Linha de Produtos de Software

Rosana Braga

Parte do material elaborado pelo Prof. Paulo Cesar Masiero

Linha de Produtos de Software

- Um conjunto de sistemas intensivos de software que compartilha um conjunto de características gerenciadas e comuns, que satisfaz a necessidades específicas de um segmento de mercado em particular ou missão de uma empresa e que é desenvolvido a partir de um conjunto principal de recursos (ativos) de uma forma pré-estabelecida (Clements and Northrop 2002).

Linha de Produtos de Software

- Um conjunto de sistemas intensivos de software que compartilha um conjunto de características gerenciadas e comuns, que satisfaz a necessidades específicas de um segmento de mercado em particular ou missão de uma empresa e que é desenvolvido a partir de um conjunto principal de recursos (ativos) de uma forma pré-estabelecida (Clements and Northrop 2002).

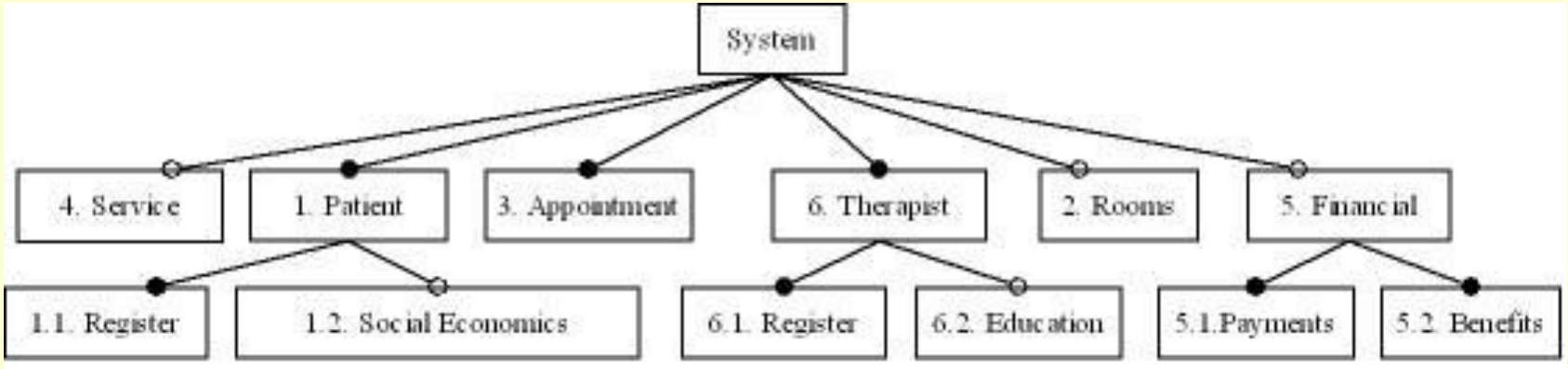
Característica (Feature): Definição

- É um conceito derivado de equipamentos físicos, surgida na área de telefonia. É usada em famílias de produtos de software. Trata-se de uma funcionalidade visível para o usuário.
- “A feature is an end-user visible characteristic of a system” (Kang et al 1990)
- Modelos de características foram introduzidos pela comunidade de projeto de software para criar abstrações a partir de um dado projeto arquitetural de alto nível. Expressa uma intenção, ou objetivo de um sistema (Pohl et al).

Característica - exemplos

- Ex. Uma característica de um aparelho de telefones é ter Viva Voz.
- Ex. em software:
  - o módulo de “cache” em um sistema de persistência de dados.
  - O sistema de biblioteca tem “reserva” de livros. (Característica opcional)
- Requisitos vs Características? A característica pode corresponder a um ou mais requisitos. Geralmente corresponde a um conjunto de requisitos relacionados entre si e que podem ser nomeados coletivamente.

O Diagrama de Características "Original"



LPS: Exemplos

- Elas são mais comuns na indústria de bens:
  - Máquinas fotográficas
  - Aparelhos telefônicos
  - Linha de aeronoves Airbus A-318,319,320,321
- Em software
  - Axix Communications AB : servidores de impressão para Impressoras IBM
  - Securitas Larm AB – Alarmes eletrônicos
  - Philips – sistemas de som
  - HP – servidores de impressão
  - Motores Diesel Cummins, etc.

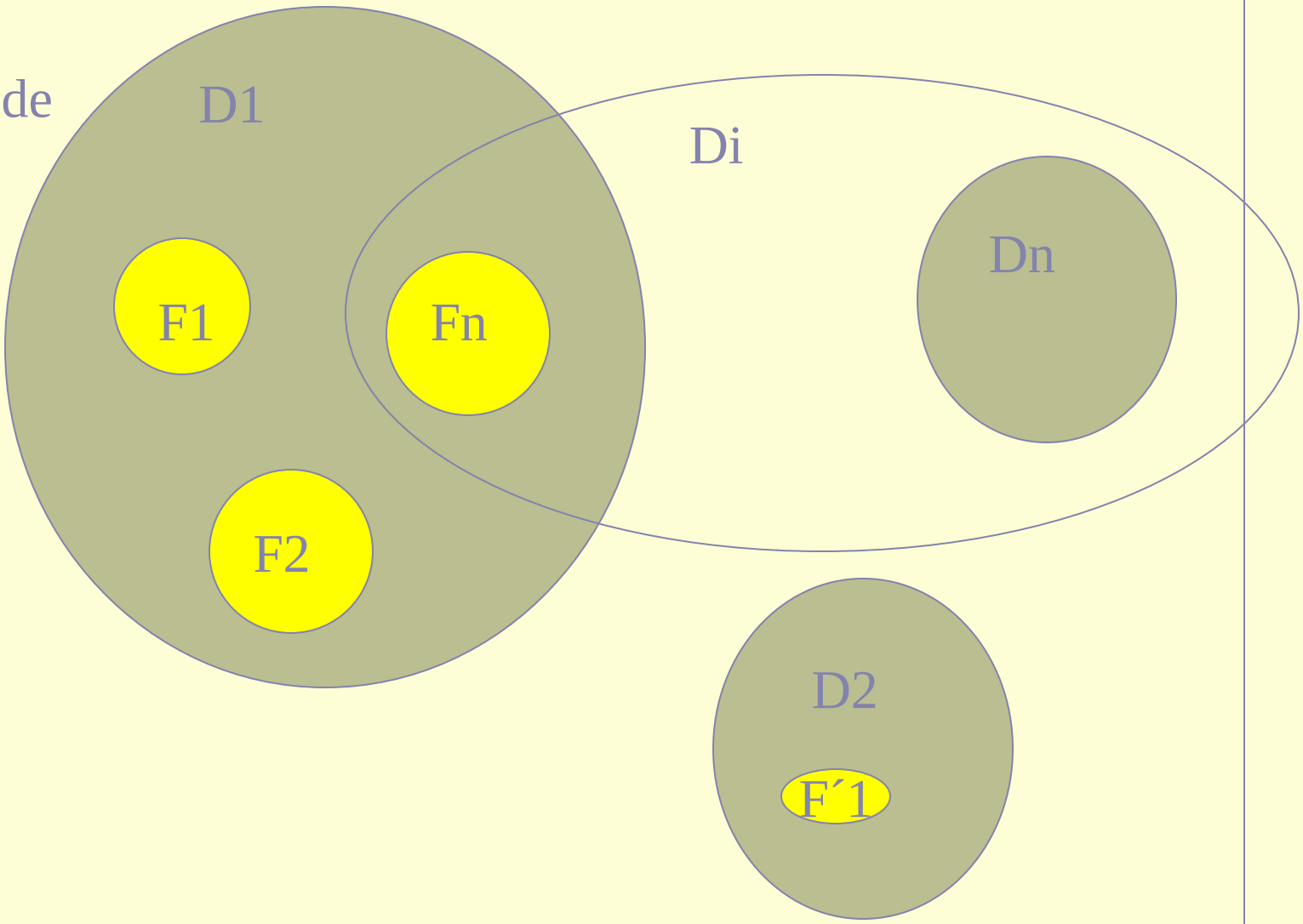
Definições Preliminares: Famílias

- Em engenharia: um conjunto de itens que têm partes comuns e variabilidades previsíveis.
  - Ex: Uma família de carros: chassi comum mas motor e transmissão diferentes.
- Famílias “fabricadas” (*engineered*) são comumente chamadas de domínios (W&L).
- Uma família de produtos projetada para se aproveitar dos aspectos comuns e das variabilidades previstas é uma linha de produtos (W&L).

Definições Preliminares: Domínio

- Em software: uma área do conhecimento ou atividade caracterizada por um conjunto de conceitos e terminologia compreendido pelos participantes dessa área (Booch et all)
  - Ex. Sistemas Financeiros (caracterizado pelo tipo de aplicação).
  - Ex. Sistemas de Informação (caracterizado pela arquitetura)

Universo: todos os sistemas de software



Variabilidades - Definição

- Significado informal: refere-se à habilidade ou tendência a mudar.
- Em software, estamos interessados em variabilidades que não ocorrem por acaso, mas são consideradas propositalmente.
- Exs: um telefone celular comunica-se com três padrões de redes, um chiclete pode ser doce ou amargo, um mesmo modelo de carro pode ter ar-condicionado ou não.

