INGI 2251 Software Engineering

Synthèse

Julien Henricot

2012-2013

Synthèse de Julien Henricot sur le cours INGI 2251. Cette synthèse est basée sur les slides du cours. Cette synthèse pas complète a 100%. Mais je la mets déjà en ligne pour les Quizz.

Table des matières

1.1. Software engineering : Quoi : 4
1.2. Software engineering : Pourquoi : 4
1.3. Software engineering : Comment : 4
2.1. Cycle de vie statique : 5
2.1.1. Requirements Engineerings : 5
2.1.2. Software Design : 6
2.1.3. Software implementation : 7
2.1.4. Maintenance : 7
2.2. Cycle de vie Dynamique : 8
2.2.1. Waterfall model : 8
2.2.2. V model : 8
2.2.3. Incremental model : 9
2.2.4. Exploratory model : 10
2.2.5. Spiral model : 10
2.3. Facteur de qualité du programme : 11
2.4. Estimation des coûts et des horaires : 11
2.5. Spécifications dans le cycle de vie d'un programme : 12
3.1. Introduction : 13
3.1.1. Motivation: le problème des exigences 13
3.1.2. Le monde du problème et la solution de la machine : 13
3.1.3. Model-Driven RE: 14
3.2. Modeling system Goals : 15
3.2.1. The goal model : 15
3.2.2. Goal categories : 15
3.2.3. Goal Types : 16
3.2.4. Goal satisfaction : 16
3.2.5. Goal Link : 17
3.2.6. Building Goal Model : 17
Règles heuristiques : 18
Précautions : 19
Utilisation de modèles préconçus : 19
3.3. Risk : 20
3.3.1. Analyse des obstacles: 20
3.3.2. Identification d'obstacle : 20
3.3.3. Evaluation et résolution des obstacles : 20
3.3.4. Solutions alternatives : 20
3.4. Scenarios : 21
3.4.1. Modélisation d'interactions entre agents avec des scénarios 21
3.4.2. Les scénarios sous forme de diagramme UML : 21
3.4.3. Scénarios vs objectifs: bénéfices complémentaires 21
3.5. Conceptual object : 22

1. Chapitre 1 – Introduction :

1.1. Software engineering : Quoi :

Ce concentré sur les produits et les processus dans le cycle de vie des grands logiciels.
Grand logiciel =

- De nombreuses personnes impliquées
- Longue durée de vie (temps de développement + temps de fonctionnement)
- Plusieurs versions (versions et révisions)

1.2. Software engineering : Pourquoi :

Crise des softwares due :

- A des grandes applications,
- Mauvaise ingénierie des exigences (requirements engineering)
- Manque de méthodes systématiques et d'outils bien établis,
- Sous-estimation des aspects non techniques,
- Répartition inadéquate des efforts / coûts.

1.3. Software engineering : Comment :

Grace à :

- Des modèles, des langues,
- des méthodes, des techniques,
- des outils automatisés

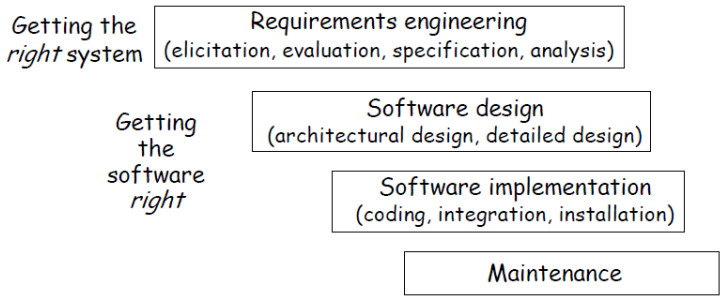
Pour :

- L'industrie et la production à grande échelle.
- Des produits logiciels de haute qualité («qualité» à définir) à faible coût.

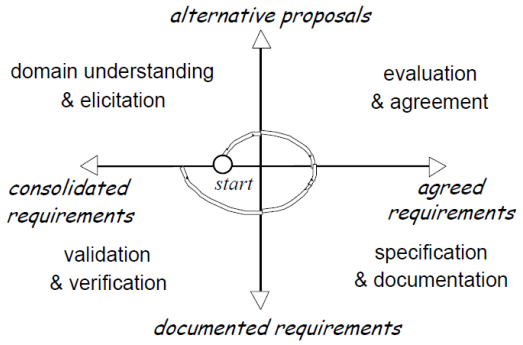
2. Chapitre 2 – Le cycle de vie d'un programme :

