

Université de Savoie

Ecole Supérieure d'Ingénieurs d'Annecy

Master Recherche

Sciences & Technologies

Mention

Systemes Intelligents

Spécialité

Informatique et Traitement d'Information

Commander & Contrôler des Ascenseurs Sûrs

Travail réalisé par Nidhal RJAIBI©

Tables des matières

1.	<i>Introduction</i>	3
2.	<i>Etude détaillé du cahier des charges</i>	3
2.1.	Objectifs	3
2.2.	Etude du cahier des charges	3
3.	<i>Identification des événements, actions, et interactions d'intérêt</i>	8
3.1.	Les événements	8
3.2.	Les actions	9
3.3.	Les interactions	10
4.	<i>Définition des processus primitifs</i>	11
4.1.	Définition des processus en FSP	11
4.2.	Composition des processus	12
5.	<i>Définition des propriétés</i>	12
5.1.	Identification des propriétés de sûretés	12
5.2.	Définition des propriétés de sûretés en FSP	13
5.3.	Identification des propriétés de progrès	13
5.4.	Définition des propriétés de progrès en FSP	14
6.	<i>Validation et vérification avec LTSA</i>	14
7.	<i>Interprétation du modèle dans les situations extrêmes</i>	15
8.	<i>Conclusion</i>	15

Liste des figures

<i>Figure1:</i>	<i>Exemple de Capteur anti-blocage de la porte de l'ascenseur</i>	5
<i>Figure2:</i>	<i>Synchronisation des portes dans le cas d'une montée dans l'ascenseur</i>	5
<i>Figure3:</i>	<i>exemple du commande de l'ascenseur « Déplacement de l'ascenseur entre les étage [0..2]</i>	8
<i>Figure4:</i>	<i>Exemple d'événements</i>	9
<i>Figure5:</i>	<i>Exemple d'actions</i>	10
<i>Figure6:</i>	<i>Interactions Ascenseur & Utilisateur</i>	11

1. Introduction

Il s'agit d'étudier la conception d'un logiciel pour le contrôle et la commande de plusieurs ascenseurs d'un immeuble. Donc il s'agit d'une part d'un *logiciel distribué* et d'autre part d'un logiciel qu'on peut le considérer comme *critique* puisqu'il peut mettre la vie des êtres humains en danger.

L'ascenseur est utilisé par l'homme donc en cas d'accident les dégâts sont principalement humains. Alors il faut une bonne modélisation du système pour pouvoir construire un logiciel sûr ; dont son comportement est prévisible et surtout essayer de le vérifier et de le valider avant de le mettre en service.

2. Etude détaillée du cahier des charges

2.1. Objectifs

Le but et de pouvoir contrôler plusieurs ascenseurs d'un immeuble, je vais essayer au début d'analyser le comportement d'un seul ascenseur, ensuite je vais modéliser en FSP le fonctionnement toujours d'un seul ascenseur et enfin essayer, par composition parallèle, je vais faire modéliser deux ou plusieurs ascenseurs.

Notre système fait intervenir plusieurs mécanismes divers, par exemple une partie mécanique, une autre électrique et à ne pas oublier la partie logicielle qui va commander efficacement tous les dispositifs.

Comme d'habitude, je vais commencer par détailler le cahier des charges d'une façon analytique dont le but et de mieux comprendre le fonctionnement de l'ascenseur.

2.2. Etude du cahier des charges

1) La cage à l'intérieur de l'ascenseur :

Cette cage qui permet de transporter les utilisateurs d'un étage à un autre doit être un milieu sûr c'est-à-dire une cage solide, non déformable, étanche, légère, présence des bras de sécurité, structure contre feu...

Pour les boutons à l'intérieur de l'ascenseur dont le rôle est de permettre aux personnes dans l'ascenseur de se déplacer d'un étage à un autre.

On peut penser à minimiser les boutons par exemple au lieu de mettre tous les numéros des étages d'un immeuble (un immeuble peut contenir plus que 100 étages) on peut tout simplement prévoir un petit écran tactile de 15*15 cm pour permettre la saisie du numéro de l'étage puis deux boutons « Valider » et « Annuler » .

A ne pas oublier le test sur la validité du numéro d'étage et l'éclairage de l'écran de saisie. En ce qui concerne l'éclairage de la cage : la cage est éclairée tant qu'elle est en mouvement **OU** bien il y a des personnes dedans.

