parties dans un contrat portant sur des logiciels. Nous décrivons deux options, l'une reposant sur une attribution *a priori* des responsabilités en fonction des dysfonctionnements constatés dans les logs, l'autre sur une analyse de causalité, et nous les illustrons sur un exemple de système de réservation d'hôtels.

Mots clés : responsabilité juridique, contrat, logiciel, plainte.

1 Introduction

Les logiciels sont aujourd'hui des produits "grand public" largement distribués et même indispensables pour une grande partie de nos activités aussi bien personnelles que professionnelles. En dépit de cette large diffusion, la question des responsabilités juridiques en matière de logiciels reste difficile à appréhender par le droit : les distributeurs imposent souvent des dispositions très générales qui les exonèrent de toute responsabilité ou qui limitent très strictement la portée de celles-ci. Mais les clauses limitatives de responsabilité peuvent dans un certain nombre de cas être invalidées par les juges en cas de litige. Ces invalidations peuvent être prononcées sur différents fondements (protection des consommateurs, non respect d'une obligation essentielle du contrat, produits défectueux, faute lourde, etc.), certains d'entre eux donnant lieu à une jurisprudence hésitante [1]. Ces incertitudes sont des sources de risques pour les parties prenantes qui cherchent parfois à les circonscrire en tentant de définir plus précisément la répartition de leurs responsabilités dans le contrat lui-même. Mais une telle définition n'est pas aisée, notamment pour les responsabilités portant sur les dysfonctionnements de logiciels. C'est à ce défi que s'est attaqué le projet LISE [2] en adoptant une démarche reposant sur l'usage de méthodes formelles. L'objectif du projet était double : il s'agissait d'une part de définir précisément les responsabilités liées à des défaillances de logiciels et d'autre part d'offrir les moyens d'établir ces responsabilités après les faits.

1. On pourra notamment se référer à deux grandes sagas judiciaires récentes : l'affaire dite Oracle contre Faurecia et la décision dite Chronopost [31].

2. LISE (Liability Issues in Software Engineering) est un projet pluridisciplinaire financé par l'ANR (Agence Nationale de la Recherche) dans le cadre du programme SeSur (ANR-07-SESU-007).

L'importance de la notion de responsabilité et son impact potentiel sur la qualité des logiciels ont déjà été soulignés par de nombreux auteurs, aussi bien informaticiens que juristes ([1, 4, 25, 26]) mais, à notre connaissance, la démarche adoptée dans le projet LISE est nouvelle. La plupart des travaux antérieurs sur la spécification des contrats traitent des obligations dans un sens général ([7, 10, 22]) ou pour des types de contrats spécifiques tels que des contrats financiers ou des règles de confidentialité ([15, 21]) ; elles ne traitent pas des responsabilités liées à des erreurs de logiciels. Les travaux sur les SLA (Service Level Agreements) traitent également des dispositions contractuelles, mais se concentrent généralement sur la qualité de service plutôt que sur des exigences fonctionnelles. De plus, la plupart ne reposent pas sur des spécifications formelles. Une exception notable est le langage Slang [27, 28] qui est doté d'une sémantique formelle et peut être utilisé pour spécifier une variété de services tels que des services Internet ou de stockage. En outre, le monitoring des services exprimés en Slang a été étudié à la fois du point de vue théorique et de manière pratique, à travers un environnement dédié [29].

D'autres domaines de recherche partagent certains de nos objectifs, comme la fiabilité des logiciels [3], le diagnostic ([5, 20]), la détection d'intrusion ([12]), la forensique ([2, 23]) et les preuves numériques ([8, 30, 32]). Ces domaines ont constitué des sources d'inspiration utiles pour le projet LISE, mais aucun d'entre eux ne fournit la réponse au problème principal abordé dans ce document, à savoir la spécification formelle et l'établissement des responsabilités.

La démarche générale du projet LISE est présentée dans [16] et [17]. Dans [19] nous avons proposé une formalisation, basée sur la méthode B, permettant de spécifier le contenu des logs, leur distribution et une procédure d'analyse de ces logs. La spécification proposée permet d'exprimer précisément des faits préalablement répertoriés (dans le contrat) et correspondant à des dysfonctionnements (plaintes) ou à des erreurs. Dans [18] nous avons proposé des critères permettant de caractériser la capacité d'une architecture de logs (définie comme la spécification des données enregistrées dans les logs, leur format et les acteurs en charge de réaliser ces enregistrements) à fournir des informations fiables pour répondre à un ensemble donné de plaintes. Une architecture satisfaisant ces critères procure des informations qui peuvent être utilisées avec confiance par les acteurs impliqués dans un litige. Enfin, [31] précise le cadre juridique dans lequel s'inscrit la démarche, sous la forme de clauses génériques et d'annexes techniques à instancier, et analyse les contraintes juridiques à respecter pour assurer la recevabilité des preuves.

La méthode d'attribution des responsabilités définie dans [16], dite "méthode par tableaux", repose sur une fonction prédéfinie (supposée résulter d'un accord entre les parties) qui permet d'associer automatiquement certaines erreurs (ou combinaisons d'erreurs) détectées dans les logs aux responsabilités de parties désignées. Le travail presenté ici développe une méthode alternative d'attribution des responsabilités reposant sur la notion de causalité introduite dans [9] en lien avec les notions de causalité en droit.

La section 2 présente la démarche générale de LISE. Dans la section 3 nous introduisons le cas d'étude qui motivera la suite des développements et permettra d'illustrer les deux approches présentées. La première méthode de spécification des responsabilités, par tableaux, est décrite dans la section 4 et la seconde, par analyse de causalité, dans la section 5. Enfin la section 6 compare les deux méthodes sur la base d'un scénario fautif décrit dans la section 3.

2 Démarche générale

Nous nous plaçons dans le cadre de contrats relatifs au développement ou la distribution de logiciels et nous nous intéressons exclusivement aux responsabilités liées aux dysfonctionnements du système. Nous ne considérons donc pas, par exemple, d'éventuels litiges résultant de retards de fournitures ou des

contestations portant sur la propriété intellectuelle (contrefaçon). La cible visée est la mise en place de solutions logicielles pour lesquelles des exigences fortes de sûreté de fonctionnement sont attendues (par exemple des systèmes critiques avec des contraintes de sûreté ou de sécurité).