Variabilidades: outras definições

- Ponto de variação: um ou mais locais onde uma variação pode ocorrer (pode ser em modelos UML, ou em um programa de computador, por exemplo). É uma representação de uma variabilidade
- Variante: é a representação de uma instância de uma variabilidade. Identifica uma única opção de um ponto de variação.
- Ex. Variabilidade: Pagamento; Variantes: em dinheiro, em cheque, por cartão de crédito.

Modelagem de Variabilidades

- Com parâmetros: em tempo de compilação ou de configuração ou de execução, ou armazenados em tabelas.
- Usando ocultamento de informação: diversos componentes com diferentes implementações e mesma interface.
- Usando herança (frameworks)
- Usando padrões de projeto
- Usando aspectos (mais recente)

Motivação para a Engenharia de Linhas de Produtos de Software

- Redução dos custos de desenvolvimento,
- Quantas aplicações devem ser desenvolvidas para que compense desenvolver uma LPS?

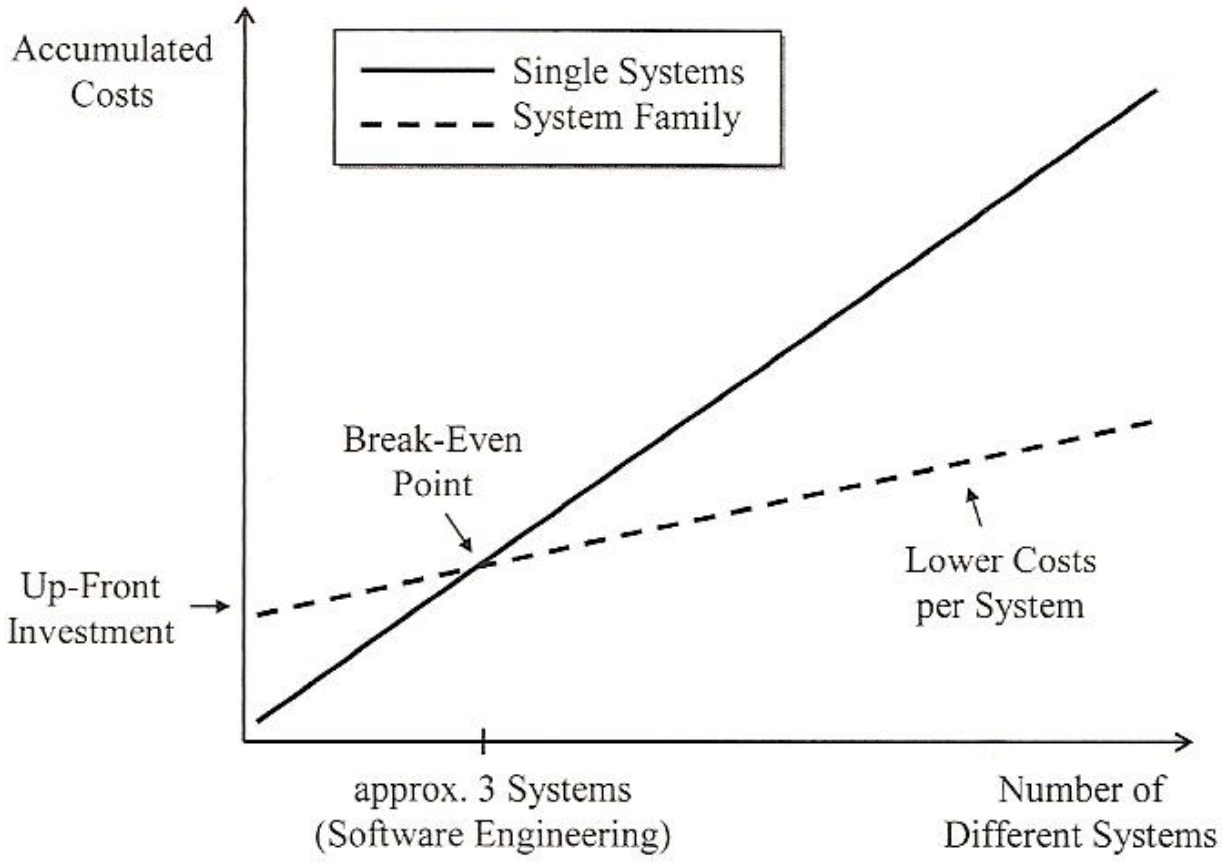


Fig. 1-2: Costs for developing n kinds of systems as single systems compared to product line engineering

Estratégias de Desenvolvimento de LPS

- Engenharia evolucionária avante (forward evolutionary engineering) ou pró-ativa
  - A organização faz um investimento inicial para desenvolver ativos reusáveis para a linha de produto e produtos são desenvolvidos usando esses ativos.
- Pode ser efetivo para domínios maduros e estáveis, nos quais os produtos podem ser previstos.
- Exige grande desenvolvimento

Estratégias de Desenvolvimento (cont.)

- Engenharia evolucionária reversa (forward evolutionary engineering) ou reativo.
  - Ativos são desenvolvidos quando necessários e reusados quando a oportunidade aparece.
- Pode funcionar bem em domínios nos quais os produtos não podem ser previstos antecipadamente.
- Não exige grande investimento inicial, mas os custos dos processo de reengenharia e reajustes de ativos reusáveis podem ser altos sem uma bem pensada arquitetura básica comum.

Estratégias de Desenvolvimento (cont.)

- Extrativo (Krueger)
  - É um meio-termo entre os dois anteriores : reusa um ou mais produtos de software existentes como a versão inicial da linha de produto.
- Pode ser efetivo para organizações que acumularam artefatos em um domínio mas querem mudar rapidamente da engenharia convencional para a engenharia baseada em LPS.

O Processo de Desenvolvimento de uma LPS

- O processo evolucionário de uma LPS elimina a distinção entre desenvolvimento e manutenção.
- O Sistema evolui por meio de uma série de iterações.
- Há dois sub-processos importantes: engenharia da LPS e engenharia da aplicação de software.

Engenharia Avante (Gomaa)

- A ênfase inicial é desenvolver o núcleo da LPS, que é constituído pelas funcionalidades comuns a todos os produtos da linha.
- Iniciar com o desenvolvimento dos casos de uso do núcleo.
- Continuar com a mesma abordagem para a modelagem estática (Diagramas de Classes)
- Continuar com a modelagem dinâmica....
- Depois, a LPS evolui com a adição dos casos de uso opcionais e alternativos.

Engenharia Avante

- Esta estratégia tem a vantagem de que a evolução faz parte do processo de desenvolvimento.
- É melhor quando uma nova linha de produtos está sendo desenvolvida e o núcleo pode ser determinado antes das funcionalidades variáveis.

Engenharia Reversa (Gomaa)

- Aplicável quando existem sistemas reais para serem analisados e modelados
- Modelos de análise de cada sistema são analisados e documentados (supõe-se que existem): Casos de Uso, Diagramas de Classe etc.
- O núcleo do sistema é constituído pelos casos de uso que aparecem em todos os sistemas e os demais se tornam opcionais ...

O Método original da SEI

- O método para desenvolvimento de LPS do SEI é baseado em características (Feature-oriented Domain Analysis).
- Um diagrama de características é uma composição de uma hierarquia de características, na qual alguns ramos são obrigatórios, e outros são variáveis: opcionais e alternativos.
- No método FoDA as características podem ser funcionais (hardware ou software), não-funcionais ou parâmetros.

Métodos para o Desenvolvimento de LPS

- UML Components (Cheesman and Daniels) , mais antigo, voltado para Sistemas de Informação.
- PLUS – Product Line UML-based Software Engineering (Gomaa) – baseado na UML, UP e componentes
- De Weiss e Lai: chamado de FAST, baseado no uso de geradores de aplicação. Ênfase nos processos.
- KOBRA
- POHL et all.- Software Product Line Engineering

Outros livros sobre LPS

- Bosch – ênfase na arquitetura
- Clements and Northrop - ênfase nos processo e na capacitação organizacionais (abrangente)

O Método de Gomaa

- Nome do método: PLUS (**P**roduct **L**ine **U**ML-Based **S**oftware Egineering)
- É iterativo e compatível com o Método Unificado da Rational e com o modelo de processo em espiral (Boehm).