2.1. Cycle de vie statique :

Plusieurs étapes de SE, qui fournissent différents produit liés par des dépendances de données sans tenir compte de leur composition séquentielle ou parallèle



2.1.1. Requirements Engineerings :



1. **Compréhension du domaine & élicitation des exigences :**
   - Etudier le système en l'état.
   - Identifier les acteurs du système.
   - Explorer le monde du problème.
   - Analyse approfondie du problème (symptômes, causes, conséquences)
   - Analyse des possibilités technologiques.
   - Identification de :
     - Des objectifs d'amélioration
     - Organisation / technique contraintes sur le système.
     - Des options alternatives pour satisfaire les objectifs.
     - Des scénarios hypothétiques d'interaction du logiciel avec environnement
     - Exigences en matière de logiciels, les hypothèses sur l'environnement

⇨ **Fournis :** Projet de proposition préliminaire.

2. **Évaluation et accord :**
   Négociation basée sur la prise de décision ...
   - Identification et résolution des préoccupations contradictoires et des risques.
   - Comparaison des différentes options (objectifs et risques).
   - Hiérarchisation des Exigences (coût, calendrier, contraintes).

⇨ **Fournis :** Dernières sections de la proposition de projet de documentant les objectifs convenu des objectifs, les exigences, les hypothèses.

3. **Spécification et documentation :**
   - Définition précise de toutes les fonctionnalités du système convenu :
     - Objectifs, concepts, propriétés de domaine concerné, le système les exigences, les hypothèses, les responsabilités.
     - Configuration logicielle requise.
     - Variantes probables et les évolutions («enveloppe changement»).
     - Les coûts estimatifs.
   - Organisation de tout cela dans une structure cohérente
   - Documentation sous une forme compréhensible par toutes les parties

⇨ **Fournis :** Requirement Document (RD)

4. **consolidation des exigences** :
   Contrôle qualité sur le RD ...
   - Validation.
   - Vérification.
   - Les contrôles de qualités.
   - Fixation des erreurs et défauts

⇨ **Fournis :** RD consolidé

2.1.2. Software Design :

1. **La conception architecturale**
   Entrée : RD
   Fournis :
   - L'architecture logicielle
   - Documentation des décisions de conception
   - Données de test en boîte noire.
   - Le plan de conception
   - Manuel de l'utilisateur

2. **La conception détaillée**

   Entrée: pour chaque élément architectural ...

   - ses spécifications, les spécifications de ses connecteurs à d'autres composants
   - la spécification de ces composantes connexes (interfaces)

   Fournis : pour chaque élément architectural :

   - Algorithmes + structures de données ...
   - Documentation des décisions de conception
   - Données de test boîte blanche

2.1.3. Software implementation :

1. **Codage**

   Entrée :

   - architecture logicielle, algorithmes abstraits / structures de données
   - spécification de la plate-forme d'implémentation cible (hw / sw)

   Fournis :

   - Le code source en langage de programmation prescrite
   - Documentation des décisions de mise en œuvre

2. **Intégration :**

   Entrée :

   - Composants de l'architecture physique: spécifications, le code source
   - Données de test en boîte noire et boîte blanche

   Fournis :

   - Le logiciel testé et débogué
   - Documentation.
   - Rapport d'essai.
   - Documentation du logiciel testé & débogué

2.1.4. Maintenance :

Entrée :

- Demande de changement
- Documentation de l'ancienne version
- Informations de gestion de projet

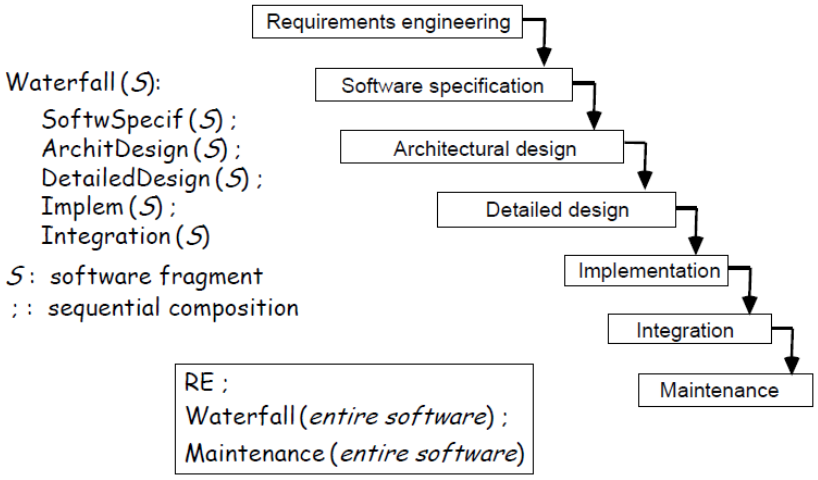
Fournis :

- Version du nouveau logiciel.
- Documentation du processus de changement.
- Rapport d'essai.
- Nouvelle version du logiciel et mode d'emploi.

