Une méthode pour l'élicitation des besoins : application au système de contrôle d'accès

Jeanine Souquières
LORIA—Université Nancy2
B.P. 239 Bâtiment LORIA
F-54506 Vandœuvre-les-Nancy, France
Jeanine.Souquieres@loria.fr

Maritta Heisel
Fakultät für Informatik
Universität Magdeburg
D-39016 Magdeburg, Germany
heisel@cs.uni-magdeburg.de

Résumé : Cet article présente l'utilisation d'une approche systématique pour clarifier et analyser les besoins sur l'étude de cas d'un système de contrôle d'accès. L'approche intègre une détection systématique des interactions entre les différents besoins.

1 Objectifs de l'approche proposée

L'élicitation des besoins pour un système informatique est une activité conceptuelle complexe, nécessitant des efforts importants de mise en œuvre. Se situant à l'interface entre clients et développeurs ou maître d'œuvre et maître d'ouvrage, cette phase nécessite une attention particulière : fondée sur la communication, elle conduit souvent à des incompréhensions. D'une part, les clients ne sont pas toujours conscients de leurs besoins. D'autre part, les analystes ne possèdent pas la même connaissance du domaine d'application que les clients. Certains besoins sont tellement évidents pour les clients qu'ils ne méritent pas, à leurs yeux, d'être mentionnés et demeurent ignorés des analystes. Les clients n'ont pas toujours une vision claire de ce que le système doit faire et changent d'avis en cours de développement. Ils savent enfin ce qu'ils veulent lorsqu'ils utilisent le système et qu'il ne répond pas à leurs attentes [AP98].

Cette phase d'élicitation des besoins comporte plusieurs processus consistant à identifier les besoins du clients, les analyser, les hiérarchiser et les exprimer. Elle peut être en partie guidée et outillée. Les objectifs de l'approche systématique proposée pour couvrir cette phase visent à :

- comprendre le problème à résoudre,
- fixer le vocabulaire en proposant un seul nom par concept manipulé,
- désambiguïser les besoins,
- détecter des incohérences,
- détecter les besoins manquants et
- établir un document de départ pour la spécification du système informatique.

Pour cela, l'élicitation des besoins repose sur leur formalisation avec une recherche des interactions entre les différents besoins en vue de supprimer des incohérences. Les besoins sont décomposés en fragments les plus petits possibles. Le point de départ de l'analyse est constitué de l'ensemble des documents fournis par le client, le cahier des charges.

Le paragraphe 2 présente une description rapide de l'approche pour l'élicitation des besoins. L'approche prévoit une analyse détaillée des besoins amenant à une meilleure compréhension du problème ainsi qu'à la détection des interactions entre les différents besoins permettant, entre

autres, de prendre en compte leur évolution. Une illustration de l'approche sur l'étude de cas d'un système de contrôle d'accès à un ensemble de bâtiments est développée dans le paragraphe 3 avec une liste de l'ensemble des documents résultats. Le paragraphe 4 présente une discussion sur les bénéfices apportés par l'utilisation de l'approche proposée ainsi que les résulats obtenus avec cette étude de cas particulière.

2 Description de l'approche proposée

Nous avons développé une méthode pour faciliter les toutes premières phases du cycle de vie d'un logiciel, phases se situant à l'interface entre clients et développeurs. Il s'agit d'une méthode systématique dans laquelle une distinction claire entre expression des besoins et expression des spécifications est effectuée. Le point de départ de la phase d'élicitation des besoins est un document informel, par exemple celui fourni par les clients. L'approche proposée [HS99c] pour guider l'analyse et l'élicitation des besoins s'inspire des travaux de Jackson et Zave [JZ95, ZJ97] et des premières phases des méthodes et des notations orientées objet comme OMT [RBP+91], FUSION [CAB+94] ou UML [FS97]. Elle commence par un processus de réflexion dans lequel les informations relevant du domaine d'application et les besoins sont exprimés en langage naturel. Elle se poursuit par une formalisation de ces informations. Les principales étapes de cette approche sont décrites ci-dessous.

Étape 1 : Introduire le vocabulaire du domaine. Les différentes notions du domaine d'application sont exprimées sous forme textuelle.

Étape 2 : Enoncer les faits, les hypothèses et les besoins du système sous la forme d'un ensemble de fragments correspondant à des scénarios. Les faits font référence à des propriétés du domaine d'application, les besoins contraignent la spécification et l'implantation du système. Certains besoins ne peuvent pas être imposés et décrivent un comportement correspondant à un utilisateur parfait. Il est possible que le système ne soit pas à même de satisfaire correctement ces besoins dans tous les cas, mais seulement sous l'hypothèse d'un comportement attendu de l'utilisateur.

Étape 3 : Identifier les différents événements pouvant se produire avec leurs paramètres et les classer. Les événements concernent la partie réactive du système. La classification adoptée est celle proposée par Jackson et Zave [JZ95]. Elle consiste à préciser qui commande les événements et qui les observe.

Étape 4 : Identifier les opérations pouvant être utilisées par les utilisateurs du système avec leurs paramètres. Cette étape concerne la partie non-réactive du système. Pour les systèmes purement réactifs, cet ensemble d'opérations peut être vide. Les opérations du système sont habituellement indépendantes des composants physiques mais font référence aux informations disponibles dans le système informatique. A chaque opération *Op*, nous associons deux événements *OpInvocation* et *OpTermination*, le premier étant commandé par l'environnement et partagé par le système informatique, le deuxième étant commandé par le système informatique et partagé par l'environnement.

Les étapes 1 à 4 peuvent être effectuées dans n'importe quel ordre, en parallèle, avec des allers et retours. Des conditions de validation sont associées aux différentes étapes, assurant une certaine garantie de la qualité du produit obtenu. Elles expriment des conditions sémantiques nécessaires que les résultats de l'étape doivent satisfaire :

– chaque notion introduite dans l'étape 1 doit être en relation avec un événement ou une opération,
– il n'y a pas d'événements commandés par le système informatique et non partagés par l'environnement,
– toutes les notions utilisées dans les faits, les hypothèses et les besoins doivent être mises en relation avec le domaine, les événements ou les opérations introduits dans les étapes 1, 3 et 4,
– chaque notion du domaine et chaque événement doit être en relation avec au moins un fait, une hypothèse ou un besoin.

Étape 5 : Formaliser les faits, les hypothèses et les besoins comme des contraintes sur l'ensemble des traces possibles des événements du système. Dans cette étape, les contraintes sont formalisées les unes après les autres. Chaque nouvelle contrainte est analysée en relation avec l'ensemble des contraintes déjà formalisées avant d'être intégrée à cet ensemble, en vue de détecter si des inconsistences ou des comportements non désirés peuvent se produire [HS98]. Cette phase d'analyse et d'intégration d'une nouvelle contrainte peut être guidée; elle se décompose elle-même en cinq étapes décrites ci-dessous.