O Processo ESPLEP

- Evolutionary Software Product Line Engineering Process (ESPLEP)
- É um processo de software que elimina distinção entre desenvolvimento e manuteção.
- É um processo da família do PLUS

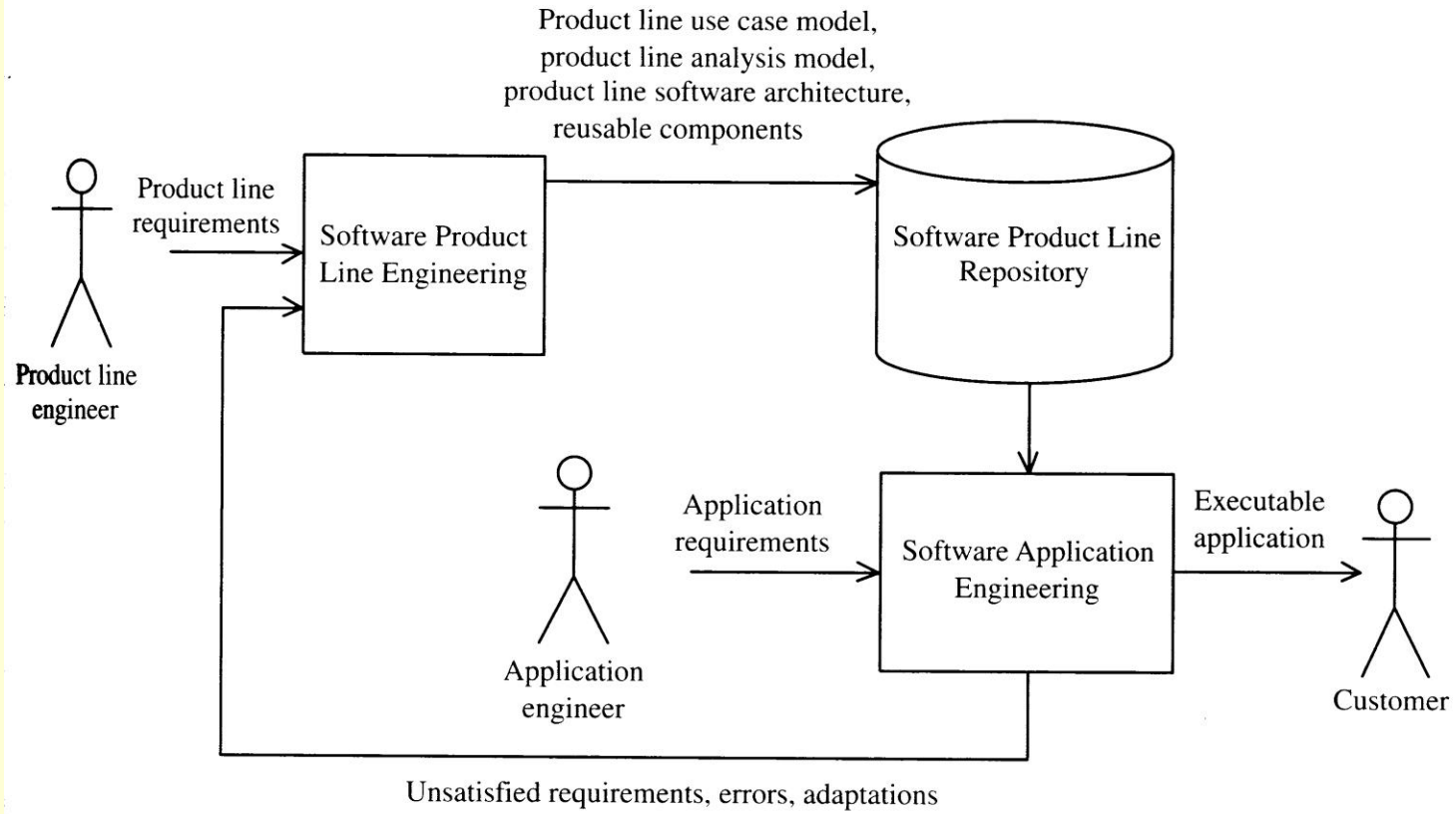
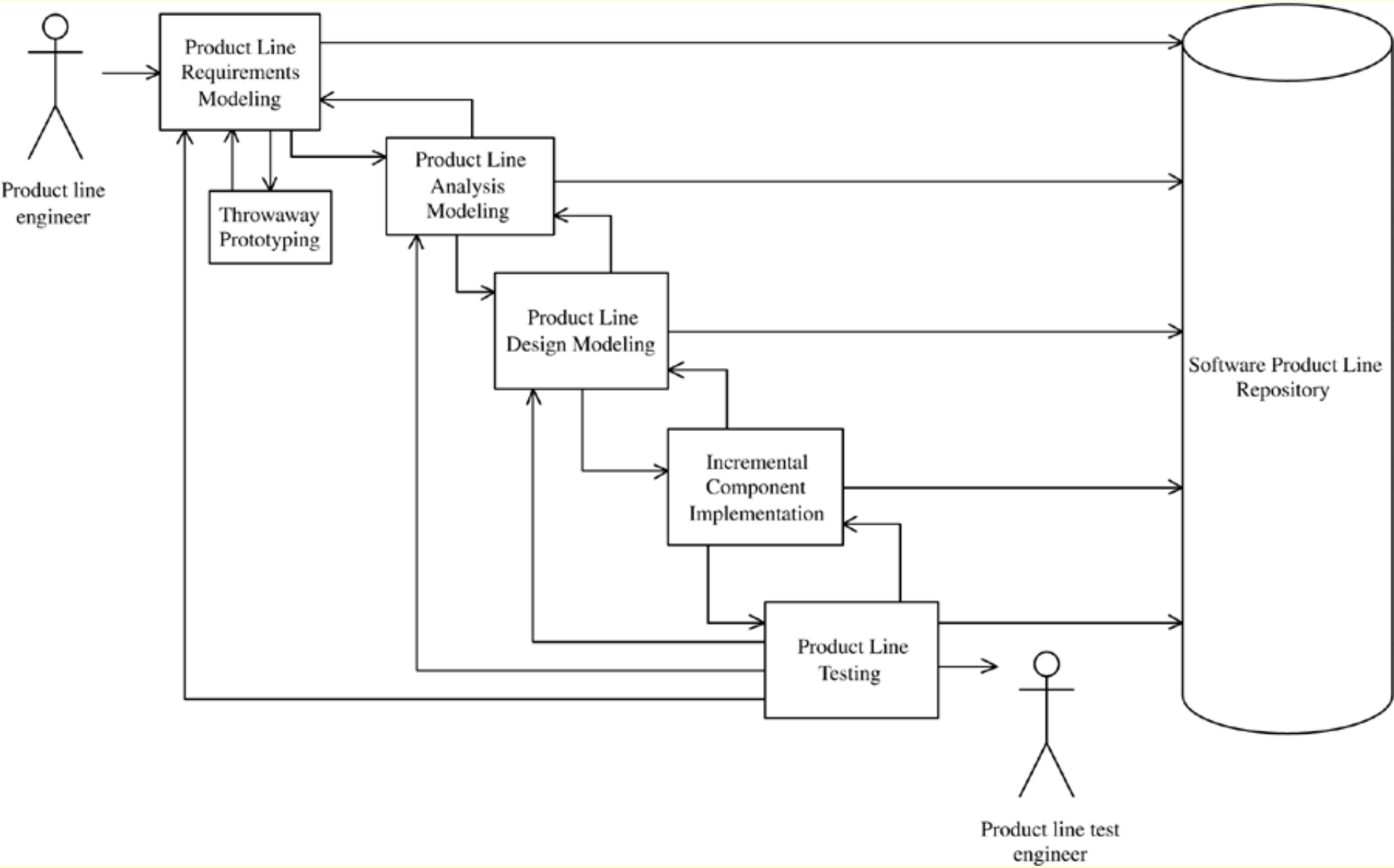


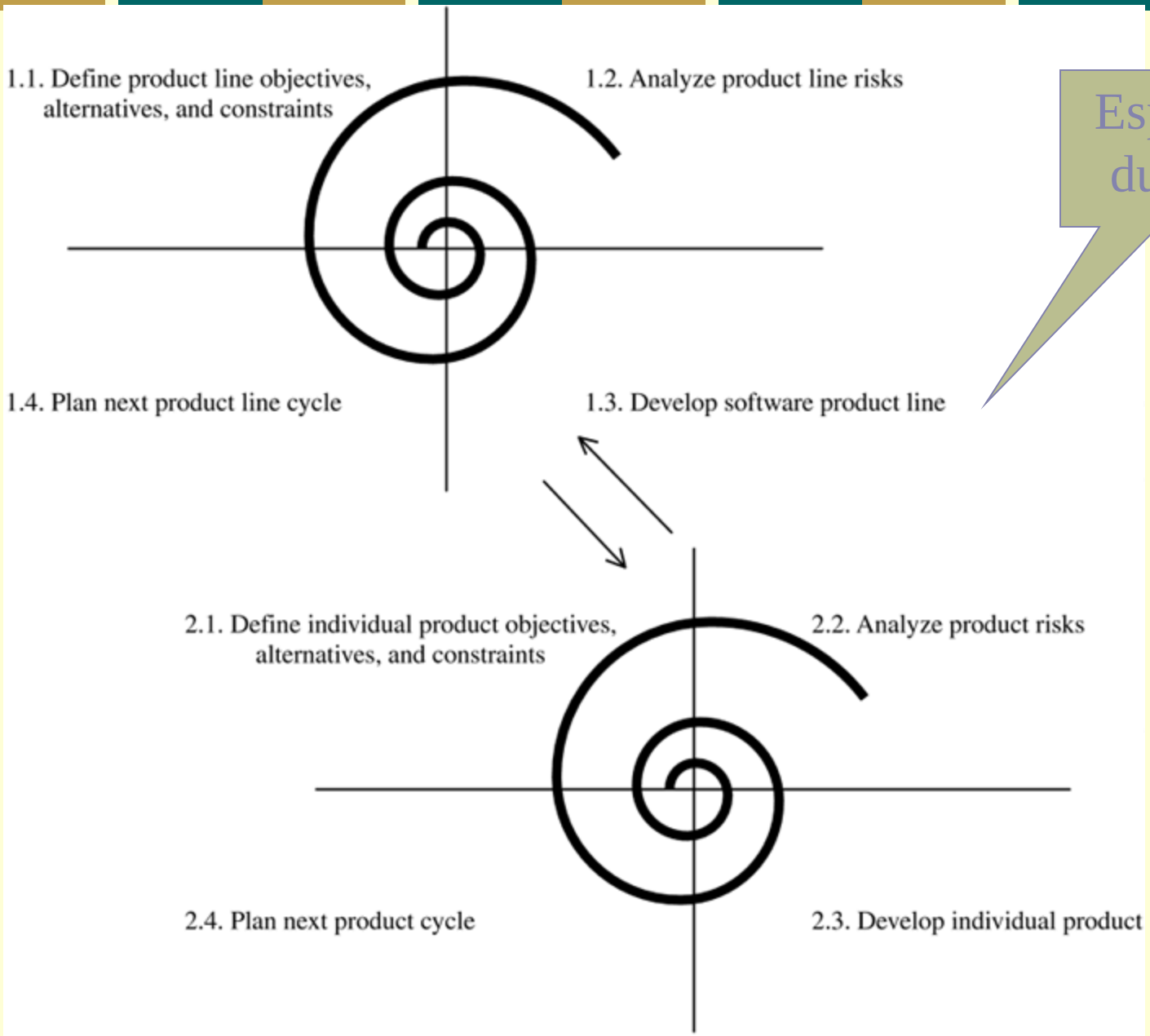
Figure 3.1 *Evolutionary Software Product Line Engineering Process*