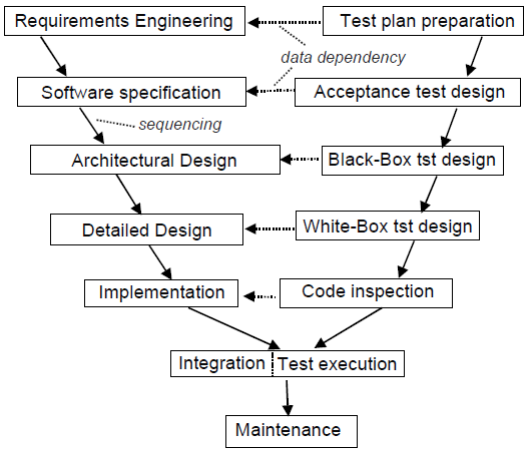
2.2. Cycle de vie Dynamique :

Les modèles de processus logiciels décrivent la manière dont les tâches SE doivent être composé (séquentiellement ou en parallèle) et comment elles interagissent (quand, dans quelles conditions).

2.2.1. Waterfall model :



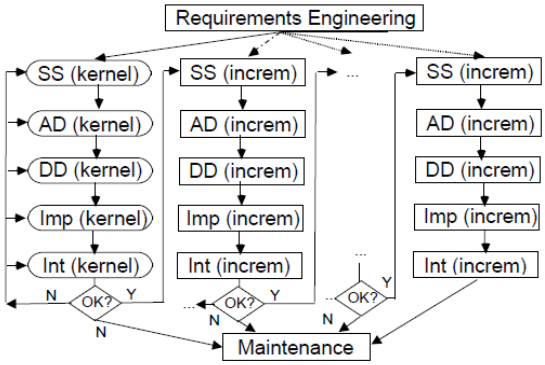
2.2.2. V model :



Caractéristiques :

- Améliore modèle de cascade en rendant explicite.
- Toujours monolithique, très limitée.
- Toujours modèle populaire dans l'industrie.

2.2.3. Incremental model :



SS: Software Specification
AD: Architectural Design
DD: Detailed Design
Imp: Implementation
Int: Integration

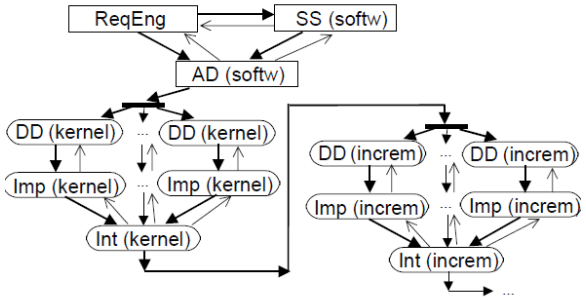
Points forts

- Rétroaction des utilisateurs plus tôt.
- Portions du produit livré plus tôt.
- Entretien Beaucoup plus tôt.
- Travail parallèle sur différents incréments rendue possible.

```
Requirements Engineering ;
S := kernel ;
while S ≠ Ø do  Waterfall (S) ;
                Get user feedback ;
                if KO then Revise (S) ;
                S := next increment od ;
Maintenance (sofware)
```

A. Waterfall (S) =
B. SoftwSpecif (S); ArchitDesign (S); DetailedDesign (S); Implem (S);
C. Integration (S)

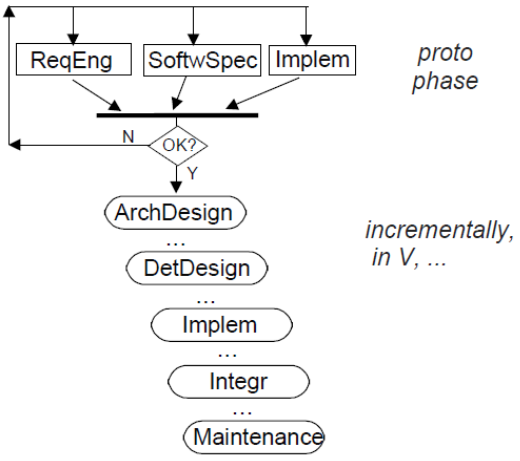
Incremental model:
variant for INGI2255 project