Étape 5.1 : Formaliser la nouvelle contrainte à l'aide d'une formule sur les traces du système. Pour formaliser les besoins, nous utilisons des traces, c'est-à-dire des suites d'événements se produisant sur un état du système à un instant donné. Le système démarre à l'état S_1. Lorsqu'un événement e_1 se produit à l'instant t_1, le système passe à l'état S_2, et ainsi de suite :

$$S_1 \xrightarrow[t_1]{e_1} S_2 \xrightarrow[t_2]{e_2} \ldots \; S_n \xrightarrow[t_n]{e_n} S_{n+1} \; \ldots$$

Soit Tr l'ensemble des traces permises. Les contraintes sont des formules qui restreignent l'ensemble Tr. Pour une trace $tr \in Tr$, $tr(i)$ dénote le i-ème élément d'une trace, $tr(i).s$ l'état du i-ème élément et $tr(i).e$ l'événement qui se produit dans cet état. Chaque préfixe d'une trace valide est aussi une trace valide. Une spécification formelle des traces est donnée dans l'annexe A. Pour exprimer les contraintes, des prédicats sur l'état global du système peuvent être introduits. Pour chaque prédicat introduit, on précisera les événements qui le modifient sa valeur.

Afin de disposer d'une manière systématique pour exprimer formellement les besoins et faciliter l'analyse des interactions, nous recommandons d'exprimer les contraintes à l'aide d'implications où soit la précondition fait référence à un état antérieur à celui de la postcondition, soit la pré- et la postcondition font référence au même état, i.e. il s'agit alors d'un invariant du système.

Étape 5.2 : Donner une version simplifiée de la contrainte. L'analyse des interactions et son automatisation nécessitent de manipuler les contraintes formalisées sous une forme simplifiée. La forme retenue est la suivante :

$$x_1 \diamond x_2 \diamond \ldots \diamond x_n \rightsquigarrow y_1 \diamond y_2 \diamond \ldots \diamond y_k$$

où les x_i, y_j sont des littéraux (symboles d'événements, de prédicats ou leur négation) et $\diamond$ signifie soit la conjonction, soit la disjonction. Le symbole $\rightsquigarrow$ indique que la précondition fait référence à un état antérieur à celui de la postcondition. Si la contrainte correspond à un invariant sur l'état du système, il est remplacé par $\Rightarrow$.

Pour transformer une contrainte en sa version simplifiée, il est fait abstraction des quantificateurs ainsi que des paramètres des prédicats et des événements.

Étape 5.3 : Mettre à jour les tables de relations sématiques. La syntaxe seule ne suffit pas pour détecter les interactions. Les relations sémantiques entre les différents symboles doivent aussi être prises en compte : un prédicat peut impliquer un autre prédicat, un événement peut se produire seulement si l'état du système satisfait un prédicat et pour chaque prédicat, nous devons connaître les événements qui le modifient. Trois tables de relations sémantiques sont construites :

(i) La table des conditions nécessaires pour chaque événement. Elle précise quels prédicats portant sur l'état du système doivent être vrais pour que l'événement puisse se produire. Si l'événement e ne peut se produire que lorsque pl est vrai (où pl désigne un symbole de prédicat ou sa négation), alors cette table aura une nouvelle entrée, $pl \leftsquigarrow e$.

(ii) La table des événements qui modifient les prédicats, c'est à dire les prédicats qui prennent la valeur vrai ou faux après qu'un événement se soit produit. Pour tout pl, où pl désigne un symbole de prédicat ou sa négation, nous devons connaître l'ensemble des événements qui le rendent vrai : $e \rightsquigarrow pl$.

(iii) La table des relations entre prédicats. Pour tout prédicat p, nous déterminons

- l'ensemble des prédicats qu'il entraîne : $p_{\Rightarrow} = \{q : PLit \mid p \Rightarrow q\}$,
- l'ensemble des prédicats que sa négation entraîne : $\neg\, p_{\Rightarrow} = \{q : PLit \mid \neg\, p \Rightarrow q\}$.

Étape 5.4 : Déterminer les candidats aux interactions. Les candidats aux interactions sont calculés à partir de l'expression simplifiée des besoins proposée à l'étape 5.2 et des tables de relations sémantiques de l'étape 5.3 en utilisant une procédure automatique décrite dans [HS98]. Cette procédure est constituée de deux parties comme illustré figure 1. L'analyse des *préconditions* détermine les contraintes dont les préconditions ne sont pas exclusives ou indépendantes les unes des autres. Si les postconditions sont incompatibles, il peut y avoir interaction. L'analyse des *postconditions* détermine comme candidats les contraintes dont les postconditions sont incompatibles. Si les préconditions de ces contraintes ne sont pas exclusives, il peut y avoir une interaction.

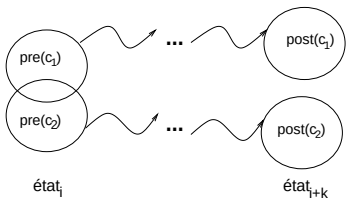

FIG. 1 – *Recherche des candidats aux interactions*

Analyse des préconditions. La figure 2 montre l'approche générale pour détecter l'ensemble des candidats $C_{pre}(c',far)$ aux interactions par une analyse de la précondition de la nouvelle contrainte c' relativement à l'ensemble *far* des faits, hypothèses et besoins déjà formalisés.

$$
\begin{array}{l}
C_{pre}(c',far) = \\
\quad \{c : far \mid precond(c) \cap precond(c') \neq \varnothing\} \\
\quad \cup \\
\quad \bigcup_{x \in pre\_predicates(c')} \{c : far \mid ((_{\Rightarrow}x \cup x_{\Rightarrow}) \cap precond(c) \neq \varnothing) \\
\qquad\qquad\qquad\qquad \vee \\
\qquad\qquad\qquad\qquad (\exists\, e : precond(c) \cap EVENT;\ y : {}_{\Rightarrow}x \cup x_{\Rightarrow} \bullet y \leftsquigarrow e)\}
\end{array}
$$

avec

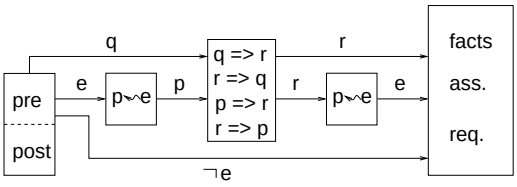

FIG. 2 – *Détection des candidats aux interactions par une analyse des préconditions*

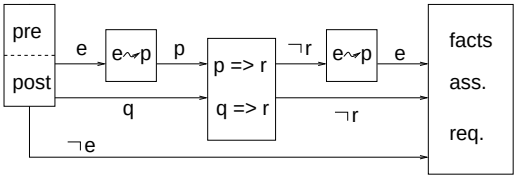

FIG. 3 – *Détection des candidats aux interactions par une analyse des postconditions*

$\leftsquigarrow e = \{pl : PLit \mid pl \leftsquigarrow e\}$
$pre\_predicates(c) = (precond(c) \cap PLit) \cup \bigcup_{e \in precond(c) \cap EVENT} \leftsquigarrow e$

Analyse des postconditions. Pour détecter les postconditions conflictuelles, nous effectuons un chaînage avant sur les postconditions de la nouvelle contrainte, nous prenons la négation des littéraux résultants et vérifions si la négation de l'un de ces littéraux est une conséquence de la postcondition d'une contrainte formalisée. Si c'est le cas, la contrainte est un candidat possible aux interactions. Pour effectuer le chaînage sur les événements, nous utilisons la table des événements qui modifient les prédicats. Aucun chaînage n'est effectué sur la négation des événements. La figure 3 donne une vue générale de la procédure.