L'objectif est donc de concevoir des solutions techniques (1) permettant une attribution précise des responsabilités entre les parties contractantes et (2) valides d'un point de vue juridique. La démarche proposée est la suivante :

1. Intégration dans le contrat de dispositions portant sur les preuves électroniques (conventions de preuve) qui seront utilisées en cas de litige.
2. Définition précise des responsabilités pour des cas de dysfonctionnements identifiés au préalable.
3. Définition d'une procédure précise, acceptée par les parties, et ne laissant aucun doute sur l'attribution des responsabilités pour les dysfonctionnements identifiés.

Le choix a été fait de constituer les conventions de preuve sous la forme d'un système de logs que les parties s'engagent à produire et protéger. La démarche proposée permet de faciliter les règlements à l'amiable et, le cas échéant, de produire des éléments de preuve convaincants pour le juge. Elle se distingue des techniques d'investigation numérique (forensique) qui s'appliquent *a posteriori* : puisque nous nous plaçons dans un cadre contractuel, il est possible (et souhaitable) d'adopter une approche *a priori* qui permet de prévoir les informations utiles pour déterminer les responsabilités et les conditions de leur enregistrement et de leur analyse. Elle permet également d'anticiper sur le principe de droit selon lequel " null ne peut se constituer sa preuve à soi-même" [6] en distribuant les logs ou en introduisant des tiers de confiance.

L'utilisation d'un langage formel pour décrire les logs, leur gestion, leur analyse, ainsi que l'attribution de responsabilités offre plusieurs avantages :

- Elle permet de lever les ambiguïtés propres à la langue naturelle (et même juridique).
- Le fait de disposer, dès le contrat, d'une modélisation des logs et de l'analyseur de logs permet d'effectuer des simulations. Il est également essentiel de pouvoir établir la correction de l'analyseur de logs.

Nous avons utilisé la méthode B qui répond bien à ces critères. Son langage de modélisation utilise des concepts connus. Elle inclut une notion de raffinement jusqu'au code effectif. De plus, elle offre de nombreux outils, en particulier pour l'animation.

La modélisation attendue dans l'approche LISE peut être décomposée en quatre étapes principales :

1. Spécification des composants et de leurs interfaces
2. Définition des logs comme sous-ensembles des échanges entre composants.
3. À partir des propriétés importantes du système, identification d'un ensemble de plaintes correspondant à des dommages importants qui pourraient être causés par le système.
4. Pour chaque type de plainte, définition des responsabilités en fonction des erreurs découvertes dans les logs.

La spécification des composants peut être plus ou moins détaillée, en fonction de l'effort que les différentes parties sont prêtes à fournir. Le fait de lister *a priori* les dysfonctionnements pour lesquels des responsabilités seront engagées permet de limiter les responsabilités à des dysfonctionnements entraînant de réels préjudices.

La définition des responsabilités peut être effectuée de deux manières : soit en établissant une relation pré-définie entre erreurs et responsabilités, soit en ayant recours à une notion plus générale de "causalité logique". Ces deux possibilités sont étudiées dans les sections 4 et 5 après la présentation du cas d'étude considéré dans la section suivante.

3 Cas d'étude

Nous considérons un système de réservation de chambres d'hôtel comportant deux types de composants :

- Un composant $Agency$, gérant pour une agence de voyage les transactions avec les clients et les prestataires de service.
- Un composant $HotelId$ par hôtel Id impliqué.

Pour simplifier la présentation, nous nous limitons à une seule agence de réservation et nous omettons d'autres acteurs essentiels (comme les banques) qui pourraient être traités selon les mêmes principes. Les clients sont aussi partie prenante mais ne sont pas associés à des composants pour lesquels des responsabilités sont définies : en d'autres termes, les responsabilités considérées ici relèvent d'un contrat B2B entre l'agence de réservation et les hôtels.

Lors d'une demande de réservation, un numéro de session (sid) est attribué à la transaction. A partir des informations fournies par le client ($details$), l'agence interroge successivement les hôtels afin d'obtenir une réservation (réponse YES/NO de la part des hôtels). Un hôtel honorant une demande doit d'une part enregistrer la réservation et en informer l'agence et, d'autre part, effectuer la demande de débit et en informer le client.

La figure 1 décrit un scénario dans lequel l'agence doit consulter deux hôtels avant d'obtenir satisfaction.

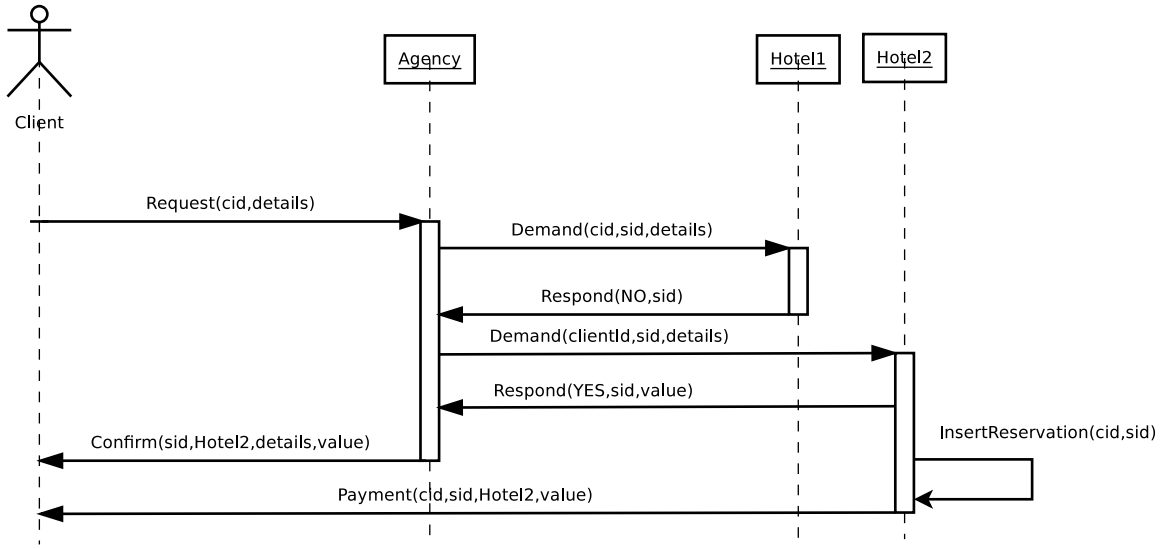

FIGURE 1 – Cas d'usage : deux consultations pour une réservation

Une des étapes de l'élaboration du contrat consiste à déterminer les informations qui devront être logguées. Par exemple, nous choisissons ici de ne pas enregistrer les détails des réservations ni leur coût (qui n'interviennent pas dans les types de plaintes considérés plus loin). Nous supposons que chaque acteur (l'agence et chaque hôtel) gère ses propres logs et que le client dispose aussi de son log (décrivant les messages reçus de l'agence et des hôtels). Les événements enregistrés dans les logs sont les interactions et prennent la forme $(Send, Source, Dest, Action, Params)$ ou $(Rec, Source, Dest, Action, Params)$ selon que l'enregistrement corresponde à un envoi ou une réception de message. Par exemple, la demande d'une réservation par l'agence à $Hotel1$ se traduit par l'entrée $(Send, Agency, Hotel1, Demand, [sid1, cid1])$