On peut penser aussi à annuler au fur et à mesure une montée ou une descente de l'ascenseur : une personne a pris l'ascenseur du premier étage vers le dernier étage mais au milieu de la course il se rappelle qu'il a oublié quelque chose dans sa voiture au parking.

→ Donc il peut aisément annuler la montée et ajouter une autre tâche à la liste des tâches de l'ascenseur c'est-à-dire descendre au parking.

Les boutons d'ouverture et de fermeture de la porte on peut ne pas les mettre parce que d'une part on peut automatiser ces deux actions et d'autre part ces boutons présentent le jeu numéro UN pour les enfants d'où le gain du facteur temps qui est important pour l'utilisation de l'ascenseur.

2) La porte de la cage :

La porte est le seul et unique accès entre la cage et la porte de l'étage. Donc il faut que la porte reste verrouillée tant que l'ascenseur soit en mouvement.

Je pense que la sortie de secours n'est pas nécessaire pour des raisons de sécurité pour les personnes utilisant l'ascenseur c'est mieux de rester dans la cage !

Un petit système pour ne pas bloquer la porte de la cage doit être mise en place par exemple une grille de rayons lasers (pour couvrir toute les tailles et tous les objets) voir figure suivante.

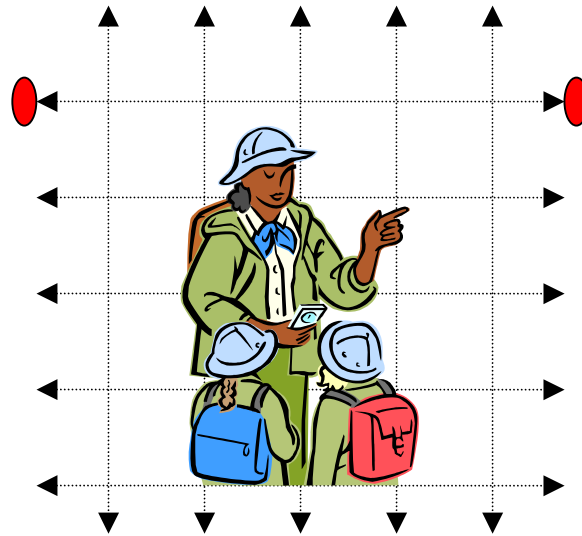


Figure1: Exemple de Capteur anti-blocage de la porte de l'ascenseur

→ Tant qu'il y a un obstacle la porte s'ouvre.

→ Tant que la porte est ouverte l'ascenseur est bloqué.

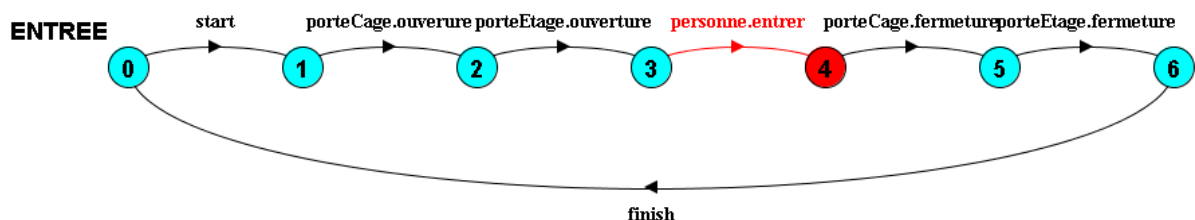
3) La porte de l'étage-ascenseur :

Cette porte est le seul accès entre l'étage et l'ascenseur. Donc il faut que cette porte ne s'ouvre que si toutes les conditions si dessous sont vraies :

- ❖ L'ascenseur n'est pas en mouvement.
- ❖ L'ascenseur *ne va pas faire* un mouvement c'est-à-dire en position d'attente.
- ❖ La porte de la cage est déjà ouverte.

Normalement avant de fermer et d'ouvrir la porte il y a un certain nombre de condition à vérifier mais je vais supposer que toutes les conditions sont vérifiées.

Figure2: Synchronisation des portes dans le cas d'une montée dans l'ascenseur



4) Les capteurs :

Ces « Flags » permettent d'intercepter des modifications sur plusieurs propriétés de l'ascenseur. Ces Flags sont binaires de valeur « True/False » ou bien des intervalles [0..100]

5) Les boutons à l'extérieur de l'ascenseur :

On peut penser à un seul bouton pour appeler l'ascenseur avec un petit écran pour afficher sous forme d'une petite animation l'état de l'ascenseur (activité, étage en cours, estimation du temps d'attente...)