$C_{post}(c', far) =$
$\{c : far \mid postcond(c)\ opposite\ postcond(c')\}$
$\cup$
$\{c : far \mid \exists x : post\_predicates(c);\ y : post\_predicates(c') \bullet x_{\Rightarrow}\ opposite\ y_{\Rightarrow}\}$

avec

$e_{\rightsquigarrow} = \{pl : PLit \mid e \rightsquigarrow pl\}$
$post\_predicates(c) = (postcond(c) \cap PLit) \cup \bigcup_{e \in postcond(c) \cap EVENT} e_{\rightsquigarrow}$
$ls_1\ opposite\ ls_2 \Leftrightarrow \exists x : ls_1 \bullet \neg x \in ls_2$

où ls_1 et ls_2 sont des ensembles de littéraux.

Étape 5.5 : Décider s'il y a des interactions entre la nouvelle contrainte et les candidats déterminés à l'étape 5.4. Ce travail doit être effectué en relation avec le client.

Étape 5.6 : Prendre en compte chaque interaction. Pour prendre en compte une interaction, trois possibilités sont offertes : (i) corriger un fait correspondant à la formalisation de la connaissance du domaine, (ii) relâcher un besoin en jouant sur sa pré- ou sa postcondition, (iii) contraindre une hypothèse. Après modification d'une contrainte, il est nécessaire d'effectuer une analyse des interactions entre la contrainte modifiée et l'ensemble des contraintes formalisées.

L'étape 5 est accompagnée des conditions de validation suivantes :

– chacun des besoins énoncés dans l'étape 2 doit être exprimé,
– l'ensemble des contraintes doit être consistant,
– pour chaque prédicat introduit dans les besoins, les événements qui modifient sa valeur doivent être partagés par le système informatique.

Le travail réalisé dans les étapes 5.1 à 5.6 permet de préserver la cohérence mutuelle entre les différentes contraintes et nous amène parfois à rétro-agir sur leur définition.

3 Etude de cas

L'approche proposée pour l'élicitation de besoins est appliquée au système de contrôle d'accès à un ensemble de bâtiments. Le point de départ de l'étude de cas est consultable à l'adresse http://www-lsr.imag.fr/Les.Personnes/Yves.Ledru/Controle_acces. Pour des raisons de place, nous ne prendrons pas en compte la détection d'un incendie.

3.1 Présentation des différentes étapes

Dans la suite, nous faisons référence au cahier des charges en utilisant la numérotation de ses différents paragraphes.

Etape 1 : Introduire le vocabulaire du domaine. L'objectif est de gérer l'accès de *personnes* à un ensemble de *bâtiments* et de mémoriser ces accès en vue de répondre à des requêtes sur la présence de personnes dans les bâtiments. Un bâtiment dispose d'un ensemble de *portes*, chaque porte pouvant être soit une porte d'entrée, soit une porte de sortie. Une porte est munie d'un *lecteur de cartes*. Un lecteur est muni de deux *v*oyants, l'un *vert* et l'autre *rouge*. L'accès à un bâtiment est réservé à certains *groupes*, un groupe étant constitué de plusieurs personnes, une personne ne pouvant appartenir qu'à un seul groupe. Une personne dispose d'une *carte d'accès* dont l'identification est unique.

Le système à modéliser est à la fois un système réactif et un système transformationnel (avec l'aspect mémorisation et requêtes sur les informations mémorisées).

Etape 2 : Enoncer les faits, les hypothèses et les besoins du système. Le cahier des charges initial est relativement bien structuré avec une description fragmentée des différents besoins, chacun d'eux étant numéroté.

Les *faits* identifiés concernent la porte : une porte peut seulement être bloquée ou débloquée si elle est fermée.

Les *hypothèses* identifiées émanent du paragraphe 3.3 du cahier des charges dans lequel il est stipulé que les "usagers sont priés de respecter ce protocole" ainsi que du paragraphe 3.2.1 exprimant le protocole d'accès ("une personne qui souhaite entrer ou sortir d'un bâtiment insère sa carte dans le lecteur si le voyant n'est pas rouge").

Etape 3 : Identifier les événements et les classer. Il n'y a pas d'événements commandés par l'environnement et non partagés par le système informatique.

Les événements commandés par l'environnement et observables par le système informatique sont *inserer_carte*(*carte*,*lecteur*), *entrer*(*carte*,*porte*), *sortir*(*carte*,*porte*), *ouvrir*(*porte*), *fermer*(*porte*).

Les événements commandés par le système informatique et observables par l'environnement sont *bloquer*(*porte*), *debloquer*(*porte*), *allumer_vert*(*lecteur*), *eteindre_vert*(*lecteur*), *allumer_rouge*(*lecteur*), *eteindre_rouge*(*lecteur*).

Comme requis par la condition de validation de cette étape, il n'y a pas d'événements commandés par le système informatique et non partagés par l'environnement.

Un diagramme entité-relation, présenté figure 4, visualise les résultats informels de ces trois premières étapes. Les rectangles représentent des entités et les losanges des relations. Les flèches indiquent les directions des relations. La relation *avoir* décrit les aspects statiques du système alors que les autres relations décrivent les aspects dynamiques correspondant aux événements identifiés.

