Chapitre 8: Diagramme de machine à états

Introduction et composants

Définition

Objectif: Illustrer le cycle de vie d'un objet à l'aide d'états et de transitions **Usage:** Modéliser le comportement d'objets dépendants de leur état **Équivalence:** Correspond aux automates finis (finite state machine)

Composants de base

État initial: Point de départ du cycle de vie **État final:** Point d'arrivée du cycle de vie **États intermédiaires:** Conditions de l'objet entre stimulus **Stimulus (événement):** Déclencheur notable d'une transition **Transitions:** Relations entre états suite à un événement

Quand les utiliser

Applications principales

Objets réactifs: Dispositifs contrôlés par logiciels (téléphone, four, ascenseur) **Protocoles:** Communications TCP/IP, sessions utilisateur **Parsers et grammaires:** Expressions régulières, langages de programmation **Transactions:** Systèmes d'information, suivi de livraisons **Navigation:** Interfaces utilisateur, sites web

Distinction état-dépendant vs état-indépendant

État-indépendant: Objet répond toujours pareil (ex: interrupteur) **État-dépendant:** Réponse varie selon l'état actuel (ex: téléphone, four)

Concepts avancés

Transitions conditionnelles

Guard conditions: Transitions conditionnées par propriétés de l'état **Syntaxe:** `événement [condition] / action` **Exemple:** Téléphone actif si utilisateur a abonnement valide

États imbriqués

Principe: États peuvent contenir des sous-états **Héritage:** Sous-états héritent des transitions de l'état parent **Avantage:** Modélisation hiérarchique du comportement

Actions et comportements

Actions do: Exécutées tant qu'on reste dans l'état (interruptibles) **Actions entry:** Exécutées à l'entrée dans l'état (non interruptibles) **Actions exit:** Exécutées à la sortie de l'état (non interruptibles) **Actions événement:** Associées à des événements (obsolètes UML)

Historique

H (shallow): Retour au dernier état de sortie **H (deep):** Retour au dernier état des sous-états imbriqués **Usage:** Mémoriser l'état précédent (ex: machine à laver)

Régions de concurrence

Principe: Plusieurs états actifs simultanément dans un état composite **Synchronisation:** Barres de synchronisation comme diagrammes d'activité **Applications:** Modélisation de processus parallèles

Exemples pratiques

Téléphone

États: Idle, Active (avec sous-états: PlayingDialTone, Dialing, Talking, Connecting) **Transitions:** off hook/on hook avec conditions d'abonnement **Héritage:** Possibilité de raccrocher depuis tous les sous-états

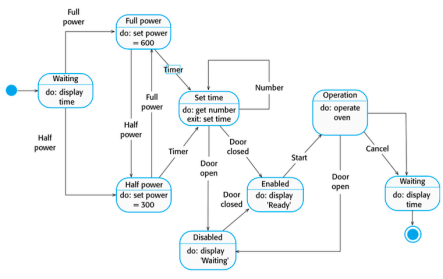
Four micro-ondes

États: Waiting, Half/Full power, Set time, Enabled/Disabled, Operation **Sécurité:** Transition vers Disabled si porte ouverte **Sous-états Operation:** Checking, Cook, Done, Alarm

Navigation web

États: Home Page, Product List Page, Product Page, Help Page **Transitions:** search, select, help, add to shopping cart **Simplicité:** Modélisation basique du parcours utilisateur

Four micro-onde (états)



State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Chapitre 9: Méthodes Agiles

Contexte et motivations

Problématique: Méthodes pilotées (cascade) trop lentes, processus lourds, difficultés d'adaptation aux changements **Objectif:** Réduire radicalement les temps de livraison, alléger les processus **Principe clé:** Le logiciel doit s'adapter aux besoins et non l'inverse

Manifeste Agile (2001)

Valeurs fondamentales

Individus et interactions: > processus et outils **Logiciels opérationnels:** > documentation exhaustive **Collaboration avec clients:** > négociation contractuelle **Adaptation au changement:** > suivi d'un plan

Caractéristiques principales

Activités entrelacées: spécification, conception, développement, validation **Livraisons fréquentes:** séries de versions validées avec parties prenantes **Automatisation clé:** tests unitaires, intégration continue, livraison continue **Feedback rapide:** cycles courts permettant évaluation fréquente

Extreme Programming (XP)

Valeurs XP

Simplicité: faire uniquement le nécessaire, maximiser la valeur **Communication:** travail collaboratif, face-à-face quotidien **Feedback:** logiciel fonctionnel, démonstrations précoces, adaptations **Respect:** chacun apporte de la valeur, expertise mutuelle **Courage:** dire la vérité, s'adapter aux changements

Pratiques clés

Client sur site: représentant utilisateur disponible plein temps **Planification incrémentale:** user stories, estimation, priorisation **Programmation en binôme:** deux développeurs, un ordinateur **Test-first programming:** tests développés avant fonctionnalités **Intégration continue:** intégrations multiples par jour, tests automatiques **Refactoring:** amélioration continue du code sans changer comportement **Propriété collective:** tous développeurs responsables de tout le code