6) Le moteur :

Le moteur électrique va permettre le déplacement de la cage de l'ascenseur ; Le mouvement est vertical (Haut&Bas).

Et comme le moteur utilise l'énergie électrique pour fonctionner on peut prévoir une petite batterie pour alimenter l'ascenseur en cas de coupure d'électricité si cette dernière est maintenue et le niveau de la batterie atteint un niveau minimum le processus d'urgence est déclenché et le rôle de la batterie est de :

- ❖ Maintenir l'éclairage de l'ascenseur.
- ❖ Permettre le déplacement de l'ascenseur vers l'étage le plus proche.
- ❖ Ouvrir les deux portes pour permettre l'évacuation.
- ❖ Bloquer l'ascenseur à cet étage
- ❖ Dès que l'ascenseur est vide fermer les deux portes.
- ❖ Transmettre un appel de dépannage au responsable de sécurité.

Après le retour du courant électrique et automatiquement déclenchement du processus du chargement de la batterie de secours dès que le niveau de la charge atteint le niveau minimum alors l'ascenseur revient au fonctionnement normal tout en terminant la recharge de la batterie de secours.

7) Les capteurs à chaque étage :

Les capteurs permettent la détection d'une situation particulière comme l'arrivée à un étage. Généralement les capteurs ont un rôle très important dans la construction des *systèmes intelligents* donc c'est bien de les modéliser et surtout de vérifier leurs fonctionnements parce que un mauvais capteur implique une fausse information implique un horrible comportement du système !

Par exemple les capteurs au niveau des étages permettent de :

- ❖ Réguler la vitesse de l'ascenseur.
- ❖ Vérifier la succession des étages.
- ❖ Permettre la *confirmation* de l'ouverture des portes.

Liste des capteurs :

Déclenchement
d'un événement
à chaque
passage de la
cage.

- ❖ Capteur de Surcourse Haut et Bas.
- ❖ Capteur pour Ralentir la descente.
- ❖ Capteur pour Ralentir la montée.
- ❖ Capteur pour Stopper la descente ou la montée.
- ❖ Capteur d'anti blocage de la porte.
- ❖ Capteur de mouvement dans l'ascenseur pour s'assurer qu'il y a des personnes dans la cage. Aussi pour contrôler l'éclairage dans l'ascenseur.
- ❖ Capteur porte ouverte ou bien fermée.
- ❖ Capteur de fumé (risque d'incendie).
- ❖ Capteur pour le poids.
- ❖ Capteur pour la température.

8) Le centre de contrôle et de commande

Tout simplement c'est la partie centralisée de chaque ascenseur permettant l'administration de ce dernier. Aussi un centre de contrôle et de commande pour tous les ascenseurs de l'immeuble.

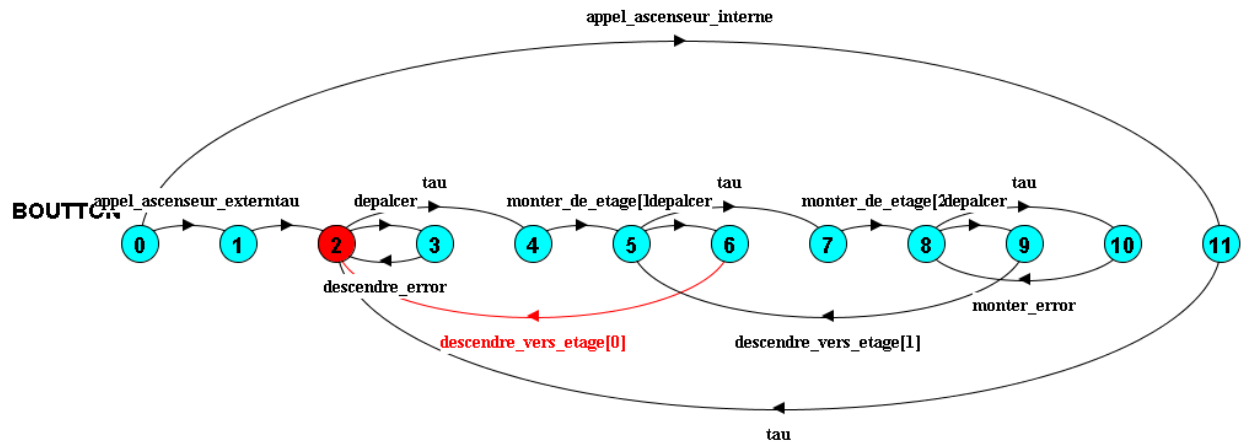


Figure3: Exemple de commande d'un ascenseur « Déplacement de l'ascenseur entre les étages [0..2]