dans le log de l'agence et par l'entrée $(Rec, Agency, Hotel1, Demand, [sid1, cid1])$ dans le log de l'$Hotel1$. Le paramètre $sid1$ identifie le numéro de session et $cid1$ l'identifiant du client. Dans le cas d'une action locale, nous prenons la convention de désigner l'acteur concerné à la fois comme origine et comme destinataire. Par exemple, l'enregistrement de la réservation effective d'une chambre par $Hotel1$ correspond à l'entrée $(Send, Hotel1, Hotel1, InsertReservation, [cid1, sid1])$ dans le log de $Hotel1$.

Nous faisons l'hypothèse que l'infrastructure de logs permet d'enregistrer les événements relatifs à un même log dans leur ordre d'apparition. L'entrelacement des logs repose sur la technique d'ordonnancement des envois et réceptions proposé dans [14] [3]. L'algorithme de fusion des logs est décrit dans [19].

Considérons à titre d'exemple le scénario décrit dans la Figure 2. Le client d'identifiant cid adresse une demande à l'agence qui attribue à cette demande l'identifiant sid. Celle-ci s'adresse à l'hôtel $Hotel1$ qui effectue la réservation, lance le débit mais répond "NO" à l'agence. L'agence sollicite donc l'hôtel $Hotel2$ qui accepte la réservation, lance un débit mais omet d'enregistrer la réservation dans sa base de données.

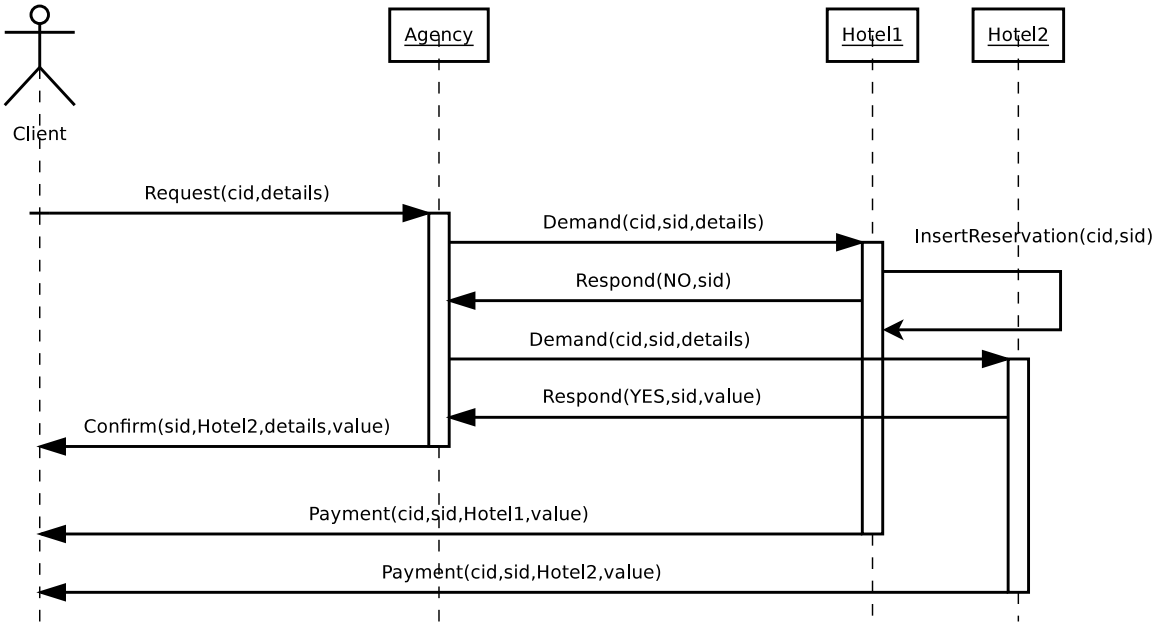

FIGURE 2 – Un dysfonctionnement

Les logs correspondant à ce scénario peuvent se présenter de la manière suivante :

3. On suppose que deux événements ne sont jamais identiques. L'ajout d'un numéro de session permet, le cas échéant, de garantir cette hypothèse.

Composant	Log
$Client$	<(Send,Client,Agency,Request,[cid1]), (Rec,Agency,Client,Confirm,[sid1,Hotel2]), (Rec,Hotel1,Client,Payment,[cid1,sid1,Hotel1]), (Rec,Hotel2,Client,Payment,[cid1,sid1,Hotel2])>
$Agency$	<(Rec,Client,Agency,Request,[cid1]), (Send,Agency,Hotel1,Demand,[cid1,sid1]), (Rec,Hotel1,Agency,Respond,[NO,sid1]), (Send,Agency,Hotel2,Demand,[cid1,sid1]), (Rec,Hotel2,Agency,Respond,[YES,sid1]), (Send,Agency,Client,Confirm,[sid1])>
$Hotel1$	<(Rec,Agency,Hotel1,Demand,[cid1,sid1]), (Send,Hotel1,Agency,Respond,[NO,sid1]), (Send,Hotel1,Hotel1,InserReservation,[cid1,sid1]), (Send,Hotel1,Client,Payment,[cid1,sid1,Hotel1])>
$Hotel2$	<(Rec,Agency,Hotel2,Demand,[cid1,sid1]), (Send,Hotel2,Agency,Respond,[YES,sid1]), (Send,Hotel2,Client,Payment,[cid1,sid1,Hotel2])>

Ce scénario présente donc deux erreurs de deux composants différents. Il se traduit pour le client par deux types de dysfonctionnements, qui peuvent conduire à deux types de plaintes différents : un double débit ou une réservation dans un hôtel différent de celui qui a été confirmé par l'agence. Nous montrons dans les deux sections suivantes comment prendre en compte les plaintes pour établir des relations précises entre les erreurs de composants et les responsabilités des acteurs.