SCRUM - Gestion de projet agile

Équipe SCRUM (≤10 personnes)

Développeurs: planification sprint, adaptation quotidienne, responsabilité mutuelle **Product Owner:** maximise valeur produit, gère backlog, formule objectifs **Scrum Master:** efficacité équipe, coach méthodologie, protège des interférences

Artefacts

Product Backlog: liste ordonnée améliorations produit (features, bugs, NFR) **Sprint Backlog:** objectif + éléments + plan d'action du sprint **Incrément:** résultat sprint, version utilisable, critères "Fini" satisfaits

Événements Sprint (1-4 semaines)

Sprint Planning: définition objectif, sélection éléments, planification **Daily Scrum:** 15min, progrès, obstacles, plan jour suivant **Sprint Review:** présentation progrès, collaboration sur suite **Sprint Rétrospective:** amélioration qualité/efficacité équipe

User Stories et Personas

Template User Story

Format: "En tant que , je veux pour que " **Caractéristiques:** tient sur carte/post-it, complétée par discussions **Hiérarchie:** Epic → Stories → Tâches (≤1 jour)

Personas

Définition: personnages fiction représentant utilisateurs **Usage:** partagés, visibles, utilisés dans user stories **Objectif:** humaniser besoins utilisateurs

Outils de suivi

Tableau Kanban: visualisation flux travail (Backlog → Doing → Review → Done) **Burndown Chart:** graphique travail restant vs temps, feedback estimation **GitHub Projects:** gestion backlog, vues multiples (Kanban, Roadmap, Tableau)

Chapitre 10: PERT - CPM

Définitions

PERT: Program Evaluation and Review Technique - outil statistique pour analyser les tâches d'un projet avec incertitude dans les estimations **CPM:** Critical Path Method - méthode pour identifier le chemin critique et le temps minimum d'exécution d'un projet **Historique:** PERT développé par l'US Navy (1958), CPM introduit (1957). Partie des certifications PMP et Six Sigma

Algorithme CPM

Étapes principales

Établir: la liste des tâches, antécédents et durées **Construire:** le réseau avec dates au plus tôt (gauche → droite) **Calculer:** les dates au plus tard (droite → gauche) **Calculer:** les marges (date plus tard - date plus tôt) **Identifier:** le chemin critique (marge = 0)

Éléments graphiques

Nœud: Date au plus tôt / Date au plus tard / Marge **Tâche:** Nom(Durée) **Tâche fictive:** Durée = 0, pour résoudre dépendances partielles **Chemin critique:** Tâches avec marge nulle

PERT - Estimations probabilistes

Estimations à trois points

o: estimation optimiste **a:** estimation probable **p:** estimation pessimiste

Calculs

Durée attendue (E): Distribution PERT: $E = \frac{o+4a+p}{6}$ **Déviation standard (SD):** $\mathrm{SD} = \frac{p-o}{6}$

Application projet

Durée projet: $E(\text{projet}) = \Sigma E(\text{tâche})$ **Écart-type projet:** $\mathrm{SD}(\text{projet}) = \sqrt{\Sigma\ \mathrm{SD}(\text{tâche})^2}$ **Hypothèse:** Non-corrélation entre estimations des tâches

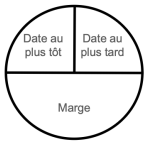
Intervalles de confiance

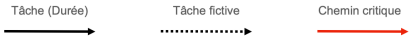
Distribution normale supposée

68.26%: E(projet) ± SD(projet) **95.44%:** E(projet) ± 2×SD(projet) **99.72%:** E(projet) ± 3×SD(projet)

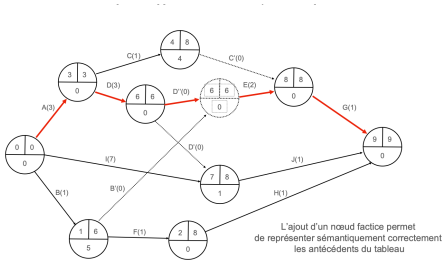
Applications pratiques

Gestion de projet: estimation des coûts et délais avec incertitude **Outils:** Microsoft Project automatise les calculs **Avantages:** quantification des risques, planification probabiliste





Tâche	Antécédent(s)	Durée
A	-	3
B	-	1
I	-	7
C	A	1
D	A	3
E	D, B	2
J	D, I	1
F	B	1
G	C, E	1
H	F	1



Chapitre 11: Testing and Refactoring

Test Driven Development (TDD)

Cycle Red-Green-Refactor

Red: Ajouter un test qui échoue pour une fonctionnalité non implémentée **Green:** Implémenter la fonctionnalité de manière simple et rudimentaire **Refactor:** Améliorer le code sans changer le comportement

Specification by example

Principe: Tests servent de spécification exécutable **Avantages:** Plus compréhensible qu'une spec, guide la conception **Approche:** Définir le comportement attendu dans les tests avant l'implémentation

Deux chapeaux distincts

Refactoring: Petits changements préservant le comportement, tests réussis **Ajouter fonctionnalité:** Nouveaux tests, peut casser tests existants