SS: Software Specification
AD: Architectural Design
DD: Detailed Design
Imp: Implementation
Int: Integration

sequencing
backtracking

2.2.4. Exploratory model :

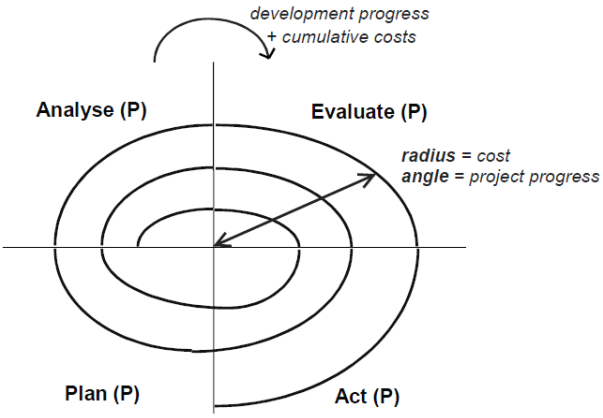



```
while Spec not OK do  RE || SoftwSpecif || Implem ;
                        {executable mock-up available}
                      Get user feedback ;
                      if KO then Revise (Spec) od ;
ArchitDesign, DetailedDesign, Implem, Integr:
                      using another model, from Spec ;
Maintenance (Software)
```

Points forts

- Permet l'élicitation des exigences cachées.
- Une meilleure garantie de l'adéquation des besoins réels spécifications.
- Autres utilisations de proto: la formation, Stubb pour les tests d'intégration.

2.2.5. Spiral model :




```
P := initial product fragment;
while P ≠ final product
   do Analyse (P) ;
      Evaluate (P) ;
      Act (P) ;
      Plan (P) od
```

Points Forts :

- Modèle générique: peut être instancié pour tout autre développement modèle (cascade, V, incrémentielle, exploratoire, ...)
- Flexible: permet de rediriger dynamiquement projet
- Axé sur les objectifs : minimiser les risques associés aux objectifs du projet
- Cumulatif

Point faible :

- Requiert plus d'expertise pour l'analyse, l'évaluation, la planification

2.3. Facteur de qualité du programme :

Différents facteur de qualité :

- Fiabilité = correcte (en accord avec RE) et robuste.
- Vérifiabilité = Facilité de contrôle ou d'analyse des propriétés des éléments de SE.
- Maintenabilité = Facilité de modification, d'extension d'articles SE.
- Imperméabilité = l'indépendance des articles de SE de la même étape.
- Traçabilité = la facilité de voir d'où un élément de SE vient, pourquoi il vient de là, et où il va.
- Visibilité = qualité de la documentation des processus d'élaboration des objets de SE.
- Réutisabilité = Capacité pour les objets/processus de SE d'être réutilisé dans différents contextes.
- Portabilité = indépendance maximale entre les éléments de SE et les plateformes hw / sw.
- Efficacité en temps et en espace.
- Utilisabilité.
- Interface utilisateur pratique.
- Lisibilité de la documentation et du manuel d'utilisation.

2.4. Estimation des coûts et des horaires :

Certaines lois expérimentales

- Coût d'entretien = 2 × coût de développement.
- Les coûts de développement = fonction (taille estimative du produit).
- La cause majeure d'écart par rapport aux estimations = changements dans l'équipe de développement.
- L'ajout de main-d'œuvre vers la fin du développement accrois encore plus le retard [Brooks].
- Les horaires peuvent être comprimé jusqu'à 25% de leurs valeur nominale mais pas plus [Boehm].

Modèle de base COCOMO :

$$MM = \mathbf{a}\ KSID^{b}$$
$$T_{dev} = 2.5\ MM^{c}$$

MM: number of man-months required

KSID: thousands of source instructions delivered

T_{dev} : optimal development time

a, b, c: parameters depending on kind of software

2.5. Spécifications dans le cycle de vie d'un programme :

Objectif :

Pour vous assurer une solution résout le problème correctement, il faut d'abord s'assurer que le problème est formulé correctement.

Différents types de spécifications :

- Lors du requirement engineering :
  - Spécifications des exigences du système.
  - Spécifications des exigences logicielles.
- Lors de la conception architecturale :
  - Spec externe des composants architecturaux.
- Lors de la conception détaillée :
  - Spec interne des composants architecturaux.