Fases do ESPLEP


Product line engineer
Product Line Requirements Modeling
Throwaway Prototyping
Product Line Analysis Modeling
Product Line Design Modeling
Incremental Component Implementation
Product Line Testing
Software Product Line Repository
Product line test engineer

Integração do PLUS com o Modelo Espiral

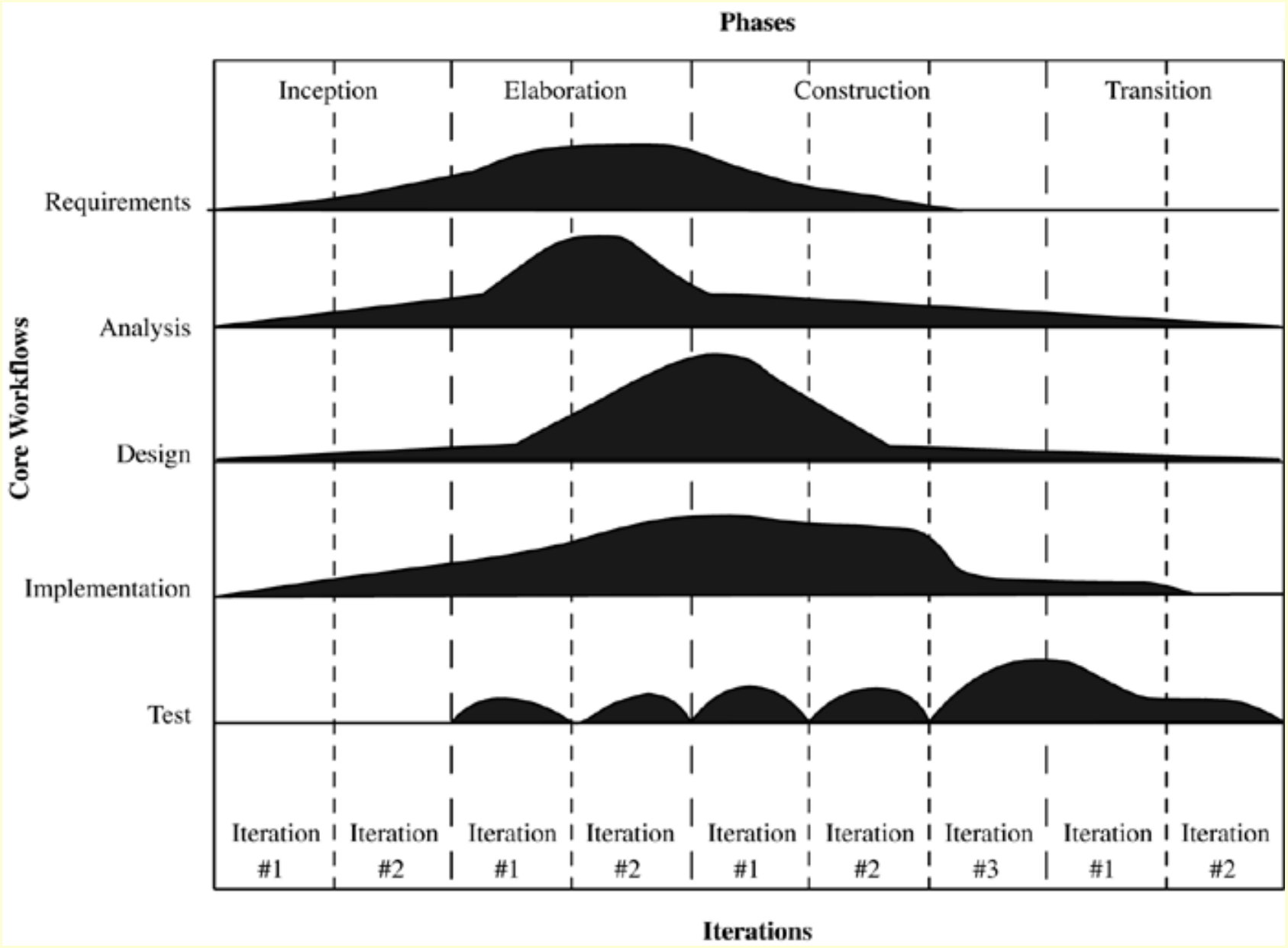
- O modelo em espiral pode ser aplicado para o desenvolvimento da LPS e para o desenvolvimento de produtos individuais.
- Durante o planejamento do projeto para um certo ciclo do modelo em espiral, o gerente do projeto decide que atividades técnicas específicas devem ser executadas no 3o. Quadrante.
- A análise de risco define quantas interações das atividades técnicas são necessárias.



Espirais duplas

Integração do PLUS com o Processo Unificado

- Cada fase do ciclo de vida do PLUS corresponde a um workflow do PU.
- As fases do PLUS têm o mesmo nome dos workflows do PU.


Phases
Inception
Elaboration
Construction
Transition
Requirements
Analysis
Design
Implementation
Test
Core Workflows
Iteration #1
Iteration #2
Iteration #1
Iteration #2
Iteration #1
Iteration #2
Iteration #3
Iteration #1
Iteration #2
Iterations

As Fases do PU para a Engenharia da LP

- Concepção
- Elaboração: Iteração 1: O Núcleo Primeiro
- Elaboração: Iteração 2: Evolução da LP
- Construção: Iteração 1: construção dos componentes do núcleo
- Transição: Iteração 1: Teste do núcleo do sistema
- Outras iterações da Engenharia da LP

As Fases do PLUS

- Requisitos
  - Definição do Escopo
  - Modelagem dos Casos de Uso (similaridades e variabilidades)
  - Modelo de Características
- Modelo de Análise
  - Modelagem estática
  - Estruturação dos objetos
  - Modelagem Dinâmica
  - Modelagem por Máquina de Estados Finitos
  - Análise da dependência Características/Classes
- Modelo de Projeto
  - Projeto baseado em padrões de arquitetura
  - Projeto arquitetural da LPS
    - Arquitetura baseada em componentes

As Fases do PLUS

- Requisitos
  - Definição do Escopo
  - Modelagem dos Casos de Uso (similaridades e variabilidades)
  - Modelo de Características
- Modelo de Análise
  - Modelagem estática
  - Estruturação dos objetos
  - Modelagem Dinâmica
  - Modelagem por Máquina de Estados Finitos
  - Análise da dependência Características/Classes
- Modelo de Projeto
  - Projeto baseado em padrões de arquitetura
  - Projeto arquitetural da LPS
    - Arquitetura baseada em componentes