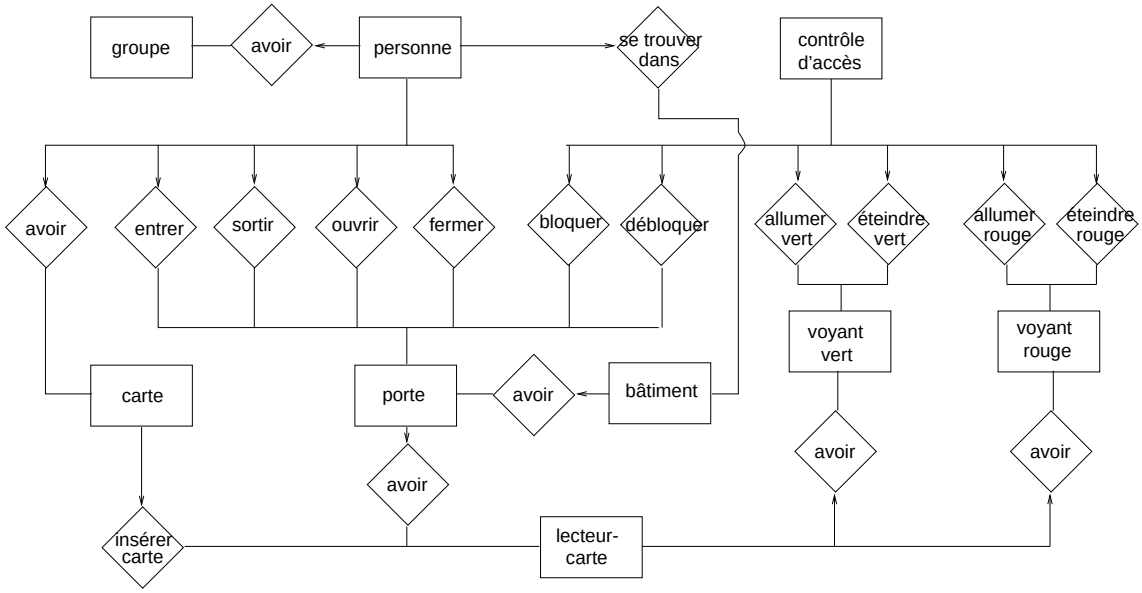

FIG. 4 – *Visualisation graphique résultante des trois premières étapes*

Etape 4 : Identifier les opérations. Elles concernent la partie transformationnelle du système:

Opérations	motivées par
definir_autorisations(*bat*,*groupe*)	1.1
definir_groupes	1.2
liste_des_presents(*bat*,*heure*)	2.1—2.3
cartes_en_circulation	3.1.1
cartes_perdues	3.1.1

Etape 5 : Formaliser les faits, les hypothèses et les besoins. Dans la suite, nous décrivons les étapes de formalisation pour un sous-ensemble des faits, hypothèses et beoins de l'étude de cas. Une description complète de cette formalisation est disponible dans le rapport [HS99a]. De manière générale, toutes les traces utilisées sont locales et font référence à un seul lecteur et à sa porte correspondante.

Fait. Parmi les propriétés du domaine d'application, citons le fonctionnement de la porte : une porte peut être bloquée ou débloquée seulement si elle est dans l'état fermée.

$$\forall\, tr : Tr;\ po : PORTE;\ \bullet\ \forall\, i : \mathsf{dom}\ tr \bullet$$
$$tr(i).e = bloquer(po) \Rightarrow fermee(tr(i).s, po)$$

Cette formule exprime le fait que si le *i*-ième élément de la trace *tr* est l'événement *bloquer*, alors la porte est dans l'état *fermee*, où *fermee* correspond à une des valeurs des états possibles de la porte.

Remarque. Pour le débloquage de la porte, la formule est analogue.

Expressions schématiques. La version simplifiée des faits s'exprime à l'aide des deux formules suivantes. Si la porte n'est pas fermée, les événements *bloquer* et *debloquer* ne peuvent pas se produire :
$\neg$ *fermee* $\leadsto \neg$ *bloquer*
$\neg$ *fermee* $\leadsto \neg$ *debloquer*

Tables sémantiques. Les événements *bloquer* et *debloquer* ne peuvent se produire que si la porte est fermé :
fermee $\leftsquigarrow$ *bloquer*
fermee $\leftsquigarrow$ *debloquer*

La formalisation de ce fait a introduit le prédicat *fermee* sur l'état du système. Il faut spécifier ici les événements qui modifient ce prédicat. Ce sont les événements *fermer* et *ouvrir*, événements observables par le système informatique, par l'intermédiaire de capteurs décrits dans les paragraphes 4.2.1(b) et 4.2.1(c) du cahier des charges :
fermer $\leadsto$ *fermee*
ouvrir $\leadsto \neg$ *fermee*

Il n'y a pas d'interaction entre les deux faits. Un résumé des expressions schématiques et des candidats aux interactions est présenté figure 5.

Besoin 2. Il concerne l'enregistrement des accès : "tout accès est enregistré (personne, bâtiment, heure) tant en entrée qu'en sortie du bâtiment".

$$\forall\, tr : Tr;\ c : CARTE;\ po : PORTE \bullet \forall\, i : \mathsf{dom}\, tr \mid i < \#tr \bullet$$
$$tr(i).e = entrer(c,po) \Rightarrow dans\_batiment(tr(i+1).s,c,bat(po),tr(i).t)$$

Si le *i*-ième élément de la trace *tr* est événement *entrer*, cet élément n'étant pas le dernier élément de la trace, alors la personne associée à la carte sera présente dans le bâtiment dans l'état suivant du système. Le système ne connait que des cartes, ce qui explique le paramètre de l'evénement *entrer*. Le prédicat *dans_batiment* a comme paramètres une carte, un bâtiment et l'instant où la personne entre dans le bâtiment. Ce prédicat, qui doit être implanté dans le système informatique, exprime l'enregistrement de l'entrée.

Ceci nous amène à introduire une nouvelle fonction: $bat : PORTE \rightarrow BATIMENT$, qui associe à chaque porte, le bâtiment auquel elle conduit.

Expression schématique: *entrer* $\leadsto$ *dans_batiment*

Tables sémantiques. L'événement *entrer* ne peut se produire que si la personne entrant n'est pas dans le bâtiment. De manière symétrique, l'evénement *sortir* ne peut se produire que si la personne sortant est présente dans le bâtiment.
$\neg$ *dans_batiment* $\leftsquigarrow$ *entrer*
dans_batiment $\leftsquigarrow$ *sortir*

L'événement *entrer* établit le prédicat *dans_batiment* et l'événement *sortir* le rend faux :
$entrer \rightsquigarrow dans\_batiment$
$sortir \rightsquigarrow \neg\ dans\_batiment$

Il n'y a pas de candidats aux interactions entre ce besoin et les faits précédemmment formalisés.

Besoin 3.2.3. "Après lecture de la carte, si l'accès est autorisé, le voyant vert s'allume, l'ouverture de la porte est débloquée et la personne dispose de 30" pour entrer sans reblocage de la porte."

$$\forall tr : Tr;\ lect : LECTEUR;\ c : CARTE \bullet \forall i : \mathsf{dom}\, tr \mid i > 1 \wedge tr(\#tr).t > tr(i+1).t + 30 \bullet$$
$$acces\_autorise(tr(i).s, c, porte(lect)) \wedge \neg\ acces\_autorise(tr(i-1).s, c, porte(lect))$$
$$\Rightarrow voyant\_vert(tr(i+1).s, lect) \wedge debloquee(tr(i+1).s, porte(lect))$$
$$\wedge\ (\exists j : \mathsf{dom}\, tr \bullet tr(j).e = bloquer(porte(lect)) \wedge tr(j).t = tr(i+1).t + 30$$
$$\wedge\ (\forall k : i+1\,..\,j-1 \bullet debloquee(tr(k).s, porte(lect))))$$

Si l'accès n'était pas autorisé en l'état du $i-1$-ième élément du système et est autorisé en l'etat du i-ième élément, alors dans l'etat suivant du système, le voyant sera vert et la porte sera débloquée. Elle restera débloquée pendant 30".