3. Chapitre 3 – Requirements Enginnering :

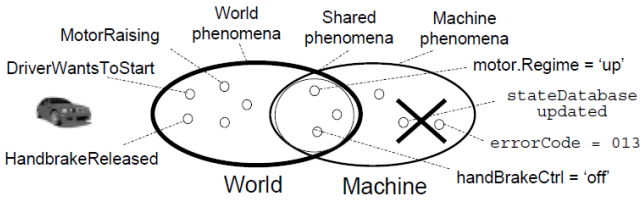
3.1. Introduction :

3.1.1. Motivation: le problème des exigences

- Les RE pauvres sont omniprésents.
- Coût élevé des corrections tardives.
- RE difficile et critique.
- Les problèmes dus au mauvais RE persistent en dépit des progrès réalisés dans la technologie des logiciels

3.1.2. Le monde du problème et la solution de la machine :

- Le monde et la machine ont des phénomènes qui leur sont propre et des phénomènes partagés.
- Les RE concernent uniquement les phénomènes du monde, y compris ceux partagés.



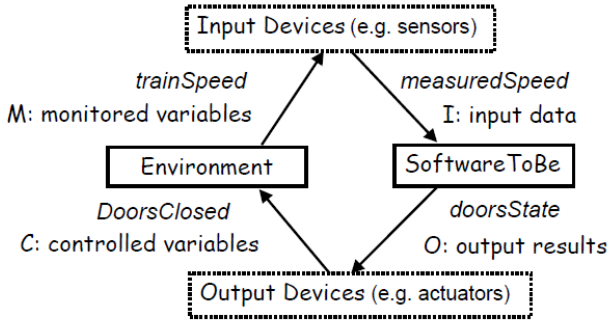
Le monde du problème implique deux versions de système :

- Système: L'ensemble d'éléments structurels en interaction dans le monde du problème.
- Système-as-is : Le système tel qu'il existe avant que l'on construise la machine.
- System-to-be: Le système tel qu'il sera lorsque la machine opérera à l'intérieur.

Exigences système vs exigences logicielle :

- Software-to-be: logiciel à développer - une partie de la machine, un composant du système-to-be.
- Environnement: tous les autres composants du système-to-be, y compris les personnes, les dispositifs, les logiciels préexistants, etc
- Exigences système: ce que le système-to-be doit réunir; formulées en termes de phénomènes dans l'environnement.
    - *Ex : « Le frein à main est libérée lorsque le conducteur veut commencer. »*
- Exigences logicielle: ce que le logiciel-to-be devra répondre à lui seul, formulée en termes de phénomène partagé par le logiciel et l'environnement.
    - Ex : « La variable logicielle de sortie handBrakeCtrl doit avoir la valeur 'off' lorsque la variable logicielle d'entrée motorRegime obtient la valeur 'up'. »

Relier les exigences logicielles aux exigences système: Le modèle à 4 variables



$SysReq \subseteq M \times C$ relation on environment monitored/controlled variables

$SofReq \subseteq I \times O$ relation on software input/output variables

$SofReq = Map(SysReq, Dom, Asm)$

translates SysReq using domain properties and assumptions

3.1.3. Model-Driven RE:

- Modèle :
  - Représentation abstraite du système (as-is ou to-be).
  - souligne, précise, relie, les caractéristiques clés du système.
- Modèle multi-vue:
  - Différentes facettes système pour une intégralité des exigences

Pourquoi des modèles pour les RE ?

- Mettre l'accent sur les principaux aspects (abstraction faite de multiples détails).
- Fournit une structure pour les activités de RE.
- Facilite l'analyse.
- Prise en charge de la compréhension, l'explication aux parties prenantes.
- Base pour la prise de décisions.
- Base pour générer le document des exigences (avec outil).

Différents modèle de RE :

- Goals.
- Risks.
- Conceptual object.
- Agents.
- Opérations.
- Threats.
- Behaviors – Scenarios.
- Behaviors - State machines.

3.2. Modeling system Goals :

3.2.1. The goal model :

- Vue intentionnel du système modélisé
- Objectif = objectif à atteindre par le système
    - Mots indicatif : ‘’shall’’, ‘’should’’, ‘’must’’, …
    - déclaration d'intention prescriptive
    - "système": logiciel + environnement

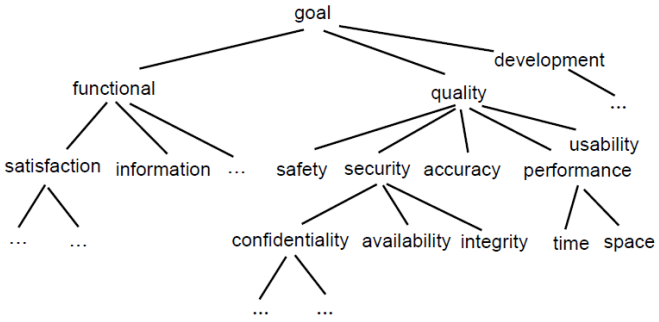
Les objectifs sont différents des propriétés de domaine :