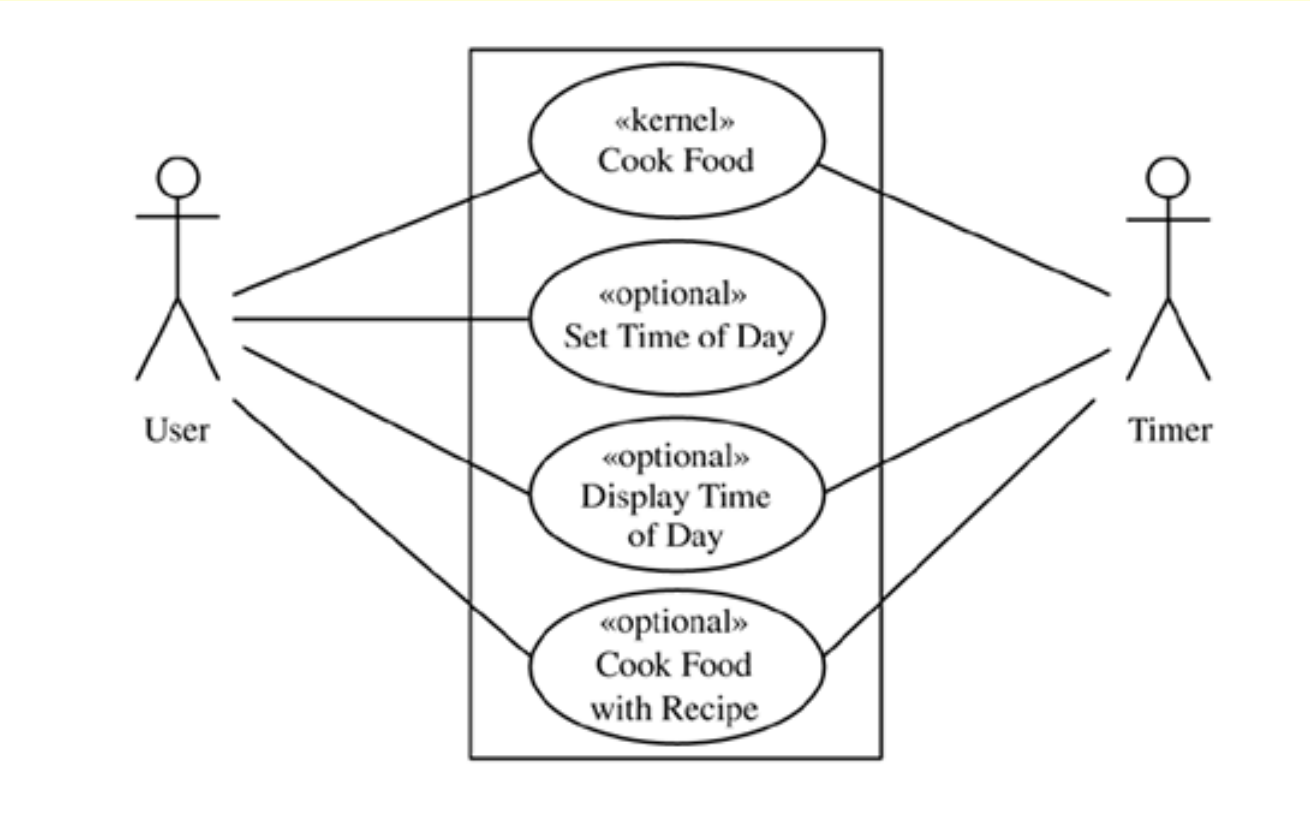
Abrangência da LPS

- “Scoping”
- É a atividade de avaliação de qual funcionalidade – em particular, que grau de similaridade e variabilidade – deve ser oferecida pela LPS.

Modelo de Casos de Uso para LPS

- Casos de uso podem ser requeridos por todos os produtos da linha ou só por alguns deles:
  - Casos de uso do núcleo
  - Casos de uso opcional
  - Casos de uso alternativos, mutuamente exclusivos.
- A distinção entre os tipos de casos de uso é feita por estereótipos:
  - <<kernel>>
  - <<optional>>
  - <<alternative>>

Exemplo: forno de microondas



Descrição dos casos de uso

- Nome
- Categoria de reuso: Kernel, optional ou alternative
- Resumo
- Atores
- Dependências
- Pré-Condições
- Descrição
- Alternativas
- Pontos de Variação
- Pós-condições
- Questões relevantes

Use case name: Cook Food.
Reuse category: Kernel.
Summary: User puts food in oven, and microwave oven cooks food.
Actors: User (primary), Timer (secondary).
Precondition: Microwave oven is idle.
Description:

1. User opens the door, puts food in the oven, and closes the door.
2. User presses the Cooking Time button.
3. System prompts for cooking time.
4. User enters cooking time on the numeric keypad and presses Start.
5. System starts cooking the food.
6. System continually displays the cooking time remaining.
7. Timer elapses and notifies the system.
8. System stops cooking the food and displays the end message.
9. User opens the door, removes the food from the oven, and closes the door.
10. System clears the display.

Alternatives:
Line 1: User presses Start when the door is open. System does not start cooking.
Line 4: User presses Start when the door is closed and the oven is empty. System does not start cooking.
Line 4: User presses Start when the door is closed and the cooking time is equal to zero. System does not start cooking.
Line 6: User opens door during cooking. System stops cooking. User removes food and presses Cancel, or user closes door
and presses Start to resume cooking.
Line 6: User presses Cancel. System stops cooking. User may press Start to resume cooking. Alternatively, user may press
Cancel again; system then cancels timer and clears display.
Postcondition: Microwave oven has cooked the food.

Modelagem de pequenas variações

- Usam-se extensões e inclusões quando o caso de uso contém um bloco de funcionalidade que pode ser descrito como uma seqüência de interações entre um ator e um sistema.
- Em algumas situações, uma pequena variação afeta apenas uma ou duas linhas da descrição do caso de uso.
- Documentar a pequena variação dentro do próprio caso de uso.

Como documentar

- Nome
- Tipo da funcionalidade a ser inserida: optional, mandadory alternative, optional alternative
- Linha do caso de uso onde a variabilidade pode ser introduzida.

Modelagem de Variabilidades com pontos de extensão

- Podem ser usados para:
  - Variabilidade alternativa
  - Variabilidade opcional.
  - Evolução futura da linha de produto.

Exemplo de Func. Opcional

- Nome: Light
- Type of functionality: Optional
- Line number(s): 1,5,8,9
- Description of functionality: If Light option is selected, light is switched on for duration of cooking and when the door is open. Light is switched off when the door is closed and when cooking stops.

Exemplo de Func. Opcional

Name: Turntable.

Type of functionality: Optional.

Line number(s): 5, 8.

Description of functionality: If Turntable option is selected, turntable rotates for duration of cooking.

Exemplo de Func. Alternativa

Name: Display Language.

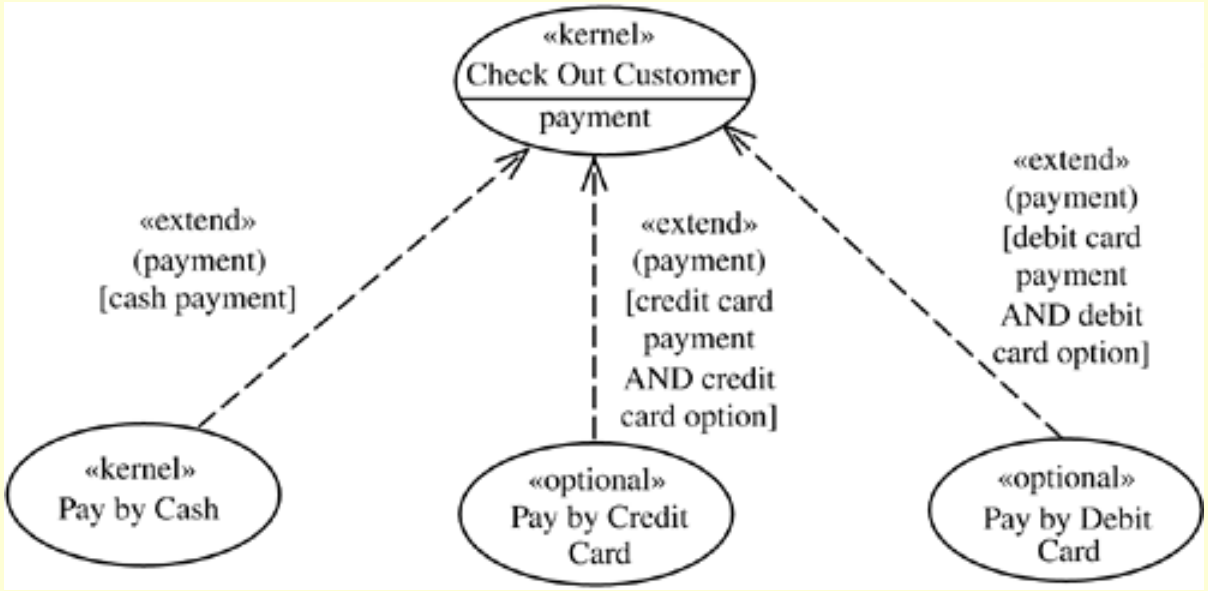
Type of functionality: Mandatory alternative.

Line number(s): 3, 8.

Description of functionality: There is a choice of language for displaying messages. The default is English. Alternative mutually exclusive languages are French, Spanish, German, and Italian.

Pontos de Extensão em LPS

- São usados para especificar a localização precisa no caso de uso base em que as extensões podem ser adicionadas.
- Cada ponto de extensão tem um nome.
- O Caso de Uso de extensão pode ter um ou mais segmentos de extensão.
- A extensão pode ser condicional.
- Descrição: marcadores (placeholders) são inseridos no caso de uso base.


«kernel»
Check Out Customer
payment
«extend»
(payment)
[cash payment]
«extend»
(payment)
[credit card
payment
AND credit
card option]
«extend»
(payment)
[debit card
payment
AND debit
card option]
«kernel»
Pay by Cash
«optional»
Pay by Credit
Card
«optional»
Pay by Debit
Card

Use case name: Check Out Customer.
Reuse category: Kernel.
Summary: System checks out customer.
Actor: Customer.
Precondition: Checkout station is idle, displaying a "Welcome" message.
Description:

1. Customer scans selected item.
2. System displays the item name, price, and cumulative total.
3. Customer repeats steps 1 and 2 for each item being purchased.
4. Customer selects payment.
5. System prompts for payment by cash, credit card, or debit card.
6. <payment>
7. System displays thank-you screen.