Pour rendre possible l'expression de ce besoin, deux nouveaux prédicats sont nécessaires : *acces_ autorise* paramétré par une carte et une porte indique si l'accès est autorisé et *voyant_vert* paramétré par un lecteur indique si le voyant vert associé au lecteur est allumé. Ceci nous amène à introduire une nouvelle fonction: $porte : LECTEUR \rightarrow PORTE$ qui associe une porte à chaque lecteur.

Remarque. Cette formalisation ne décide pas quel événement se produit le premier, *allumer_vert* $(lect)$ ou $debloquer(porte(lect))$. Elle dit seulement qu'en l'état $tr(i+1).s$, les deux événements se sont produits.

Remarque. Ce besoin ne peut pas être satisfait si la personne ouvre la porte, mais ne la ferme pas ensuite (voir hypothèse 3.3.3 décrite ci-dessous).

Expression schématique: $acces\_autorise \rightsquigarrow voyant\_vert \wedge debloquee \wedge bloquer$

Tables sémantiques:
$debloquee \leftsquigarrow entrer$
$debloquee \leftsquigarrow sortir$

$allumer\_vert \rightsquigarrow voyant\_vert$
$eteindre\_vert \rightsquigarrow \neg\ voyant\_vert$
$debloquer \rightsquigarrow debloquee$
$bloquer \rightsquigarrow \neg\ debloquee$
$bloquer \rightsquigarrow \neg\ acces\_autorise$

$voyant\_vert \Rightarrow acces\_autorise$
$debloquee \Rightarrow acces\_autorise$
$\neg\ debloquee \Rightarrow fermee$

$\neg$ *fermee* $\Rightarrow$ *debloquee*

Candidats aux interactions. L'analyse des préconditions détecte les candidats $fait_1$, $fait_2$ et besoin 2.

Remarque. Il manque des conditions suffisantes pour que l'accès soit autorisé. Il n'est pas précisé quand le voyant vert doit être éteint.

Besoin 3.2.4. "Si l'utilisateur entre, son passage est détecté et le journal est mis à jour; la porte est bloquée."

$$\forall\, tr : Tr;\ c : CARTE;\ po : PORTE \bullet \forall\, i : \mathsf{dom}\, tr \mid i < \#tr \bullet \\ tr(i).e = entrer(c,po) \Rightarrow tr(i+1).e = bloquer(po)$$

Remarque. La mise à jour du journal a été prise en compte avec le besoin 2.

Expression schématique: *entrer* $\rightsquigarrow$ *bloquer*

Tables sémantiques:
entrer $\rightsquigarrow \neg$ *acces_autorise*

Candidats aux interactions. L'analyse des précondition détecte les candidtats $fait_1$, $fait_2$, 2 et 3.2.3 (car *acces_autorise* $\Rightarrow$ *debloquee* et *entrer* $\nleftarrow$ *debloquee*). L'analyse des postconditions détecte les candidats $fait_1$ et 3.2.1(2).

Il y a une interaction avec le besoin 3.2.3. On veut que, si la personne est entrée, la porte se bloque immédiatement après sa fermeture, et non seulement après 30 secondes.

Pour résoudre cette interaction, nous proposons de remplacer "$tr(j).t = tr(i).t + 30$" dans le besoin 3.2.3 par "$tr(j).t \leq tr(i).t + 30$".

Remarque. Nous constatons qu'il manque un besoin identique pour la sortie d'un bâtiment.

Besoin 3.2.5. "Si l'accès est refusé, le voyant rouge s'allume pendant 2"."

$$\forall\, tr : Tr;\ c : CARTE;\ lect : LECTEUR \bullet \forall\, i : \mathsf{dom}\, tr \mid i > 1 \wedge tr(\#tr).t > tr(i).t + 2 \bullet \\ acces\_refuse(tr(i).s,c,porte(lect)) \wedge \neg\, acces\_refuse(tr(i-1).s,c,porte(lect)) \\ \Rightarrow voyant\_rouge(tr(i).s,lect) \wedge \\ (\exists\, j : \mathsf{dom}\, tr \bullet tr(j).e = eteindre\_rouge(lect) \wedge tr(j).t = tr(i).t + 2) \\ \wedge\ (\forall\, k : i\,..\,j \bullet voyant\_rouge(tr(i).s,lect))$$

Remarques induites par la formalisation de ce besoin :

- Les prédicats *acces_autorise* et *acces_refuse* ne sont pas très clairs. Ils sont rendus vrais par l'événement *inserer_carte*, mais combien de temps est-ce qu'ils restent vrais? Concernant *acces_autorise*, nous avons décidé qu'il est rendu faux par les événements *bloquer* et *entrer*. *acces_refuse* est rendu faux après 2 secondes par l'événement *eteindre_rouge*.
- Les besoins ne précisent pas qu'en cas d'un refus d'accès la porte doit rester bloquée.

Expression schématique: $acces\_refuse \rightsquigarrow voyant\_rouge \wedge eteindre\_rouge$

Tables sémantiques:
$inserer\_carte \rightsquigarrow acces\_refuse$
$eteindre\_rouge \rightsquigarrow \neg\ acces\_refuse$

$acces\_refuse \Rightarrow voyant\_rouge$
$voyant\_rouge \Rightarrow acces\_refuse(?)$
$acces\_refuse \Rightarrow \neg\ debloquee(?)$
$acces\_refuse \Rightarrow \neg\ acces\_autorise$
$acces\_autorise \Rightarrow \neg\ acces\_refuse$

Il n'y a pas de candidats aux interactions avec l'ensemble des contraintes formalisées.

Hypothèses. Elles concernent les usagers disciplinés décrits dans le paragraphe 3.3 du cahier des charges.

Hypothèse 3.3.1. "Il est interdit de permettre l'entrée ou la sortie d'une personne qui n'a pas inséré sa carte dans le lecteur."

$$\begin{array}{l}\forall\, tr : Tr;\ c_1,c_2 : CARTE;\ lect : LECTUER \bullet\ \forall\, i,j : \mathsf{dom}\, tr \mid i < j \bullet \\ \quad tr(i).e = entrer(c_1,porte(lect)) \wedge tr(j).e = entrer(c_2,porte(lect)) \wedge \\ \qquad (\forall\, k : i+1\,..\,j-1;\ c : CARTE \bullet\ tr(i).e \neq entrer(c,porte(lect)) \\ \quad \Rightarrow (\exists\, k : i\,..\,j \bullet\ tr(k).e = inserer\_carte(c_2,lect))\end{array}$$

La formule ci-dessus exprime le fait que si les *i*-ième et *j*-ième éléments de la trace *tr* sont l'événement *entrer*, cette trace n'ayant pas d'autres événements *entrer* entre *i* et *j*, alors un événement *inserer_carte* s'est forcément produit entre les deux événements *entrer*.