4 Table de responsabilités

Avant de définir notre première méthode d'attribution des responsabilités (par tableaux) en section 4.2, nous présentons la spécification des logs dans la section 4.1, ainsi que la notion de plaintes, basée sur la description de propriétés paramétrées sur les logs.

4.1 Logs et propriétés sur les logs

Un log est représenté par un couple $(comps, content)$ (type LOG_FILE) qui décrit l'ensemble des composants dont les actions sont journalisées et le contenu des entrées du log. Par exemple, si chaque composant $Comp_i$ possède son propre log, la première composante de ce log est $\{Comp_i\}$; si la solution retenue est une infrastructure de log centralisée, alors la première composante de ce log est $\{Comp1, \ldots, CompN\}$. Pour un log de la forme $(comps, content)$ alors $content$ contient tous, et que, les messages envoyés ou reçus par chaque composant C de $comps$.

Une propriété paramétrée P est une fonction qui prend en entrée une séquence de paramètres et un log et qui retourne vrai si P est vérifiée par le log et faux sinon (type $\text{seq}(PARAM) \nrightarrow (LOG\_FILE \nrightarrow \mathsf{BOOL})$). Par exemple, la propriété $prop_{DoubleDebit}$ ci-dessous décrit le fait qu'un log contient deux débits, impliquant deux hotels $Hotel1$ et $Hotel2$, pour un même client et un même numéro de session :
Propriété $prop_{DoubleDebit}$
Paramètres : $[cid, sid, Hotel1, Hotel2]$ tel que $Hotel1 \neq Hotel2$

Log : $log = (comps, content)$ tel que $\{Client\} \subseteq comps$
Valeur de la propriété :

$$bool((Rec, Hotel1, Client, Payment, [cid, sid, Hotel1]) \in content \wedge \\ (Rec, Hotel2, Client, Payment, [cid, sid, Hotel2]) \in content)$$

Rappelons que dans la notation B l'expression $bool(P)$ transforme la valeur de vérité du prédicat P en une valeur booléenne (TRUE si P est vraie FALSE sinon). Une plainte est alors un quadruplet qui lie un plaignant, un défenseur, des paramètres et une propriété décrivant une classe de sources possibles d'un dysfonctionnement. Ci-après l'exemple de la plainte $DoubleDebit$:

Nom	$DoubleDebit$
Plaignant	$Client$
Défenseur	$Agency$
Paramètres	$[cid, sid, Hotel1, Hotel2]$
Source dysfonctionnement	$prop_{DoubleDebit}([cid, sid, Hotel1, Hotel2])$

Une plainte peut être paramétrée et le dysfonctionnement est une propriété non paramétrée de type $LOG\_FILE \nrightarrow \mathsf{BOOL}$. Une instance de plainte est un couple de la forme (nom de la plainte, valeur des paramètres). L'exemple de plainte ci-dessous concerne le fait que le client reçoit une confirmation erronée.

Nom	$ReservationError$
Plaignant	$Client$
Défenseur	$Agency$
Paramètres	$[cid, sid, Hotel]$
Source dysfonctionnement	$prop_{ReservationError}([cid, sid, Hotel])$

La propriété $prop_{ReservationError}$ étant décrite de la manière suivante :
Propriété $prop_{ReservationError}$
Paramètres : $[cid, sid, Hotel]$
Log : $log = (comps, content)$ tel que $\{Client, Hotel\} \subseteq comps$
Valeur de la propriété :

$$\exists bool(\ ((Rec, Agency, Client, Confirm, [sid, Hotel]) \in content \wedge \\ (Send, Hotel, Hotel, InsertReservation, [cid, sid]) \notin content) \\ \vee \\ ((Rec, Agency, Client, Confirm, [sid, Hotel]) \notin content \wedge \\ (Send, Hotel, Hotel, InsertReservation, [cid, sid]) \in content)\ )$$

4.2 Attribution des responsabilités

La démarche proposée dans [31] préconise l'inscription dans le contrat d'une liste de plaintes possibles correspondant aux dysfonctionnements du système jugés suffisamment importants pour que les parties s'accordent au préalable sur les responsabilités associées. L'attribution des responsabilités passe par l'identification de certaines sources de dysfonctionnement. A titre d'exemple, nous supposons que les parties ont identifié quatre erreurs potentielles pouvant être à l'origine d'une plainte pour double débit :

– $Prop_{OuiNon}$: $Hotel$ répond négativement à l'agence mais lance tout de même une demande de débit
Paramètres : $[cid, sid, Hotel]$
Log : $log = (comps, content)$tel que $\{Hotel, Agency\} \subseteq comps$
Valeur de la propriété :
$$bool((Send, Hotel, Agency, Respond, [NO, sid]) \in content \wedge (Send, Hotel, Client, Payment, [cid, sid, Hotel]) \in content)$$
– $Prop_{IgnoreOui}$: l'agence lance une demande de réservation auprès d'un hôtel alors qu'elle a déjà reçu une réponse positive d'un autre hôtel :
Paramètres : $[cid, sid, Hotel1, Hotel2]$
Log : $log = (comps, content)$ tel que $\{Agency\} \subseteq comps$
Valeur de la propriété :
$$bool((Send, Agency, Hotel1, Demand, [cid, sid]) \in content \wedge (Send, Agency, Hotel2, Demand, [cid, sid]) \in content \wedge (Rec, Hotel1, Agency, Respond, [YES, sid]) \in content \wedge (Rec, Hotel2, Agency, Respond, [YES, sid]) \in content)$$
– $Prop_{IgnoreNon}$: l'agence reçoit une réponse négative mais confirme la réservation
Paramètres : $[sid, Hotel]$
Log : $log = (comps, content)$ tel que $\{Agency\} \subseteq comps$
Valeur de la propriété :
$$bool((Rec, Agency, Hotel, Respond, [NO, sid]) \in content \wedge (Send, Agency, Client, Confirm, [sid, Hotel]) \in content)$$
– $Prop_{ErreurInsert}$: l'hotel répond positivement mais omet d'enregistrer la réservation
Paramètres de l'application : $[cid, sid, Hotel]$
Paramètre log : $log = (comps, content)$ tel que $\{Hotel, Agency\} \subseteq comps$
Valeur de la propriété :
$$bool(((Rec, Hotel, Agency, Respond, [YES, sid]) \in content \wedge (Send, Hotel, Hotel, InsertReservation, [cid, sid]) \notin content) \vee ((Rec, Hotel, Agency, Respond, [NO, sid]) \in content \wedge (Send, Hotel, Hotel, InsertReservation, [cid, sid]) \in content))$$