Use case name: Pay by Cash.
Reuse category: Kernel.
Summary: Customer pays by cash for items purchased.
Actor: Customer.
Dependency: Extends Check Out Customer.
Precondition: Customer has scanned items but not yet paid for them.
Description:

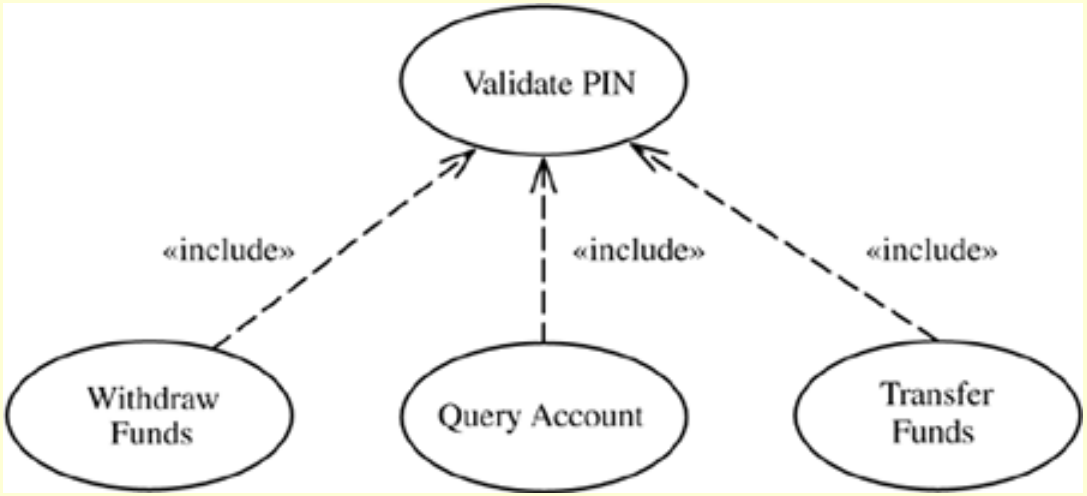
1. Customer selects payment by cash.
2. System prompts customer to deposit cash in bills and/or coins.
3. Cashier enters cash amount.
4. System computes change.
5. System displays total amount due, cash payment, and change.
6. System prints total amount due, cash payment, and change on
7. receipt.

Use case name: Pay by Credit Card.
Reuse category: Optional.
Summary: Customer pays by credit card for items purchased.
Actor: Customer.
Dependency: Extends Check Out Customer.
Precondition: Customer has scanned items but not yet paid for them.
Description:

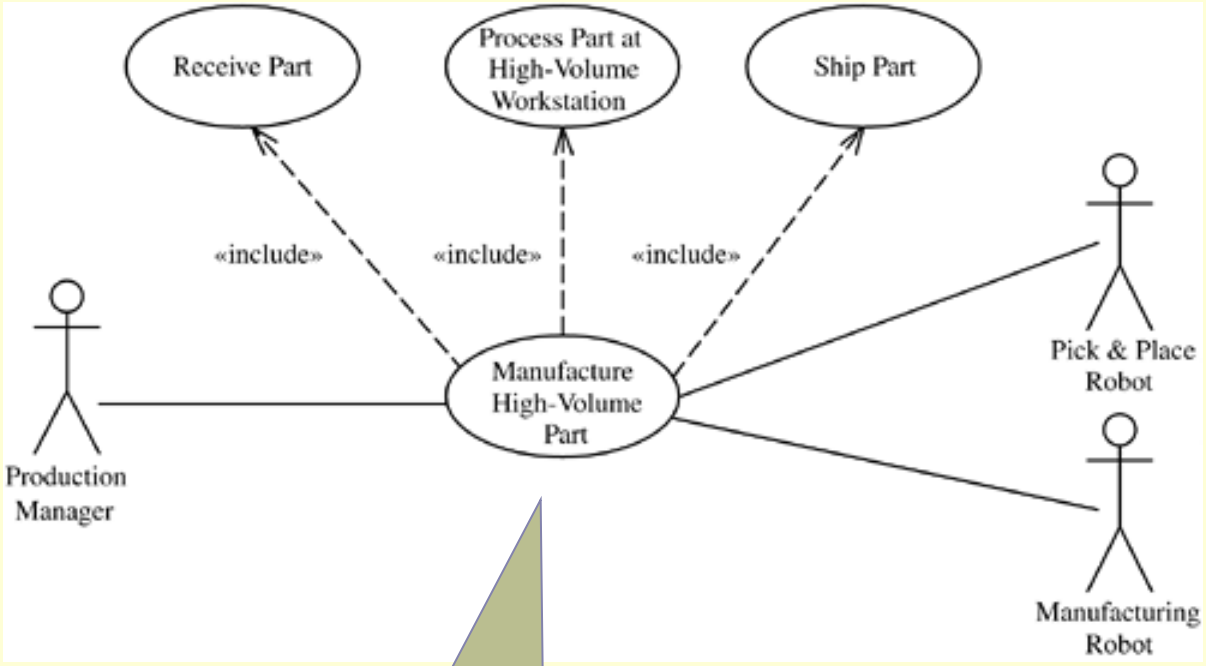
1. Customer selects payment by credit card.
2. System prompts customer to swipe card.
3. Customer swipes card.
4. System reads card ID and expiration date.
5. System sends transaction to authorization center containing card ID, expiration date, and payment amount.
6. If transaction is approved, authorization center returns positive confirmation.
7. System displays payment amount and confirmation.
8. System prints payment amount and confirmation on receipt.

Modelagem de Variabilidade com a relação Includes

- São usadas na seguinte situação: depois que um caso de uso inicial para uma aplicação foi desenvolvido, seqüências comuns de interação entre o ator e o sistema são determinadas e abrangem vários casos de uso.
- Pode ser usada para casos de uso opcionais.

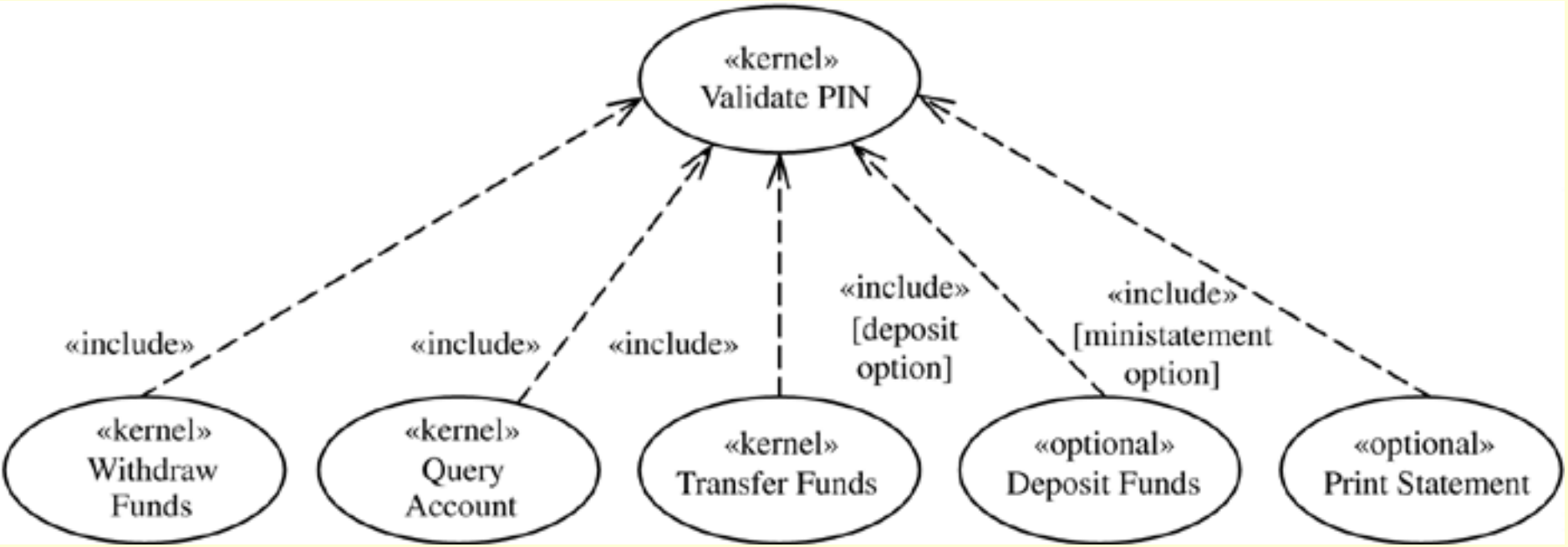


Caso de Uso Abstrato



Multiplos casos de uso abstratos

Várias combinações



As Fases do PLUS

- Requisitos
  - Definição do Escopo
  - Modelagem dos Casos de Uso (similaridades e variabilidades)
  - Modelo de Características
- Modelo de Análise
  - Modelagem estática
  - Estruturação dos objetos
  - Modelagem Dinâmica
  - Modelagem por Máquina de Estados Finitos
  - Análise da dependência Características/Classes
- Modelo de Projeto
  - Projeto baseado em padrões de arquitetura
  - Projeto arquitetural da LPS
    - Arquitetura baseada em componentes

Definição

- Uma *feature* (ou features) é um requisito ou uma característica que é oferecida por um ou mais membros da linha de produtos.
- São usadas para diferenciar um membro do outro.
- Originou-se no Software Engineering Institute, em 1990: Feature-Oriented Domain Analysis.