Expression schématique: $\neg\ inserer\_carte \rightsquigarrow \neg\ entrer$

Un candidat aux interactions est détecté avec l'analyse des préconditions, il s'agit du besoin 3.2.1.(2).

Hypothèse 3.3.2. "Une personne qui a inséré sa carte et reçu le feu vert est tenue de franchir la porte."

$$\begin{array}{l}\forall\, tr : Tr;\ lect : LECTEUR;\ c : CARTE \bullet\ \forall\, i : \mathsf{dom}\, tr \mid 1 < i < \#tr \bullet \\ \quad acces\_autorise(tr(i).s,c,porte(lect)) \wedge \neg\ acces\_autorise(tr(i-1).s,c,porte(lect)) \\ \qquad \Rightarrow tr(i+1).e = entrer(c,po)\end{array}$$

Considérons le *i*-ième élément de la trace *tr*. La formule ci-dessus exprime le fait que dans cet état, si l'accès est autorisé et ne l'était pas dans l'état précédent, alors l'événement *entrer* se produira dans l'état suivant.

Expression schématique : $acces\_autorise \rightsquigarrow entrer$

Candidats aux interactions. Une analyse des préconditions détecte les candidats : $fait_1$, $fait_2$, 2, 3.2.3 et 3.2.4. Une analyse des postconditions détecte le candidat 3.3.1.

Il y a une interaction entre cette hypothèse et le besoin 3.2.3, car ce dernier précise qu'après un accès autorisé, if faut allumer le voyant vert et débloquer la porte. Pour résoudre cette interaction, il est nécessaire de décrire explicitement un scénario d'accès :

- $tr(i-1).e = inserer\_carte$
- $tr(i).e = allumer\_vert$, $tr(i+1).e = debloquer$ ou vice versa
- $tr(i+2).e = ouvrir$
- $tr(i+3).e = entrer$
- $tr(i+4).e = fermer$

Il est à noter que ce scénario n'est réaliste que si l'on considère un seul lecteur par porte. L'événement *eteindre_vert* a été omis dans ce scénario.

Pour résoudre l'interaction entre l'hypothèse 3.3.2 et le besoin 2, nous proposons une nouvelle version de l'hypothèse 3.3.2 :

$$\forall tr : Tr;\ lect : LECTEUR;\ c : CARTE \bullet \forall i : \mathsf{dom}\, tr \mid 1 < i < \#tr - 2 \bullet$$
$$acces\_autorise(tr(i).s, c, porte(lect)) \land \neg\, acces\_autorise(tr(i-1).s, c, porte(lect))$$
$$\Rightarrow tr(i+\mathbf{3}).e = entrer(c, po)$$

Hypothèse 3.3.3. "Une fois la porte franchie, il est obligatoire de la refermer avant le passage du suivant (ceci pour garantir qu'une seule personne a franchi la porte)."

$$\forall tr : Tr;\ c : CARTE;\ po : PORTE \forall i : \mathsf{dom}\, tr \mid i < \#tr \bullet$$
$$tr(i).e = entrer(c, po) \Rightarrow tr(i+1).e = fermer(po)$$

Expression schématique : $entrer \rightsquigarrow fermer$

Candidats aux interactions. L'analyse des préconditions détecte les candidats : $fait_1$, $fait_2$, 2, 3.2.3, 3.2.4, 3.2.5 et 3.3.2.

Il y a une interaction entre l'hypothèse 3.3.3 et le besoin 3.2.4. Pour la résoudre, il faut d'abord fermer la porte avant qu'elle puisse être bloquée : l'événement $tr(i+1).e$ dans l'hypothèse 3.3.3 est bien $fermer(po)$ et l'événement "$tr(i+1).e = bloquer(po)$" doit être réécrit en "$tr(i+\mathbf{2}).e = bloquer(po)$" dans le besoin 3.2.4.

Nouvelle version du besoin 3.2.4.

$$\forall tr : Tr;\ c : CARTE;\ po : PORTE \bullet \forall i : \mathsf{dom}\, tr \mid i+1 < \#tr \bullet$$
$$tr(i).e = entrer(c, po) \Rightarrow tr(i+\mathbf{2}).e = bloquer(po)$$

Hypothèse 3.3.4. "Si deux personnes veulent emprunter le même accès, la deuxième personne attend que la première ait terminé la procédure avant d'introduire sa carte."

Cette hypothèse est déjà prise en compte avec la formalisation de l'hypothèse 3.3.3. Elle implique qu'il ne peut pas y avoir deux événements *entrer* consécutifs.

Hypothèse 3.3.5. "Une fois la porte franchie, il est interdit de revenir sur ses pas."

Cette hypothèse est déjà prise en compte avec la formalisation de l'hypothèse 3.3.3. Elle implique qu'il ne peut pas y avoir un événement *sortir* consécutif à un événement *entrer*.

3.2 Résumé des documents obtenus

L'approche proposée conduit à la production de l'ensemble des documents suivants :

1. **Un diagramme entité-association** permettant de fixer le vocabulaire du domaine et précisant les aspects statiques identifiés (voir figure 4).
2. **Une liste d'événements classés** (voir les résultats de l'étape 3), ce classement jouant un rôle important dans le développement car l'analyste est amené à décider ce qui sera réalisé par le logiciel. Les événements décrivent la partie réactive du système.
3. **Une liste des opérations du système** correspondant à la partie transformationnelle (voir les résultats de l'étape 4).
4. **Un document informel** correspondant à la mise à jour du document de départ tenant compte des résultats de la phase d'élicitation.
5. **Une liste de fonctions et de prédicats** portant sur l'état du système informatique, introduits lors de la formalisation des besoins.
6. **Une formalisation des besoins, des faits et des hypothèse** exprimée en termes de contraintes sur les traces d'événements possibles.
7. **Des expressions schématiques des besoins, avec les candidats aux interactions.**

contraintes	expressions schématiques	candidats aux interactions	interactions
$fait_1$	$\neg$ *fermee* $\leadsto \neg$ *bloquer*	–	–
$fait_2$	$\neg$ *fermee* $\leadsto \neg$ *debloquer*	$fait_1$	–
2	*entrer* $\leadsto$ *dans_batiment*	–	–
3.2.1	*voyant_rouge* $\leadsto \neg$ *inserer_carte* $\neg$ *inserer_carte* $\leadsto \neg$ *acces_autorise*	– –	– –
3.2.3	*acces_autorise* $\leadsto$ *voyant_vert* $\wedge$ *debloquee* $\wedge$ *bloquer*	$fait_1$, $fait_2$, 2	–
3.2.4	*entrer* $\leadsto$ *bloquer*	$fait_1$, $fait_2$, 2, 3.2.1.(2), 3.2.3	3.2.3
3.2.5	*acces_refuse* $\leadsto$ *voyant_rouge* $\wedge$ *eteindre_rouge*	–	–
3.3.1	$\neg$ *inserer_carte* $\leadsto \neg$ *entrer*	3.2.1.(2)	–
3.3.2	*acces_autorise* $\leadsto$ *entrer*	$fait_1$, $fait_2$, 2, 3.2.3, 3.2.4, 3.3.1	3.2.3
3.3.3	*entrer* $\leadsto$ *fermer*	$fait_1$, $fait_2$, 2, 3.2.3, 3.2.4, 3.2.5, 3.3.2	3.2.4
4.1.2	*inserer_carte* $\wedge$ *dans_batiment* $\leadsto$ *acces_refuse*	3.2.3, 3.2.5	–