A titre d'exemple, les parties peuvent se mettre d'accord sur la répartition de responsabilités pour la plainte $DoubleDebit$ (pour les paramètres $[sid, cid, Hotel1, Hotel2]$) :

Plainte : $DoubleDebit$	
Caractérisation de l'erreur	Partie responsable
$Prop_{OuiNon}([sid, cid, Hotel1])$	$Hotel1$
$Prop_{OuiNon}([sid, cid, Hotel2])$	$Hotel2$
$Prop_{IgnoreOui}([sid, cid, Hotel1, Hotel2])$	$Agency$

Cette table peut être lue de la manière suivante : un acteur porte une responsabilité si l'une des lignes dans lesquelles il apparaît en colonne droite correspond à une classe d'erreurs identifiée dans les logs (colonne gauche). La répartition des responsabilités entre les acteurs concernés (quand plusieurs d'entre eux satisfont cette condition) peut être effectuée à parts égales ou selon tout autre modalité définie dans le contrat.

De la même façon, les parties peuvent se mettre d'accord sur la répartition de responsabilités pour la plainte $ReservationError$ (avec les paramètres $[sid, cid, Hotel]$) :

Plainte : $ReservationError$	
Caractérisation de l'erreur	Partie responsable
$Prop_{IgnoreNon}([sid, Hotel])$	$Agency$
$Prop_{ErreurInsert}([sid, cid, Hotel])$	$Hotel$

Si on applique les règles définies dans les deux tables au scénario fautif présenté dans la section 3, on peut conclure que c'est $Hotel1$ qui est désigné responsable en cas de plainte pour double débit (occurrence de l'erreur $Prop_{OuiNon}([sid, cid, Hotel1])$), alors que pour une plainte pour erreur de réservation, les deux hôtels sont responsables (occurrences de chacune des deux erreurs).

D'un point de vue pratique, la recherche des propriétés dans les logs (validité des plaintes et occurrences d'erreurs) est mise en oeuvre via l'outil d'analyse de logs LoGan [33] qui permet de rechercher une propriété CTL sur des logs distribués. Le langage accepté par l'analyseur comporte des restrictions par rapport à la formulation des propriétés en B (quantification bornée par exemple) mais toutes les propriétés ayant trait à l'apparition, la non-apparition ou à l'ordre d'occurrence des événements sont traitées.

5 Analyse de causalité logique

Les répartitions de responsabilités définies dans la section précédente reposent sur une définition fixée au préalable des relations entre les erreurs potentielles des composants et les plaintes auxquelles elles peuvent conduire. Une autre manière de procéder consiste à caractériser ce lien de manière plus générale à travers une relation de causalité. En droit, la notion de causalité joue un rôle essentiel dans la définition de la responsabilité civile. Dans le droit civil francais, une victime doit prouver la présence de trois conditions afin de prétendre à une réparation :

1. L'existence d'une faute ou *fait générateur* qui ne doit pas relever de la force majeure, ou être du fait de la victime
2. La présence d'un dommage qui doit consister en une atteinte à un intérêt légitime
3. Un lien de cause à effet entre la faute et le dommage

Le lien de causalité doit être direct et certain, ce qui est généralement difficile à établir car un événement est souvent le fruit d'une pluralité de facteurs. Plusieurs théories de la causalité ont été proposées par les juristes ([24, 13]), les plus connues étant les suivantes :

- La théorie de l'*équivalence des conditions* selon laquelle tous les antécédents nécessaires du dommage sont des causes. Cette théorie est la plus simple à appliquer, mais elle ne permet pas de distinguer la gravité des fautes.
- La théorie de la *cause proche* qui privilégie la dernière cause, celle qui est la plus proche du dommage dans le temps. Cependant l'ordre chronologique n'est pas nécessairement l'ordre causal.
- La *cause adéquate* est définie comme un fait qui rend le dommage objectivement probable. Il s'agit donc d'une définition difficile à formaliser et qui laisse plus de marge de manœuvre au juge.

La jurisprudence française rejette en général les théories de l'équivalence des conditions et de la cause proche, jugées trop arbitraires, pour privilégier celle de la cause adéquate ([24]).

Les travaux décrits ici s'inscrivant dans un cadre contractuel, les parties ont la possibilité de définir leurs propres règles d'attribution des responsabilités : en d'autres termes, nous traitons de responsabilités contractuelles et non de responsabilités civiles. Cependant, comme il a été rappelé dans [11], des

précautions doivent être prises lors de la rédaction du contrat pour éviter l'invalidation des clauses attributives de responsabilité par le juge. L'utilisation de notions de causalité retenues par la jurisprudence peut concourir à renforcer ces clauses et assurer leur effectivité. Nous présentons dans cette section une définition formelle de causalité qui permet d'exprimer de manière logique le principe à l'oeuvre dans la notion de causalité adéquate.

5.1 Causalité logique et analyse de traces

Les définitions de causalité logique proposées ci-dessous permettent de caractériser la causalité comme une relation entre la violation de la spécification d'un acteur et la violation d'une spécification globale, les deux étant des propriétés de sûreté. La notion de causalité impose de disposer de spécifications plus détaillées, en particulier pour caractériser les comportements corrects. Nous considérons un ensemble de spécifications $\mathcal{S}_i$ sur le comportement attendu des acteurs i et une spécification globale $\mathcal{S}$ du système. Nous notons $log_i \models \mathcal{S}_i$ l'acceptation de log_i par $\mathcal{S}_i$.

On considère ici qu'un log est une séquence d'entrées (partie $content$ de la formalisation des logs donnée section 4.1). On dénotera par log un log relatif à l'ensemble du système et $\pi_i(log)$ la projection de log sur le composant i (partie $comps$ de la formalisation des logs donnée section 4.1), obtenue en conservant, dans leur ordre d'apparition, les entrées de la forme $\langle Send, i, -, -\rangle$ et $\langle Rec, -, i, -\rangle$. Nous utilisons un prédicat $Comm$ pour caractériser l'ensemble de logs bien formés, dans le sens où chaque réception de la forme $\langle Rec, j, i, e\rangle$ est précédée de son émission de la forme $\langle Send, i, j, e\rangle$.