3. Identification des événements, actions, et interactions d'intérêt

3.1. Les événements

On entend parler par événement un fait qui arrive ou qui se produit, ce qui nous permet de distinguer ces deux éléments :

- ❖ *L'émetteur* c'est-à-dire l'objet qui crée ou bien qui produit l'événement.
- ❖ Un ou plusieurs *récepteurs* qui réagit suite à la production de l'événement.

Normalement les capteurs se sont eux responsables de produire des événements qu'on peut citer quelques-uns :

Événement
Appel ascenseur à partir d'un bouton extérieur
Appel pour accélérer suite à un arrêt
Appel pour ralentir pour un éventuel arrêt
Batterie de secours au seuil minimum
Batterie de secours chargé
Batterie de secours déchargé
Bug logiciel
Coupure courant
Détection de fumée dans la cage
Panne électrique
Panne mécanique
Porte de l'étage bloqué
Porte de l'étage fermé
Porte de l'étage ouverte
Porte de la cage bloquée
Porte de la cage fermée
Porte de la cage ouverte
Stop étage à servir
Stop Sur course
Surcharge ascenseur
Température cage baisse
Température cage élevée
Température du moteur élevée
Un bouton interne activé

Figure4: Exemple d'événements

3.2. Les actions

Les actions se sont les opérations réalisées par l'ascenseur (par ses composants comme le moteur) et qui sont ordonnancés par la carte de contrôle. Une petite complication consiste à la composition des actions : Une action peut se composer d'un certain nombre actions qui peuvent à leur tour comporter d'autres actions. Mais je vais supposer que les actions citées au-dessus sont plus au moins élémentaires.

Le tableau suivant présente un exemple d'actions.

Action
Batterie de secours Activer la charge
Batterie de secours Désactiver la charge
Bouton externe Activé
Bouton interne Activé
Circuit de secours Activer
Circuit de secours Désactiver
Lumière Activer
Lumière Désactiver
Moteur accélérer
Moteur décélérer
Moteur démarrer
Moteur maintenir vitesse constante
Moteur Stop
Porte cage Fermer
Porte cage Ouvrir
Porte étage Fermer
Porte étage Ouvrir
Tempo fermeture porte Initialiser
Tempo fermeture porte Stopper

Figure5: Exemple d'actions

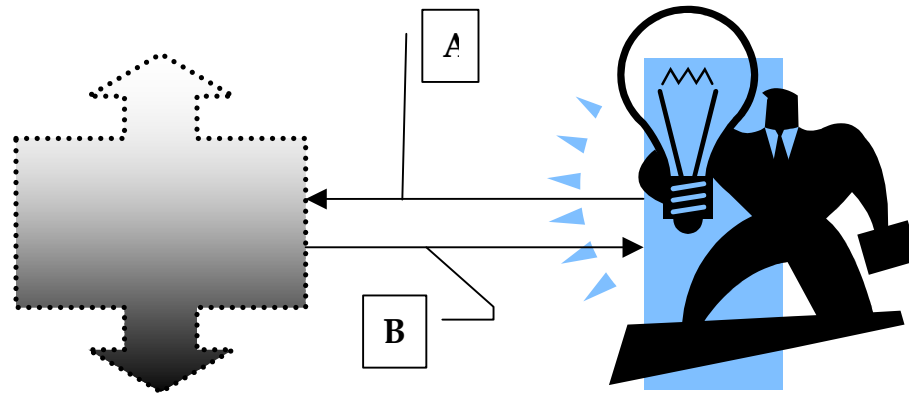
Comme le moteur c'est le responsable du déplacement de la cage je ne vais pas associer des actions à la cage car si le moteur accélère cela implique aussi que la cage accélère etc.

Mais on peut mettre un processus de vérification entre le moteur et la cage, *exemple* : Si le moteur accélère et si la cage n'avance pas alors un problème est survenu et pour éviter le chauffage du moteur (moteur bloqué) il faut envoyer une erreur de déplacement de la cage au circuit de commande.

3.3. Les interactions

Les interactions décrivent les échanges entre un ensemble d'objets on peut ici parler des interactions entre le système et l'utilisateur.