- Propriété de domaine = état descriptif sur l'environnement
    - Mots indicatif : ‘’is’’, ‘’are’’, …
- La distinction entre les objectifs et les propriétés de domaine est essentiel pour les RE :
    - Les objectifs peuvent être négociés, affaibli, ...
    - Les propriétés de domaine ne peut pas
    - Tous deux nécessaires dans la documentation des exigences

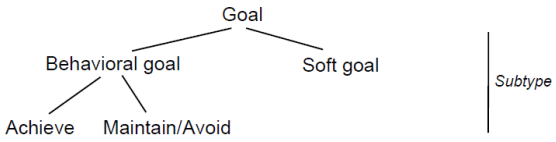
Les objectifs sont formulés à différents niveaux d'abstraction

- Des objectifs de haut niveau :
    - stratégique, à gros grains
- Des objectifs de niveau inférieur :
    - technique, à grain fin

3.2.2. Goal categories :



3.2.3. Goal Types :



Behavioral goals (objectifs comportementaux) :

- Prescris les comportements du système de façon déclarative.
- Peut être satisfaite dans un sens très clair: OUI ou NON.
- Utilisé pour construire les modèles d'opération.
- **Sous-types et les modèles de spécification :**
    - Achieve [TargetCondition]: [if CurrentCondition then] sooner-or-later TargetCondition
    - Maintain [GoodCondition]: [if CurrentCondition then] always GoodCondition always (if CurrentCondition then GoodCondition)
    - *Avoid* [BadCondition]: dual of *Maintain* ... **[**if CurrentCondition then**]** never BadCondition

Soft goals :

- Préférences entre différents comportements.
- Ne peut être satisfaite en nette sens:
    - plus satisfaits dans une option, moins satisfaits dans un autre
- Utilisé pour comparer différentes options pour sélectionner la préféré
- Prennent souvent la forme :
    - Agrandir / Réduire, Incresase / Réduire, Améliorer, ...

3.2.4. Goal satisfaction :

La satisfaction des objectifs nécessite la coopération d'agents :

- Agent = composant du système actif
    - responsable de la satisfaction objectif. Il doit limiter son comportement pour répondre à ses objectifs assignés.
    - rôle, plutôt qu'individuelle.
- Types d'agents:
    - logiciel (software-to-be, logiciels existants, logiciels étrangers).
    - dispositif (capteur, actionneur, ...).
    - humain.
- But plus fin => peu d'agents nécessaires à la satisfaction de l'objectif.
- Objectif = exigence attribué à un agent unique dans le logiciel-to-be.
- Attente = objectif assigné à un agent unique dans l'environnement.

Objectifs appellent à la satisfaction des arguments

R, E, D =| G

Compte tenu des propriétés du domaine D, les exigences R assure que le but G est satisfait sous les exigences E.

3.2.5. Goal Link :

Liaisons spécialisées liant les objectifs à

- D'autres objectifs.
- D'autres sous-modèles.

Raffinement AND/ OR :

- Un raffinement AND d'un but G en sous-buts [G_1, ..., G_n] tel que G est satisfait si [G_1, ..., G_n] est satisfait.
    - L'ensemble {G1, ..., Gn} est appelée raffinement de U.
    - Un sous-but G_i est dit de contribuer positivement à G.
- {G1, ..., Gn} est un système complet AND-refinement de G ssi satisfaire [G_1, ..., G_n] est suffisante pour que G soit satisfait, compte tenu des propriétés du domaine connu

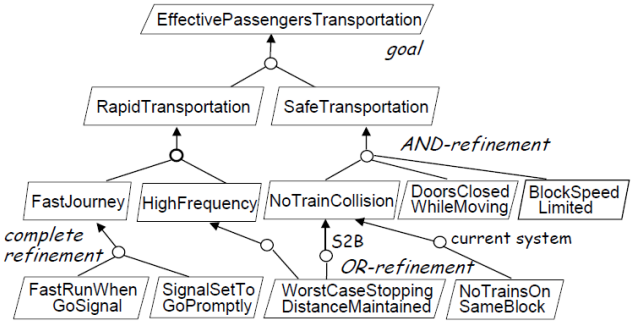
  {G1, ..., Gn, Dom} | = G
- Un raffinement OR-G de but en états de remplacement R_1, ..., R_m tel que G peut être satisfait en satisfaisant tous les sous-buts de l'un des états de remplacement R_i. raffinement Ri est appelé alternative pour G

3.2.6. Building Goal Model :

Qu'est-ce qu'un modèle de but?