Les spécifications S_i et S doivent être telles que, si tous les acteurs satisfont $\mathcal{S}_i$, $i = 1, ..., n$, et que les communications entre les composants sont correctes, alors le système satisfait $\mathcal{S}$:

$$\forall\, log : ((\forall\, i : \pi_i(log) \models \mathcal{S}_i) \wedge Comm(log) \qquad \Rightarrow \qquad log \models \mathcal{S})$$

Cette hypothèse nous permet d'imputer toute violation de $\mathcal{S}$ à la violation d'au moins une spécification $\mathcal{S}_i$. Dans la suite nous discutons trois définitions de causalité permettant d'analyser les responsabilités dans la violation de $\mathcal{S}$ dans le cas où plusieurs acteurs ont violé leurs spécifications.

Definition 1 (Causalité faible) *Soit log un log tel que $log \not\models S$ et soit log' le plus petit préfixe de log tel que $log' \not\models \mathcal{S}$. $log_k = \pi_k(log)$ est une cause faible pour la violation de $\mathcal{S}$, noté $log_k \nearrow \mathcal{S}$, si $log_k \not\models \mathcal{S}_k$ et $i \leq |\pi_k(log')|$, où $i = \min\{m \mid log_k[1..m] \not\models \mathcal{S}_k\}$ est la position où $\mathcal{S}_k$ est violée pour la première fois.*

Intuitivement, log_k est une cause faible si la violation de $\mathcal{S}_k$ par log_k se produit avant la violation de $\mathcal{S}$ par log.

Definition 2 (Causalité nécessaire) *Soit log un log tel que $log \not\models S$. $log_k = \pi_k(log)$ est une cause nécessaire pour la violation de $\mathcal{S}$, noté $log_k \nearrow^{\mathbf{n}} \mathcal{S}$, si $log_k \nearrow \mathcal{S}$ et pour toute séquence finie d'événements log' qui satisfait $Comm(log')$:*

$$\forall\, j \in \{1, ..., n\} \setminus \{k\} : \pi_j(log') = \pi_j(log) \ \wedge\ \pi_k(log') \models \mathcal{S}_k \quad \Rightarrow \quad log' \models \mathcal{S}$$

La définition de causalité nécessaire utilise un raisonnement contrefactuel pour analyser des scénarios alternatifs si l'acteur k avait respecté $\mathcal{S}_k$. Plus précisément, log_k est une cause nécessaire pour la violation de $\mathcal{S}$ si log_k est une cause faible, et en remplaçant log_k par tout comportement qui satisfait $\mathcal{S}_k$, $\mathcal{S}$ aurait été satisfaite ou le log n'aurait pas été bien formé ($\neg\, Comm(log)$).

Etant donné un ensemble L de logs, $log \in L$ est dit *maximal* s'il n'est le préfixe strict d'aucun autre log de L.

Definition 3 (Causalité suffisante) *Soit log un log tel que $log \not\models S$. $log_k = \pi_k(log)$ est une cause suffisante pour la violation de $\mathcal{S}$, noté $log_k \nearrow^{\mathbf{s}} \mathcal{S}$, si $log_k \nearrow \mathcal{S}$ et pour toute séquence d'événements log' :*

$$Comm(log') \wedge (\forall\, p \in \mathcal{I} : \pi_p(log') \models \mathcal{S}_p) \wedge (\forall\, p \notin \mathcal{I} : \pi_p(log') = log_p)) \quad (1)$$
$$\wedge\ log' \textit{ est maximal parmi les logs qui satisfont } (1) \quad \Rightarrow \quad log' \not\models \mathcal{S}$$

où $\mathcal{I} = \{i \in \{1, ..., n\} \setminus \{k\} \mid log_i \not\models \mathcal{S}_i\}$.

log_k est une cause suffisante pour la violation de $\mathcal{S}$ si log_k est une cause faible et le fait de remplacer *tous les autres comportements* erronés sauf log_k par des comportements corrects donne toujours lieu à des scénarios qui violent $\mathcal{S}$. La condition de maximalité assure que le log est assez long pour propager les effets de la violation de $\mathcal{S}_k$.

Les définitions de causalité ont été implémentées dans l'outil LoCa qui automatise l'analyse de causalité. LoCa prend en entrée les spécifications $\mathcal{S}_i$ des acteurs, une spécification globale $\mathcal{S}$, un fichier avec les logs et la requête précisant quel type de causalité (nécessaire ou suffisante) est à analyser. L'analyse est effectuée de manière symbolique — sans passer par une énumération des comportements — en utilisant l'approche décrite dans [9]. Afin de faciliter la compréhension du résultat de l'analyse, LoCa donne des scénarios témoins, par exemple pour illustrer un comportement log' qui ne satisfait pas la quantification universelle dans les définitions 2 et 3.

Il n'existe bien sûr pas de correspondance exacte de nos définitions avec les différentes notions de causalité en responsabilité civile, la principale raison étant que ces dernières ne sont pas définies formellement et restent sujettes à interprétation. Néanmoins la notion de *causalité nécessaire* permet d'approcher la théorie de l'*équivalence des conditions* : toute cause nécessaire fait partie de l'ensemble des conditions équivalentes. La notion de *causalité suffisante*, en association avec la *causalité nécessaire*, est une bonne approximation de la théorie de la *cause adéquate* (fait rendant objectivement probable le dommage). Il peut bien sûr y avoir plusieurs faits nécessaires ou suffisants. Dans tous les cas, nos trois définitions contribuent à l'établissement d'un lien de cause à effet entre la faute et le dommage, ce lien pouvant être qualifié par les différentes notions proposées.

5.2 Application à l'étude de cas

Nous utilisons deux spécifications qui expriment les deux types de plaintes considérés dans la section 4. La spécification $\mathcal{S}_A$ exige qu'à tout instant, le nombre de confirmations soit supérieur au nombre de débits. La spécification $\mathcal{S}_B$, montrée en Figure 3, stipule que toute demande de réservation soit suivie soit d'une réponse négative, soit d'une réservation et d'une réponse positive par le même hôtel.