FIG. 5 – *Expressions schématiques des besoins avec les candidats aux interactions*

8. **Trois types de tables de relations sémantiques entre événements et prédicats :**
   (a) **Tables des conditions nécessaires pour les événements**

$$
\begin{aligned}
\mathit{fermee} &\leftsquigarrow \mathit{bloquer}\\
\mathit{fermee} &\leftsquigarrow \mathit{debloquer}\\
\neg\ \mathit{dans\_batiment} &\leftsquigarrow \mathit{entrer}\\
\mathit{dans\_batiment} &\leftsquigarrow \mathit{sortir}\\
\neg\ \mathit{voyant\_rouge} &\leftsquigarrow \mathit{allumer\_rouge}\\
\mathit{voyant\_rouge} &\leftsquigarrow \mathit{eteindre\_rouge}\\
\mathit{debloquee} &\leftsquigarrow \mathit{entrer}\\
\mathit{debloquee} &\leftsquigarrow \mathit{sortir}
\end{aligned}
$$

(b) **Tables des événements qui rendent vrai/faux les prédicats**

$$
\begin{aligned}
\mathit{fermer} &\rightsquigarrow \mathit{fermee}\\
\mathit{ouvrir} &\rightsquigarrow \neg\ \mathit{fermee}\\
\mathit{entrer} &\rightsquigarrow \mathit{dans\_batiment}\\
\mathit{sortir} &\rightsquigarrow \neg\ \mathit{dans\_batiment}\\
\mathit{allumer\_rouge} &\rightsquigarrow \mathit{voyant\_rouge}\\
\mathit{eteindre\_rouge} &\rightsquigarrow \neg\ \mathit{voyant\_rouge}\\
\mathit{inserer\_carte} &\rightsquigarrow \mathit{acces\_autorise}\\
\mathit{bloquer} &\rightsquigarrow \neg\ \mathit{acces\_autorise}\\
\mathit{allumer\_vert} &\rightsquigarrow \mathit{voyant\_vert}\\
\mathit{eteindre\_vert} &\rightsquigarrow \neg\ \mathit{voyant\_vert}\\
\mathit{debloquer} &\rightsquigarrow \mathit{debloquee}\\
\mathit{bloquer} &\rightsquigarrow \neg\ \mathit{debloquee}\\
\mathit{entrer} &\rightsquigarrow \neg\ \mathit{acces\_autorise}\\
\mathit{inserer\_carte} &\rightsquigarrow \mathit{acces\_refuse}\\
\mathit{eteindre\_rouge} &\rightsquigarrow \neg\ \mathit{acces\_refuse}
\end{aligned}
$$

(c) **Tables des relations entre prédicats**

$$
\begin{aligned}
\mathit{fermee}_{\Rightarrow} &= \{\}\\
\neg\ \mathit{fermee}_{\Rightarrow} &= \{\mathit{debloquee}\}\\
\mathit{dans\_batiment}_{\Rightarrow} &= \{\}\\
\neg\ \mathit{dans\_batiment}_{\Rightarrow} &= \{\}\\
\mathit{voyant\_rouge}_{\Rightarrow} &= \{\ \mathit{acces\_refuse}\}\\
\neg\ \mathit{voyant\_rouge}_{\Rightarrow} &= \{\ \}\\
\mathit{voyant\_vert}_{\Rightarrow} &= \{\mathit{acces\_autorise}\}\\
\neg\ \mathit{voyant\_vert}_{\Rightarrow} &= \{\}\\
\mathit{acces\_autorise}_{\Rightarrow} &= \{\mathit{debloquee},\mathit{voyant\_vert},\neg\ \mathit{acces\_refuse}\}\\
\neg\ \mathit{acces\_autorise}_{\Rightarrow} &= \{\ \}\\
\mathit{debloquee}_{\Rightarrow} &= \{\mathit{acces\_autorise}\}\\
\neg\ \mathit{debloquee}_{\Rightarrow} &= \{\ \mathit{fermee}\}\\
\mathit{acces\_refuse}_{\Rightarrow} &= \{\mathit{voyant\_rouge},\neg\ \mathit{debloquee},\neg\ \mathit{acces\_autorise}\}\\
\neg\ \mathit{acces\_refuse}_{\Rightarrow} &= \{\}
\end{aligned}
$$

4 Discussion sur l'approche proposée et résultats obtenus sur l'étude de cas

Cette étude de cas nous a permis de valider l'approche proposée, celle-ci ayant été utilisée dans plusieurs contextes différents : l'étude de systèmes purement réactifs comme l'ascenseur [HS99c] ou le four à micro-ondes, les aspects transformationnels ont été abordés avec l'étude d'un automate bancaire [HS99b], l'étude des interactions de services a été étudiée avec les systèmes de téléphonie. Une étude de cas plus complète vient d'être réalisée, il s'agit d'un système de contrôle

de l'éclairage d'un bâtiment proposé pour un séminaire sur l'élicitation, la documentation et la validation des besoins [1].

La méthode proposée s'est avérée adaptée au développement de cette étude de cas. Même si les étapes n'ont pas toujours été effectuées séparemment et exactement dans l'ordre donné, elles permettent de structurer le processus d'élicitation et aident à focaliser l'attention sur les points importants. Il est à noter que :

- plusieurs répétitions des différentes étapes ont été nécessaires,
- les composants statiques, réactifs et transformationnels du système ont pu être clairement distingués,
- la traçabilité entre les différents documents et les besoins est garantie.

De manière plus précise, notre méthode apporte un guide à la détection systématique des besoins manquants, des besoins incohérents, des besoins ambigus et redondants et à l'identification de détails de trop bas niveau pour la phase d'élicitation des besoins.

Détection des besoins manquants. Pour détecter systématiquement des besoins manquants, nous considérons des symétries : pour chaque événement et chaque opération, nous regardons s'il existe une situation duale. Si celle-ci n'est pas mentionnée dans le cahier des charge, cela peut correspondre à des besoins manquants.

Étude de cas. Ici, nous avons détecté plusieurs asymétries:

- Il y a un événement *inserer_carte*, mais pas d'événement *ejeter_carte*. Ceci nous amène à poser la question suivante : que fait le système avec les cartes inserées?
- Une deuxième asymétrie concerne l'accès et la sortie d'un bâtiment. Plusieurs besoins précisent l'accès à un bâtiment, mais aucun besoin ne précise la sortie d'un bâtiment. Aucun protocole de sortie d'un bâtiment n'est exprimé. Cette asymétrie apparaît aussi dans le prédicat *dans_batiment*: le besoin 2 specifie quel événement rend vrai ce prédicat, mais il manque un besoin qui specifie quel événement le rend faux.
- Il est à noter que les prédicats *acces_autorise* et *acces_refuse* sont seulement présents dans les préconditions des implications qui formalisent les besoins. Ceci montre que le cahier des charges ne donne pas de conditions suffisantes pour que l'accès soit autorisé ou refusé.