Definição (cont.)

- Um requisito é usado para significar uma necessidade da linha de produto de software, e portanto, ao menos um membro da LPS deve ser capaz de satisfazê-lo.
- Uma vez que a LPS capturou esse requisito, ele é referido como uma característica (feature) fornecida pela LPS.

Análise de Similaridades e Variabilidades

- As features são classificadas como comum, opcional ou alternativa.
- As opções opcional e alternativa determinam o grau de variabilidade de uma família.
- Usam-se os estereótipos:
  - <<common feature>>
  - <<optional feature>>
  - <<alternative feature>>

Features parametrizadas

- É uma feature que define um parâmetro da linha de produto, cujo valor deve ser definido em tempo de configuração do sistema.
- Deve-se especificar o tipo do parâmetro, intervalo de valores permitidos ou, opcionalmente, um valor default.
- Ex: <<parameterized feature>> ATM PasswordLenght {type=integer, permitted-value = 4..8, default value = 4}

Pré-requisito de uma Feature

- Uma feature pode depender de outra feature, que é chamada de pré-requisito.
- Não é necessário especificar features comuns como pré-requisitos.
- {prerequisite=Prerequisite Feature Name, ...}
- Ex: <<optional feature>> TOD Clock {prerequesite=Multi-line Dysplay}

Features mutuamente inclusivas

- Se duas features são sempre requeridas juntas, então são consideradas mutuamente inclusivas.
- Obs. este caso poderia ser resolvido por pré-requisitos, mas em certas situações é melhor usar features mutuamente inclusivas.
- {mutually includes=Mutually Inclusive Feature Name}
- Ex. <<optional feature>> Recipe{mutually includes=Analog Weight}

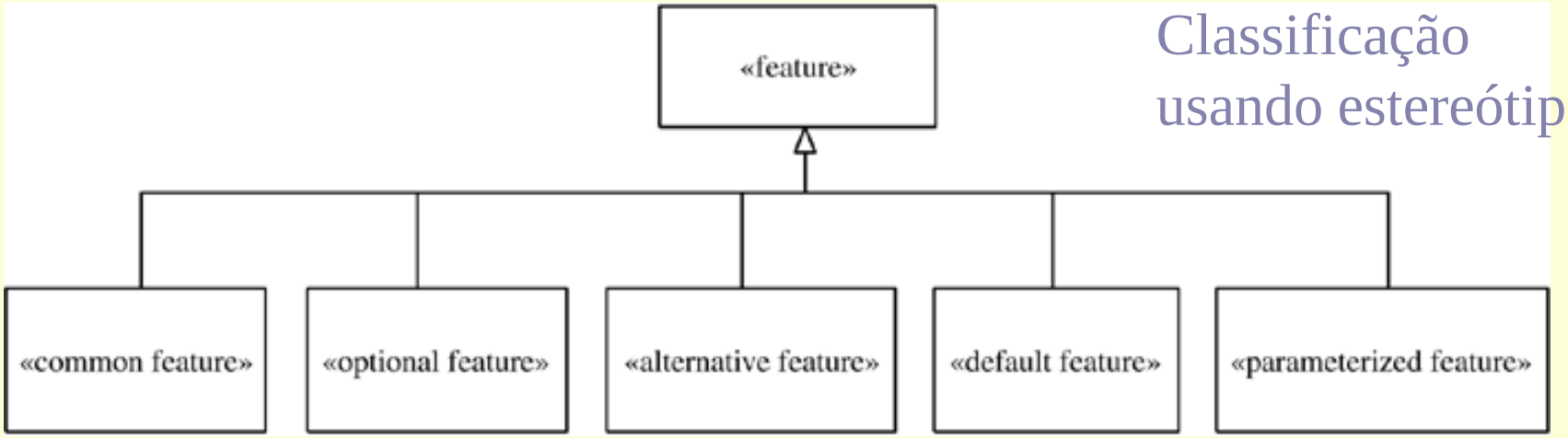
Modelagem de Features com UML

- Features como pacotes de casos de uso (já visto anteriormente)
- Features como metaclasses
- Representação como tabela
- Grupos de Features
- Modelagem avançada

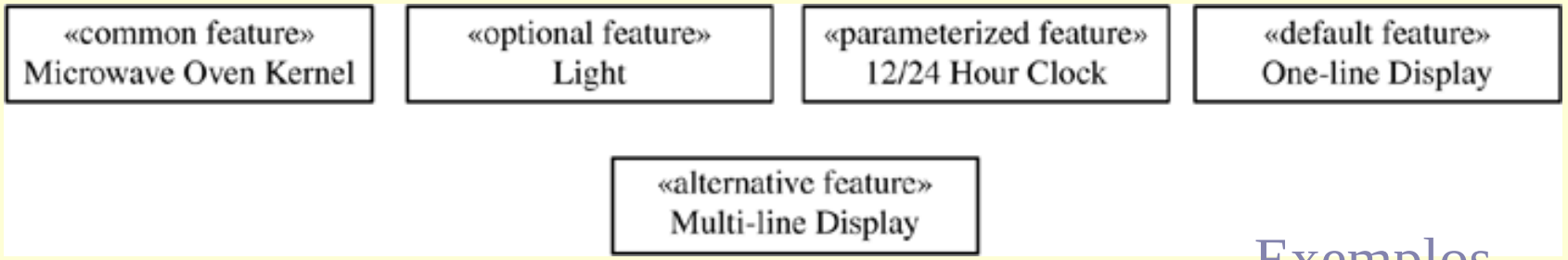
Modelagem como metaclasses

- A classe é usada para modelar um elemento de modelagem.
- Features e Grupos de Features são representados por metaclasses
- Dependências entre features são representadas como relacionamentos: requires ou mutually includes ou mutually esclusive