La table 1 résume les résultats de l'analyse effectuée avec l'outil LoCa pour le scénario fautif présenté dans la section 3 (Figure 2). Le comportement de l'agence respecte sa spécification et n'est donc pas une cause (ni faible, ni nécessaire, ni suffisante) pour la violation des spécifications $\mathcal{S}_A$ et $\mathcal{S}_B$. Le comportement de $Hotel1$ est une cause nécessaire pour la violation de $\mathcal{S}_A$: avec un comportement cohérent (pas de réservation ni débit) il n'y aurait pas eu deux débits. C'est également une cause suffisante : même si $Hotel2$ avait respecté sa spécification en effectuant la réservation, il y aurait eu deux débits pour une seule confirmation. $Hotel1$ n'est pas une cause nécessaire pour la violation de $\mathcal{S}_B$: même avec un comportement correct cohérent avec les comportements observés des autres acteurs, la spécification $\mathcal{S}_B$ reste violée par le comportement erroné de $Hotel2$ (réponse positive suivi d'un passage du temps sans réservation). Par contre, $Hotel1$ est une cause suffisante pour la violation de $\mathcal{S}_B$: même

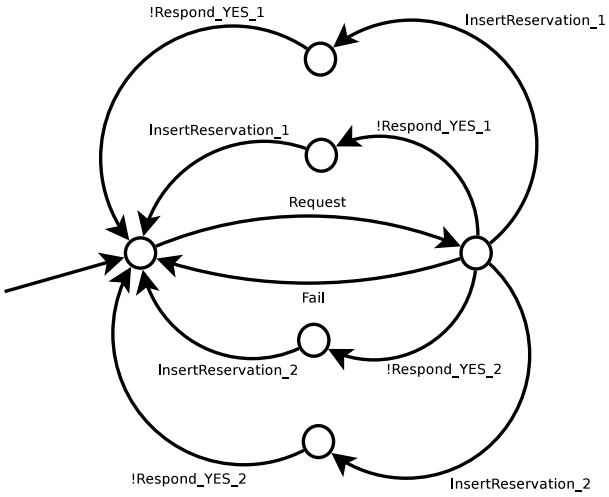

FIGURE 3 – Spécification $\mathcal{S}_B$: toute demande est suivie soit d'une réponse négative, soit d'une réservation et d'une réponse positive par le même hôtel.

Composant	$\mathcal{S}_A$	$\mathcal{S}_B$
$Agency$	–	–
$Hotel1$	nécessaire, suffisante	suffisante
$Hotel2$	–	suffisante

TABLE 1 – Résultats de l'analyse de causalité pour l'étude de cas.

si $Hotel2$ avait respecté sa spécification en effectuant la réservation, il y aurait eu deux réservations pour une seule réponse positive.

Le comportement de $Hotel2$ n'est pas une cause nécessaire pour la violation de $\mathcal{S}_A$: même si $Hotel2$ avait respecté sa spécification en effectuant la réservation, il y aurait eu deux débits pour une seule confirmation. Le comportement de $Hotel2$ n'est pas non plus une cause suffisante pour la violation de $\mathcal{S}_A$: si $Hotel1$ n'avait pas effectué de réservation ni de débit alors spécification $\mathcal{S}_A$ aurait été satisfaite.

$Hotel2$ n'est pas une cause nécessaire pour la violation de $\mathcal{S}_B$: même s'il avait respecté sa spécification en effectuant une réservation, la spécification serait restée violée par la présence d'une réservation par $Hotel1$ sans réponse positive correspondante. Par contre, le comportement de $Hotel2$ *est une cause suffisante* pour la violation de $\mathcal{S}_B$: même si $Hotel1$ n'avait pas effectué de réservation ni de débit, la spécification $\mathcal{S}_B$ resterait violée par la réponse positive sans réservation correspondante par $Hotel2$.

En résumé la table 1 permet de conclure que $Hotel1$ est seul responsable de la violation de la première spécification, tandis que la responsabilité pour la violation de la deuxième spécification est partagée par les deux hôtels dont chacun des comportements est suffisant pour violer la propriété.

6 Conclusion et perspectives

Dans cet article, nous avons présenté deux modes de définition de responsabilités des parties dans un contrat portant sur des logiciels. La première méthode, dite "par tableaux", repose sur une répartition prédéfinie, supposée résulter d'un accord entre les parties, qui permet d'associer automatiquement cer-

taines erreurs (ou combinaisons d'erreurs) détectées dans les logs aux responsabilités de parties désignées. La seconde repose sur une notion de causalité introduite dans [9] qui s'inspire de la théorie juridique de la "cause adéquate".

Ces approches reposent toutes les deux sur la formalisation des logs et de propriétés sur les logs qui peuvent aussi bien représenter des dysfonctionnements généraux, des erreurs locales sur les composants ou des comportements attendus. La généralité du cadre proposé repose sur le fait que les logs et propriétés sont paramétrés par les composants auquels ils se rapportent. Cette formalisation permet ainsi, pour les acteurs du contrat, de disposer d'une définition précise et non ambiguë de la procédure de répartition des responsabilités, pour les deux approches proposées. Un outil a été développé pour chacune de ces approches (LoGan et LoCa) pour des formes restreintes de propriétés (systèmes à état fini + propriétés temporelles).

L'analyse du scénario fautif présenté dans la section 3 conduit aux mêmes conclusions selon les deux méthodes. Ce cas d'étude permet cependant de comparer les avantages de chacune d'elles. On a pu constater que la méthode par tableaux oblige les parties à désigner un certain nombre de propriétés correspondant à des plaintes possibles de tiers (estimées les plus probables ou susceptibles de conduire aux indemnisations les plus importantes) et à des erreurs de composants (jugées les plus probables ou les plus graves). L'inconvénient de cette option est de dépendre de ces choix préalables des parties (qui peuvent mal estimer l'importance des propriétés en question, ou en ignorer certaines) mais cette flexibilité peut également représenter un avantage car elle reflète la liberté contractuelle des parties. Celles-ci peuvent par exemple convenir d'une répartition de responsabilités qui tienne compte de critères non techniques comme leurs intérêts respectifs dans le contrat ou leurs capacités financières à assumer des risques. Inversement, le principal avantage de la seconde option réside dans le fait que la définition de causalité est donnée une fois pour toutes et peut s'appliquer systématiquement à de nouveaux contrats, quelles que soient les plaintes et les dysfonctionnements. Néanmoins sa mise en oeuvre demande un effort de spécification plus important puisque les analyses de causalité reposent sur un raisonnement impliquant les comportements corrects ou incorrects des composants. En réalité, les deux méthodes peuvent être utilisées de manière complémentaire dans un contrat donné, la première pour traiter des cas bien identifiés, la seconde s'appliquant dans les cas résiduels.