- Un graphe AND/OR montrant comment les objectifs contribue l'un à l'autre :
    - Racines = objectifs de haut niveau.
    - Feuilles = les exigences ou les attentes (assignable à des agents simples).
    - Un raffinement-AND lie l'objectif parent à l'ensemble de ses sous-objectifs.
    - Un raffinement-OR lie l'objectif parent à l'ensemble des raffinements-AND.

      => Système alternatif d'options.
- En général, on a un graphe acyclique dirigé, pas un arbre :
    - racines multiples.
    - Un objectif peut contribuer aux objectifs de parents multiples.

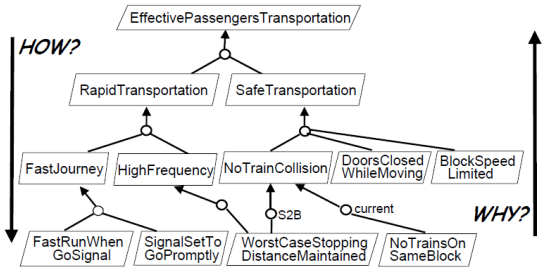


Règles heuristiques :

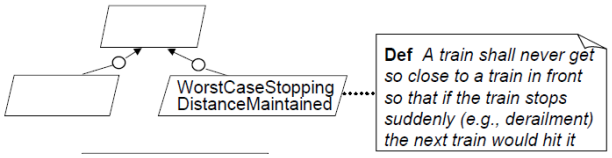
- **Découverte précoce des buts :**
  - . Analyse du système-as-is
    - Problèmes, des lacunes.
    - Objectifs du système-to-be: Éviter / Réduire / Améliorer.
  - Rechercher des mots clés intentionnels et prescriptive dans les documents disponibles, les transcriptions d'entrevues, etc …
    - - in order to, so as to, so that, ...
    - - has to, must, to be, must be, shall, ensure, want, motivate, expected to, ...
    - - purpose, objective, aim, concern, …

  Refinement link: “in order to X the system has to Y “.
- **Découverte tardives d'objectifs :**
  - Par abstraction (bottom-up) : Demander **POURQUOI**? questions au sujet de ...
    - De niveau inférieur objectifs
    - Scénarios d'interaction étant provoquée (voir plus loin)
    - D'autres matières opérationnelle disponible

      ➔ Objectifs parents.
  - Par raffinement (top-down): Demandant **COMMENT**? -> Sous-buts.
  - Par l'utilisation de modèles de raffinement (voir plus loin)
  - Par la résolution des obstacles, les conflits (voir plus loin)



- **Pour éviter toute ambiguïté dans l'interprétation des objectifs :**
  - Donner définition précise et complète objectif est essentiel
  - Bien établir les phénomènes partagés du système partagés et s’accroder dessus avec toutes les parties prenantes.



- **Vérifiez l'inverse des objectifs ‘Achieve’ disponible en tant que candidat objectif ‘Maintain’.**
  - **Achieve** [Cible **Si** Condition]: **si** condition **alors tôt ou tard** Cible

    ? ↓?

    Maintain [Cible **SSI** Condition]: **toujours** (**Si** cible **Alors** Condition)
- **Parcourir les feuilles de taxonomies des objectifs fonctionnel et non fonctionnel, et rechercher des cas spécifiques au système.**
- **Raffiner les objectifs jusqu'à ce qu’ils soient assignable à des agents simples comme :**
  - exigence (agent logiciel).
  - attente (agent de l'environnement).
- **Abstraire les objectifs jusqu'à ce que la limite des capacités du système soit atteinte:**
  - Objectifs qui ne peuvent être satisfaits uniquement par des agents du système.

Précautions :

Ne pas confondre

- Objectifs et opération. Ex : portes ouverte ssi train arrêté ≠ ouvrir porte.
- Raffinement-OR et raffinement-AND.