Testing - Vue d'ensemble

Pyramide des tests

Tests unitaires: Nombreux, rapides, isolés (base de la pyramide) **Tests d'intégration:** Moyennement nombreux, composants assemblés **Tests end-to-end:** Peu nombreux, difficiles, conditions réelles (sommet)

Définition test unitaire

Critères: Vérifie petit morceau de code, rapide, isolé **Classical School:** Unité de comportement, vraies dépendances autorisées **London School:** Unité de code (classe/méthode), dépendances simulées

Objectifs des tests

But principal: Assurer croissance durable du projet **Effet secondaire:** Améliorer conception du code **Indicateur:** Code difficile à tester = mauvaise qualité (couplage fort)

Pattern AAA et isolation

Structure AAA

Arrange: Préparer système testé et dépendances **Act:** Appeler méthodes, capturer résultats **Assert:** Vérifier valeurs de sortie et état final

Isolation avec doubles de test

Mock: Vérifie interactions (appels, paramètres) **Stub:** Fournit réponses prédéfinies **Avantages:** Granularité fine, identification erreurs, rapidité

Testcontainers

Principe: Créer instances dépendances externes à la demande **Usage:** Bases de données, serveurs, services d'authentification **Bénéfice:** Simplifier tests d'intégration sans mocking complexe

Refactoring

Types de refactoring

TDD refactoring: Phase du cycle Red-Green-Refactor **Litter-pickup:** Nettoyage simple et rapide **Comprehension refactoring:** Améliorer organisation avant compréhension **Preparatory refactoring:** Préparer arrivée nouvelle fonctionnalité **Planned refactoring:** Améliorer hygiène du code planifiée **Long-term refactoring:** Investissement architectural long terme

Code Smells principaux

Bloaters: Long Method, Large Class, Long Parameter List **Object-Orientation Abusers:** Mauvaise application principes OO **Change Preventers:** Modifications nécessitent changements multiples **Dispensables:** Code mort, duplication, commentaires excessifs

Exemples de refactoring

Constantes symboliques: Remplacer nombres magiques **Extraction méthodes:** Simplifier conditions complexes **Objets paramètres:** Remplacer listes paramètres longues **Builder pattern:** Simplifier constructeurs complexes

Chapitre 12: Design Patterns

Introduction et concepts fondamentaux

Origines historiques

Christopher Alexander: Concept originel appliqué à l'architecture urbaine **Gang of Four (GoF):** Application à la programmation orientée objet C++ **Définition:** Solutions réutilisables à des problèmes récurrents de conception

Structure d'un pattern

Nom: Vocabulaire commun pour les développeurs **Problème:** Description de la difficulté à résoudre **Solution:** Éléments requis pour implémenter le pattern **Conséquences:** Résultats et compromis de l'application

Catalogue des Design Patterns

Patterns de création (Creational)

Objectif: Gestion de la création d'objets **Exemples:** Singleton, Factory, Abstract Factory, Builder, Prototype

Patterns structurels (Structural)

Objectif: Composition de classes et objets en structures plus larges **Exemples:** Adapter, Composite, Proxy, Decorator, Bridge, Flyweight

Patterns comportementaux (Behavioral)

Objectif: Algorithmes et partage des responsabilités entre objets **Exemples:** Observer, Strategy, Command, Template Method, Visitor, Iterator

Pattern Visitor - Étude de cas

Problème résolu

Contexte: Ajout d'opérations à une hiérarchie de classes sans modification **Difficulté:** Éviter la "pollution" du modèle de domaine **Exemple:** Interface Geometry avec Point, Rectangle, Circle

Solution Visitor

Principe: Encapsuler opérations dans des visiteurs externes **Mécanisme:** Double dispatch (polymorphisme + liaison dynamique) **Structure:** Interface Visitor + ConcreteVisitor + méthode accept()

Architecture du pattern

Éléments du domaine: Classes métier avec méthode accept(visitor) **Hiérarchie Visitor:** Interface commune + implémentations spécifiques **Flux d'exécution:** element.accept(visitor) → visitor.visit(element)

Avantages et inconvénients

Avantages: Facilite ajout d'opérations, sépare algorithmes du modèle **Inconvénients:** Complique ajout de nouvelles classes, couplage fort **Trade-off:** Optimisé pour ajout d'opérations vs ajout de types

Considérations pratiques

Bonnes pratiques

Applicabilité: Utiliser pour problèmes récurrents bien identifiés **Prudence:** Éviter la sur-application systématique **Alternatives:** Considérer solutions custom plus simples

Limites et critiques

Sur-conception: Respecter principe KISS (Keep It Simple Stupid) **Évolution langage:** Certains patterns obsolètes (ex: pattern matching) **Contexte moderne:** Langages fonctionnels offrent alternatives élégantes

Valeur professionnelle

Vocabulaire commun: Communication efficace entre développeurs **Connaissance attendue:** Standard dans entretiens techniques **Apprentissage continu:** Lire le livre GoF reste recommandé