L'utilisateur ici est la personne située soit à l'intérieur ou à l'extérieur de l'ascenseur et le système et l'ascenseur avec tout ses composants :



A : Emettre une requête → Demander un service → Aller vers un autre étage.

B : Servir les requêtes utilisateurs → Fournir un service → Transporter des personnes.

Figure6: Interactions Ascenseur & Utilisateur

4. Définition des processus primitifs

4.1. Définition des processus en FSP

```
//Batterie de secours
//Niveau de la charge
range NiveauCharge = 0..10
//niveau de charge minimum
const NiveauChargeMinimum =5

//nombre des étages de l'immeuble
const NbrEtage = 2
// Intervalle de passage de l'ascenseur
range Etage = 0..NbrEtage
//Charge max de l'ascenseur
const ChargeMax =5
//Nombre personne dans l'ascenseur
range NbrPersonne = 0..ChargeMax

//*****
//UN_PROCESSUS[i][j]
//[i]==>Etage courant
//[j]==>Nombre de personne dans la cage
//*****

USE_BATTERIE = (use->BATTERIE_DECHARGER),
BATTERIE_CHARGER = BATTERIE_CHARGER[0],
BATTERIE_DECHARGER = BATTERIE_DECHARGER[0],

BATTERIE_CHARGER [j:NiveauCharge] = (
when (j < 10) charger->BATTERIE_CHARGER[j+1] |
when (j == 10) batterie_chargé->USE_BATTERIE),

BATTERIE_DECHARGER [i:NiveauCharge] = (
when (i >= 10 - NiveauChargeMinimum) batterie_déchargé->BATTERIE_CHARGER[10-i] |
when (i < 10- NiveauChargeMinimum) décharge->BATTERIE_DECHARGER[i+1]).
```

4.2. Composition des processus

Pour faire fonctionner l'ascenseur il y'a plusieurs processus qui fonctionnent chacun à part « autonomes » mais parfois il a des accès concurrents ou bien des partages de flots de données, c'est pour cela qu'il faut bien modéliser, tester la composition des processus.

Voici quelques exemples de composition de processus :

USE_BATTERIE avec BOUTTON[i:Etage][j:NbrPersonne]

|| ASCENSEUR = (BOUTTON || USE_BATTERIE).

Aussi et pour faire fonctionner plusieurs ascenseur on peut les faire composer :

|| IMMEUBLE = (a:ASCENSEUR || b:ASCENSEUR).

5. Définition des propriétés

5.1. Identification des propriétés de sûretés

La sûreté est que rien de mauvais peut arriver : dans notre cas par exemple :

- ❖ L'ascenseur monte ou décent et l'une des portes est ouverte !
- ❖ L'ascenseur est en mouvement sans lumière !
- ❖ Ouverture de la porte à un niveau différent que l'étage correspondant !
- ❖ Fermeture complète de la porte sur une personne qui est entrain d'entrer !
- ❖ Ascenseur en mouvement et il n'y a pas de personne ni à l'intérieur ni à l'extérieur !
- ❖ Nombre de personnes théorique et différent au nombre de personnes réel !
- ❖ Ouverture des deux portes non synchronisées !

5.2. Définition des propriétés de sûretés en FSP

//chaque personne qui va entrer doit nécessairement sortir
property SAFE_MOVE =(entrer->sortir->SAFE_MOVE).

On lance la verification par l'outil Check Safety

```
Composition:
DEFAULT = USE_BATTERIE || BOUTTON_EXTERNE || SAFE_MOVE
State Space:
13 * 237 * 2 = 2 ** 13
Analysing...
Depth 15 -- States: 407 Transitions: 951 Memory used: 4357K
Trace to property violation in SAFE_MOVE:
je_suis_là
entrer
porteCage.ouverture
porteEtage.ouverture
personne.in
porteCage.fermeture
porteEtage.fermeture
sortir
porteEtage.ouverture
porteCage.ouverture
personne.out
personne.out
porteEtage.fermeture
porteCage.fermeture
sortir
Analysed in: 60ms
```

→ On voit ici la détection d'une violation de la propriété SAFE_MOVE : une seule entrée dans l'ascenseur mais il y a deux sorties !!

5.3. Identification des propriétés de progrès

Le *progrès* est qu'une personne qui attend l'ascenseur pour aller d'un étage A vers un étage B ne doit pas attendre à l'infini ; il faut que l'ascenseur après un délai d'attente raisonnable (on peut réfléchir à calculer ce délai) pourra servir la requête de l'utilisateur.