Utilisation de modèles préconçus :

Remplissez les modèles de raffinement-AND du catalogue (voir le livre) :

- Encoder les tactiques de raffinement.
- Guider le processus de modélisation en suggérant des améliorations à être instancié
    - modèle instancié peuvent parfois exiger l'adaptation
- Aide dans la documentation du modèle et la compréhension
- Vérification de l'exactitude de raffinement gratuitement
  ⇒ léger, pré-digéré analyse étayée

3.3. Risk :

- Problème: les objectifs sont souvent trop idéals, et risquent d'être violés dans des conditions anormales (comportements inattendus ou d'agents malveillants)
- Obstacle = Condition sur le système en cas de violation d'objectif
    - {O, Dom} | = ¬G obstruction
    - Dom | ≠ ¬ O cohérence de domaine
    - Il existe comportements environnement E tel que : E | = O faisabilité
- Cas particuliers :
    - obstruction des safetyGoals: obstacles = dangers.
    - obstruction des securityGoals: obstacles = menaces.

3.3.1. Analyse des obstacles:

Pour chaque objectif feuilles (exigence ou attente) dans le goal model :

- Lui identifier autant d'obstacles possibles.
- Lui évaluer leur probabilité et la gravité.
- Les résoudre en fonction de la probabilité et de la gravité.

3.3.2. Identification d'obstacle :

Pour les obstacles au but G ...

- Négation G;
- Trouver autant de raffinement-And/-OR de ¬ G que possible compte tenu des propriétés du domaine jusqu'à atteindre les conditions d'obstruction

⇨ Méthode de goal-anchored construction ou Arbre de défaillance.(ex : tr51-56)

3.3.3. Evaluation et résolution des obstacles :

- Pour évaluer la probabilité et la sévérité de l'obstacle identifié:
    - l'utilisation de techniques de gestion des risques (pas dans ce cours) avec des experts du domaine.
- Pour résoudre l'obstacle identifié -> Transformer le goal model en générerant des solutions alternatives.

3.3.4. Solutions alternatives :

- **Substitution de l'objectif:** envisager de préciser le remplacement de l'objectif parent pour éviter l'obstruction de l'objectif enfant.
- **Agent obstacle:** prendre en compte les responsabilités alternatives.
- **Affaiblissement de l'objectif:** affaiblir la formulation de l'objectif tel qu'il ne soit plus obstrué.
- **Restauration de l'objectif:** Appliquer une condition cible quand un obstacle survient.
- **La prévention des obstacles:** introduire de nouveaux Avoid goal.
- **L'atténuation des obstacles:** tolère l'obstacle, mais en atténue les effets.

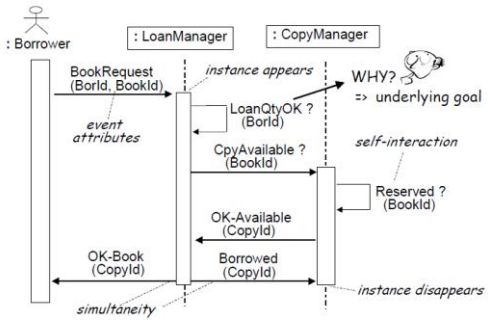
3.4. Scenarios :

3.4.1. Modélisation d'interactions entre agents avec des scénarios

- Scénario Sc =
  - Séquence historique d'événements d'interaction entre les instances d'agent
  - Illustre une manière d'atteindre l'objectif implicite G. Sc est couvert par G.
  - Les interactions sont dirigées.
    - De l'instance de l'agent source, contrôlant l'événement,
    - Vers l'instance de l'agent cible, suivant l'événement
  - Les interactions sont synchrones entre les événements de contrôle/moniteur.
- Scénario positif: le comportement souhaité (exemple typique)
  - Scénario normal: dans les cas normaux
  - Scénario anormale: dans des cas exceptionnels (n'oubliez pas de ceux-là!)
- Scénario négatif: un comportement indésirable (contre-exemple)

3.4.2. Les scénarios sous forme de diagramme UML :

- Event = objet conceptuel instantané
- Événements d'interaction correspondent à des applications d'opérations par l'agent source, notifié à l'agent cible (voir modèle de fonctionnement)
- Un diagramme de séquence définit ...
  - Un ordre partiel sur l'ensemble des événements du scénario
  - Un ordre total sur les événements le long délai d'un agent
- Il peut être composé d'épisodes (sous-scénarios).



3.4.3. Scénarios vs objectifs: bénéfices complémentaires

Les points forts des scénarios:

- . des exemples concrets
- . style narratif de la description / prescription
- . fournis des données de test d'acceptation gratuitement!

Limitations des scénarios:

- Partielle.
- Explosion combinatoire pour une bonne couverture.
- Choix prématurés.
- Buts et exigences sont conservés implicite.

⇨ Utilisation de scénarios pour le déclenchement, la validation de l'analyse des objectifs.

3.5. Conceptual object :