Détection des besoins incohérents. Notre algorithme de calcul des candidats aux interactions pour chaque nouvelle contrainte aide à détecter de manière systématique les incohérences entre les différents besoins. Cet algorithme est heuristique dans le sens ou il n'existe pas de définition précise de ce qu'est une interaction : les comportements indésirables relèvent du choix du client. Par conséquent, nous ne pouvons pas garantir que toutes les interactions seront détectées par notre algorithme. Peu d'approches se préoccupent de cette analyse des interactions, excepté des travaux autour de Kaos [LL98].

Étude de cas. Une analyse complète de l'exemple conduit à considérer 66 couples de besoins. L'algorithme de recherche des interactions a calculé 25 candidats, soit un gain de 62%.

1. La présentation de l'étude de cas ainsi qu'un résumé de nos résultats sont disponibles à l'adresse http://www.iese.fhg.de/Dagstuhl/seminar99241

Détection des besoins ambigus. La formalisation des besoins force l'analyste à les exprimer d'une manière très detaillée. Souvent, une formalisation n'est pas possible parce que l'information dans le cahier des charges est insuffisante. Dans ce cas, il faut contacter le client en vue de désambiguïser les besoins.

Étude de cas.

- Le paragraphe 3.1.4 est ambigu : qu'est-ce qui est détecté? S'agit-il du passage d'une personne, de l'ouverture de la porte ou des deux? Effectuer cette distinction est important pour le classement des événements. Mais il semble que cette question soit résolue dans le paragraphe 4.2.1, ou les capteurs sont décrits.
- La signification de accès autorisé et accès refusé n'est pas claire. Combien de temps durent ces états? Est-ce que ce sont des états ou des événements?
- Combien de lecteurs y a t-il par porte? S'il y en a deux, il faut préciser le protocole correspondant.

Détection des besoins redondants. Si l'analyste utilise plusieurs fois les mêmes formules dans la formalisation des besoins, des redondances sont présentes. Dans un document exprimé en langage naturel, des redondances peuvent être souhaitables, mais pas dans des formalisations. Les redondances identifiées par l'analyste devraient être discutées avec le client pour éviter des malentendus.

Étude de cas.

- Les besoins 2 et 3.2.4 exigent tous les deux que le journal soit mis à jour.
- Les hypothèse 3.3.4 et 3.3.5 sont déjà prises en compte avec la formalisation de l'hypothèse 3.3.3.

Identification de détails de trop bas niveau. De temps en temps, les cahiers des charges contiennent des détails inutiles pour l'élicitation des besoins. De tels détails devraient être enlevés du cahier des charges et rajoutés plus tard, lors des étapes de conception du système informatique.

Étude de cas.

- Les détails sur les identificateurs des cartes donnés en 3.1.1.(b) sont de trop bas niveau pour l'élicitation des besoins.
- Les informations sur le bus et les accusés de réception données dans le paragraphe 4.2 sont de bas niveau.

Références

[AP98] Antón (A. I.) et Potts (C.). – The use of Goals to Surface Requirements for Evolving Systems. *In : Proc. of the 20 th International Conference on Software Engineering, ICSE'98*. – Kyoto, Japan, 1998.

[CAB+94] Coleman (D.), Arnold (P.), Bodoff (St.), Dollin (Ch.), Gilchrist (H.), Hayes (F.) et Jeremaes (P.). – *Object-Oriented Development : The Fusion Method*. – Prentice Hall, 1994.

[FS97] Fowler (M.) et Scott (K.). – *UML distilled. Applying the standard Object Modelling Language*. – Addison-Wesley, 1997.

[HS98] Heisel (M.) et Souquières (J.). – A heuristic approach to detect feature interactions in requirements. *In : Proc. 5th Feature Interaction Workshop*, éd. par Kimbler (K.) et Bouma (W.). pp. 165–171. – IOS Press Amsterdam, 1998.

[HS99a] Heisel (M.) et Souquières (J.). – *Elicitation des besoins pour le système de contrôle d'accès*. – Rapport technique, LORIA, juillet 1999.

[HS99b] Heisel (M.) et Souquières (J.). – A Method for Requirements Elicitation and Formal Specification. *In : Proceedings of the 18th International Conference on Conceptual Modeling*, éd. par Springer Verlag (LNCS). – November 1999.

[HS99c] Heisel (M.) et Souquières (J.). – De l'élicitation des besoins à la spécification formelle. *TSI*, vol. 18, n7, 1999. – a paraître.

[JZ95] Jackson (M.) et Zave (P.). – Deriving Specifications from Requirements : an Example. *In : Proceedings 17th Int. Conf. on Software Engineering, Seattle, USA*. pp. 15–24. – ACM Press, 1995.

[LL98] Lamsweerde (A. Van) et Letier (E.). – Integrating Obstacles in Goal-directed Requirements. *In : Proc. of the 20 th International Conference on Software Engineering, ICSE'98*. – Kyoto, Japan, 1998.

[RBP+91] Rambaugh (J.), Blaha (M.), Premerlani (W.), Eddy (F.) et Lorensen (W.). – *Object-Oriented Modeling and Design*. – Englewood Cliffs, New Jersey, 1991, *Prentice-Hall International*.

[Spi92] Spivey (J. M.). – *The Z Notation – A Reference Manual*. – Prentice Hall, 1992, 2nd édition.

[ZJ97] Zave (P.) et Jackson (M.). – Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, vol. 6, n1, January 1997, pp. 1–30.

A Expression formelle des constraintes sur les traces

Pour décrire la manipulation des traces, nous utilisons la notation Z [Spi92] comme métalangage.

$[STATE, EVENT, TIME]$

TraceItem

$s : STATE$
$e : EVENT$
$t : TIME$

Chaque trace du système est une suite d'items, où un événement dans cette suite ne peut se produire à un temps t inférieur à celui d'un événement le précédant. Le symbole $\leq_t$ désigne une relation d'ordre partiel sur le temps :

$$TRACE : \mathbb{P}(\text{seq } TraceItem)$$
$$\forall tr : TRACE \bullet \forall i : \text{dom } tr \bullet i = \#tr \vee (tr\, i).t \leq_t (tr(i+1)).t$$

Soit Tr l'ensemble des traces possibles d'un système. Les contraintes s'expriment à l'aide de formules restreignant l'ensemble Tr. Le préfixe d'une trace est aussi une trace :

$$Tr : \mathbb{P}\, TRACE$$
$$\forall tr : Tr \bullet (\forall tr' : TRACE \mid tr' \text{ prefix } tr \bullet tr' \in Tr)$$