Une perspective intéressante est d'appliquer les critères de causalité pour détecter des "anomalies logiques" dans une répartition de responsabilités définie par tableaux, dans l'objectif de fournir des aides aux parties s'engageant dans un contrat. Si, par exemple, les tableaux de la section 4 avaient rendu l'agence responsable pour une erreur $Prop_{OuiNon}$ (soit par inadvertance, soit délibérément, parce que l'agence avait accepté cette responsabilité), l'application de l'analyse de causalité aurait permis de détecter cette anomalie. Cette utilisation de nos travaux sur la causalité nécessite néanmoins d'étendre les définitions proposées, actuellement relatives à un log donné, à un ensemble de comportements possibles et de relier plus finement les analyses en résultant aux différentes notions de causalité en droit.

Remerciements. Ce travail est financé par le projet ANR LISE (Liability Issues in Software Engineering).

Références

[1] Ross Anderson and Tyler Moore. Information security economics - and beyond. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 68–91. Springer, 2007.

[2] Ali Reza Arasteh, Mourad Debbabi, Assaad Sakha, and Mohamed Saleh. Analyzing multiple logs for forensic evidence. *Digital Investigation*, 4 :82–91, 2007.

[3] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. Fundamental concepts of computer system dependability. In *IARP/IEEE-RAS Workshop on Robot Dependability : Technological Challenge of Dependable Robots in Human Environments, Seoul, Korea*, 2001.

[4] Daniel 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 - Scientific Literature Digital Library and Search Engine, 2007.

[5] Laura Brandán Briones, Alexander Lazovik, and Philippe Dague. Optimizing the system observability level for diagnosability. In *Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 815–830, 2008.

[6] Nicolas Craipeau. La preuve électronique. Technical report, PrINT - CRDP, 2009. LISE Deliverable D1.3, http ://licit.inrialpes.fr/lise/Rapports.html.

[7] Andrew D. H. Farrell, Marek J. Sergot, Mathias Sallé, and Claudio 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.

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

[9] Gregor Gössler, Daniel Le Métayer, and Jean-Baptiste Raclet. Causality analysis in contract violation. In *Runtime Verification (RV)*, volume 6418 of *Lecture Notes in Computer Science*, pages 270–284. Stringer, 2010.

[10] Guido Governatori, Zoran Milosevic, and Shazia Wasim Sadiq. Compliance checking between business processes and business contracts. In *Enterprise Distributed Object Computing Conference (EDOC)*, pages 221–232. IEEE, 2006.

[11] Ronan Hardouin. Le sens des responsabilités en matière de contrats informatiques. Technical report, Laboratoire DANTE, Université de Versailles Saint-Quentin, 2009. LISE Livabre D1.1, http ://licit.inriaples.fr/lise/Rapports.html.

[12] Anita K. Jones and Robert S. Sielken. Computer system intrusion detection : a survey. Technical report, University of Virginia, Computer Science Department, 1999.

[13] Jurispedia. Lien de causalité (fr.jurispedia.org/index/php/Lien_de_causalité_(fr)). [Online], 2011.

[14] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM (CACM)*, 21(7) :558–565, 1978.

[15] Daniel Le Métayer. A formal privacy management framework. In *Formal Aspects in Security and Trust (FAST)*, volume 5491 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2009.

[16] Daniel Le Métayer, Manuel Maarek, Eduardo Mazza, Marie-Laure Potet, Stéphane Frénot, Valérie Tong Viet Triem, Nicolas Craipeau, and Ronan Hardouin. Liability issues in software engineering : the use of formal methods to reduce legal uncertainties. *Communicantion of the ACM (CACM)*, 54 :99–106, April 2011.

[17] Daniel Le Métayer, Manuel Maarek, Eduardo Mazza, Marie-Laure Potet, Stéphane Frénot, Valérie Viet Triem Tong, Nicolas Craipeau, and Ronan Hardouin. Liability in software engineering – Overview of the LISE approach and illustration on a case study. In *International Conference on Software Engineering (ICSE)*, volume 1, pages 135–144. ACM/IEEE, 2010.

[18] Daniel Le Métayer, Eduardo Mazza, and Marie-Laure Potet. Designing log architectures for legal evidence. In *Software Engineering and Formal Methods (SEFM)*, pages 156–165. IEEE, 2010.

[19] Eduardo Mazza, Marie-Laure Potet, and Daniel Le Métayer. A formal framework for specifying and analyzing logs as electronic evidence. In *Brazilian Symposium on Formal Methods (SBMF)*, volume 6527 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2010.

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

[21] Simon L. Peyton Jones and Jean-Marc Eber. How to write a financial contract. In Jeremy Gibbons and Oege de Moor, editors, *The Fun of Programming*, Cornerstones of Computing, chapter 6. Palgrave Macmillan, 2003.

[22] Cristian Prisacariu and Gerardo Schneider. A formal language for electronic contracts. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 4468 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2007.

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

[24] C. Renault-Brahinsky. *Droit des obligations*. Mémentos LMD. Gualino editeur, 2011.

[25] Daniel J. Ryan. Two views on security software liability : Let the legal system decide. *IEEE Security & Privacy*, 1(1) :70–72, 2003.

[26] Fred B. Schneider. Accountability for perfection. *Security & Privacy, IEEE*, 7(2) :3–4, March-April 2009.

[27] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *International Conference on Software Engineering (ICSE)*, pages 179–188. IEEE, 2004.

[28] James Skene, Franco Raimondi, and Wolfgang Emmerich. Service-level agreements for electronic services. *IEEE Transactions on Software Engineering (TSE)*, 36(2) :288–304, 2010.

[29] James Skene, Allan Skene, Jason Crampton, and Wolfgang Emmerich. The monitorability of service-level agreements for application-service provision. In *International Workshop on Software and Performance (WOSP)*, pages 3–14. ACM, 2007.

[30] Mark Solon and Penny Harper. Preparing evidence for court. *Digital Investigation*, 1 :279–283, 2004.

[31] S. Steer, N. Craipeau, D. Le Metayer, M. Maarek, M-L Potet, and V. Viet Triem Tong. Définition des responsabilités pour les dysfonctionnements de logiciels : cadre contractuel et outils de mise en oeuvre. In *Droit, sciences et techniques, quelles responsabilités*, Lexisnexis, 2011.

[32] Peter Stephenson. Modeling of post-incident root cause analysis. *International Journal of Digital Evidence*, 2(2), 2003.

[33] Valérie Viet Triem Tong, Christophe Bidan, Ludovic Me, Chistopher Humphries, and Guillaume Piolle. Utilisation de la logique temporelle CTL pour la spécification de propriétés de responsabilité. Technical report, SUPELEC, Equipe SSIR, Rennes. LISE Livrable D4.1, http ://licit.inrialpes.fr/lise/Rapports.html.