Aussi il faut qu'une personne dans l'ascenseur puisse arriver à sa destination après une certaine délais encore raisonnable.

Une autre propriété de progrès il faut au moins la porte s'ouvre un jour et qu'elle se ferme.

5.4. Définition des propriétés de progrès en FSP

Un exemple de propriété de progrès

```
progress ATTENTE_INFINI = {personne.in,personne.out}
```

L'ascenseur doit monter et descendre avec la même priorité

```
progress ASCENSEUR_MOVE_MONTER = {monter_vers_etage}
```

```
progressASCENSEUR_MOVE_DESCENDRE = {descendre_vers_etage}
```

Un exemple de l'importance de l'outil d'analyse (Check Progress) du logiciel LTSA. Pour les deux propriétés précédentes l'une est vérifiée et l'autre pas :

```
Compiled: USE_BATTERIE
Compiled: BOUTTON_EXTERNE
Composition:
DEFAULT = USE_BATTERIE || BOUTTON_EXTERNE
State Space:
13 * 237 = 2 ** 12
Progress Check...
-- States: 3081 Transitions: 7020 Memory used: 6386K
Finding trace...
Progress violation: ASCENSEUR_MOVE_DESCENDRE ASCENSEUR_MOVE_MONTER
Trace to terminal set of states:
Actions in terminal set:
{{activer, appel_externe_ascenseur, aucune_personne, batterie_chargé, batterie_déchargé, charger},
descendre_vers_etage[0..1], {décharge, entrer, error_même_etage}, etage[0..2][0..5], je_suis_là,
monter_vers_etage[1..2], personne.{in, out}, {porteCage, porteEtage}.{fermeture, ouverture}, {sortir,
surcharge_attend_le_suivant, use}}
Progress Check in: 200ms
```

6. Validation et vérification avec LTSA

La validation ce fait par les outils du logiciel :

- ❖ Utiliser le Check Safety
- ❖ Utiliser Check Progress
- ❖ Utiliser Check SuperTrace
- ❖ Vérification des propriétés déjà définies par le concepteur.
- ❖ Et comme d'habitude un petit jeu de test

7. Interprétation du modèle dans les situations extrêmes

Les situations extrêmes pour l'ascenseur sont :

- ❖ La surcharge traitée au niveau de la modélisation :

```
IN[i:Etage][j:NbrPersonne] =(
  when (j<ChargeMax)personne.in->CLOSE_IN[i][j+1] |
  when (j<ChargeMax)personne.in->IN[i][j+1] |
  when (j==ChargeMax)surcharge_attend_le_suivant->CLOSE_IN[i][j]),
```

- ❖ Le sur chauffage du moteur suite à un fonctionnement de longue durée dépend des caractéristiques techniques du matériel : On peut soit le modéliser dans notre système soit ajouter un processus SECOURS qu'on peut le déclencher suite à une interruption (comme moteur bloqué) .
- ❖ Le blocage de la porte ; je n'ai pas trouvé une solution pour trouver est qu'il s'agit d'un blocage temporel de la porte ou bien d'une panne, une idée est de faire basculer tout le système vers le processus SECOURS après une attente de quelques minutes.
- ❖ Bien gérer la liste des taches de l'ascenseur (dans le cadre de ma modélisation) j'ai supposé qu'à l'intérieur de la cage il n'y a pas des boutons pour *jouer avec* mais un petit écran pour saisir, valider ou annuler les courses.

8. Conclusion

L'approche de construction des logiciels répartis par modélisation présente beaucoup d'avantages surtout lorsqu'il s'agit d'un model formel. Cette approche nous permet de définir d'une façon claire et non ambiguë la *structure* de tout système et surtout les *relations* entre ses composants.

Pour pouvoir *définir* le modèle ou bien l'ensemble des *processus* et des *propriétés* décrites dans le langage *FSP*; les outils du logiciel *LTSA* présentent beaucoup d'atouts pour *vérifier*, *valider* et *animer* notre modèle avant de l'implémenter. Cette approche associée à un outil puissant comme *LTSA* nous permet de valider notre conception et de pouvoir changer (adaptation, perfection, changement des besoins de l'utilisateur, évolution...)le fonctionnement du système voir son *comportement* d'une façon itérative et toujours sans exécutable !