Features como metaclasses



Classificação usando estereótipos

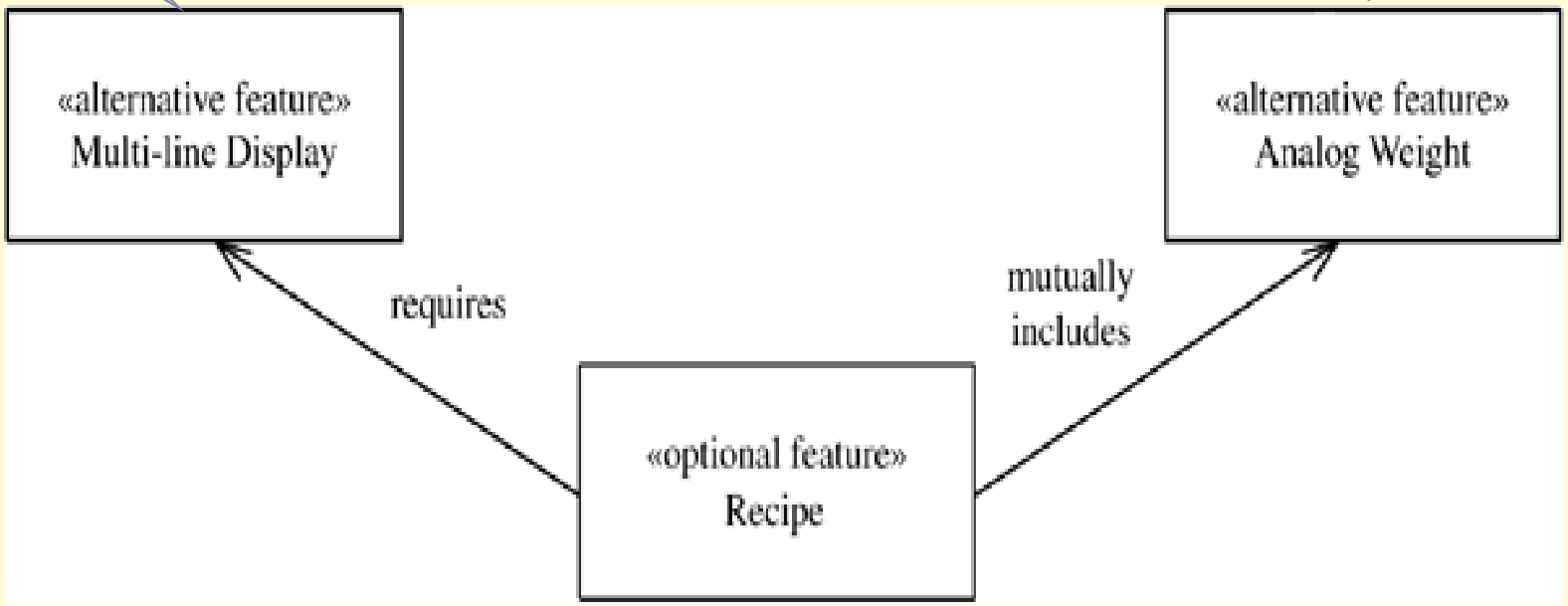


Exemplos

Usando dependências

Explicit feature

implicit feature



Como tabela

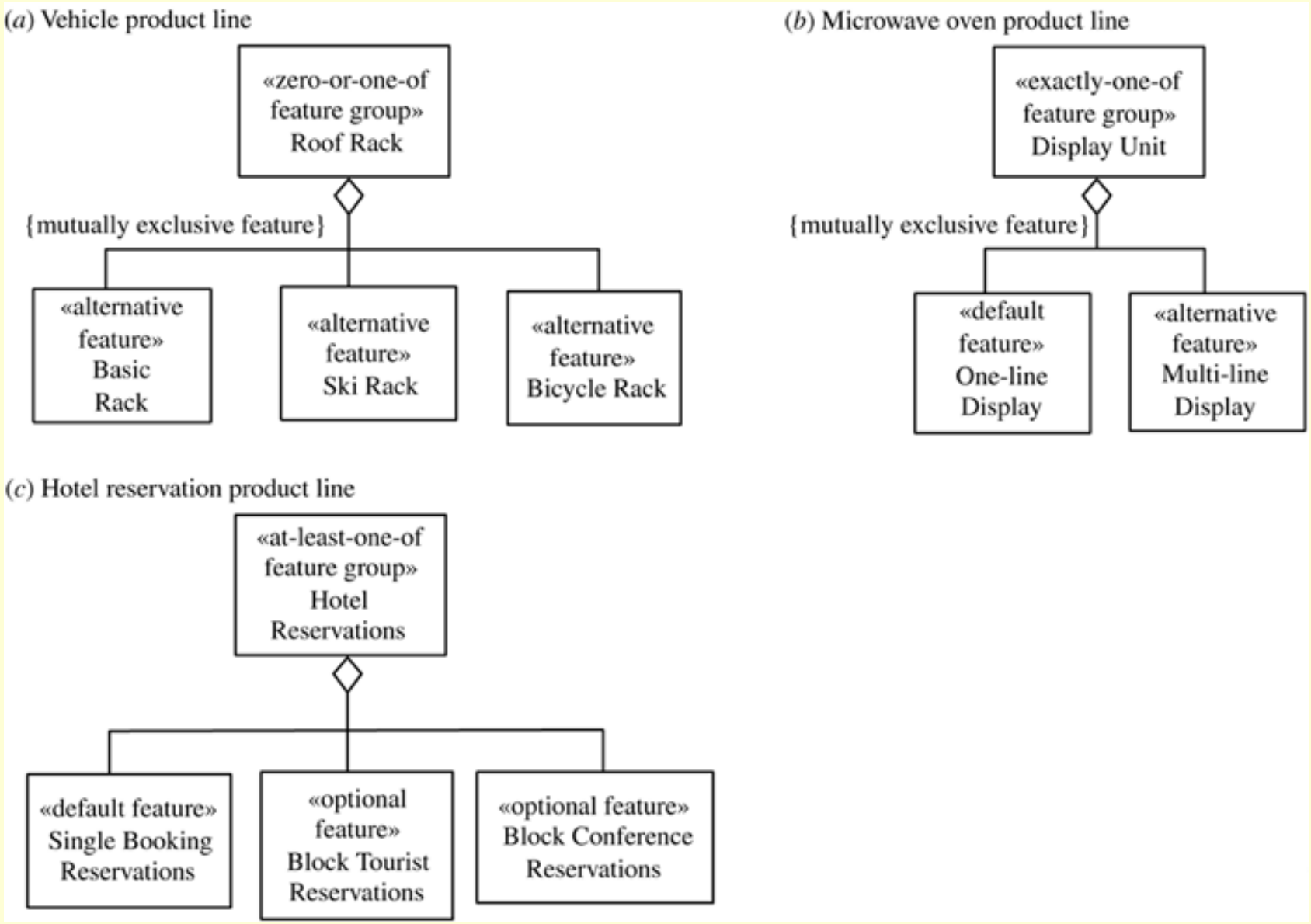
Feature Name	Feature Category	Use Case Name	Use Case Category/ Variation Point (vp)	Variation Point Name
Microwave Oven Kernel	Common	Cook Food	kernel	
Light	Optional	Cook Food	vp	Light
Turntable	Optional	Cook Food	vp	Turntable
Beeper	Optional	Cook Food	vp	Beeper
Minute Plus	Optional	Cook Food	vp	Minute Plus
One-line Display	Default	Cook Food	vp	Display Unit
Multi-line Display	Alternative	Cook Food	vp	Display Unit
English	Default	Cook Food	vp	Display Language
French	Alternative	Cook Food	vp	Display Language
Spanish	Alternative	Cook Food	vp	Display Language
German	Alternative	Cook Food	vp	Display Language
Italian	Alternative	Cook Food	vp	Display Language

Grupos de Features

- Features relacionadas podem ser agrupadas em grupos de features.
- Isso restringe a forma como as features são usadas por um membro da linha de produtos.
- Os grupos de features podem ser descritos como metaclasses ...
- ....ou por notação textual consistindo de estereótipos e etiquetas.

Grupos de Features (cont.)

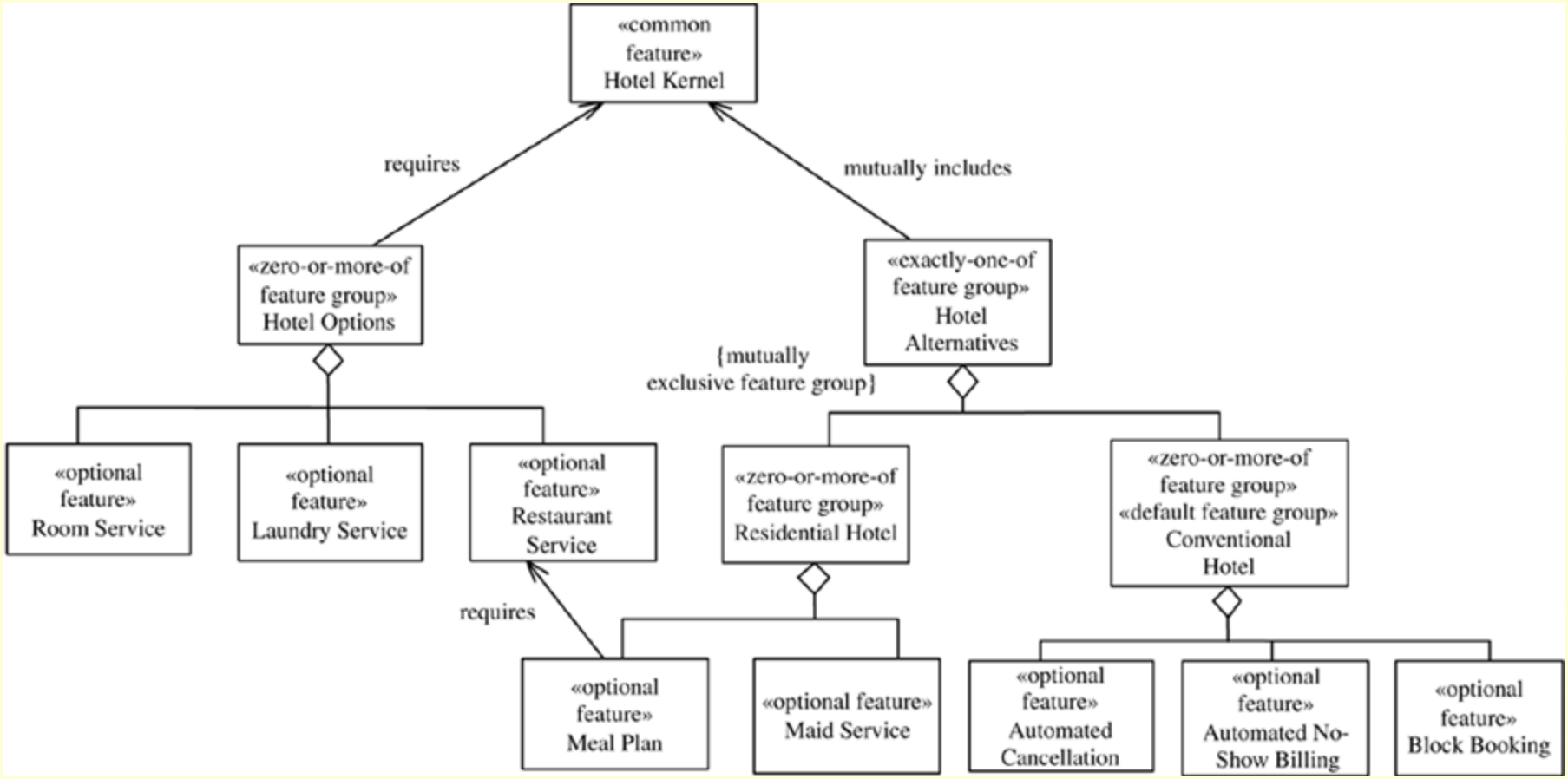
- Um grupo zero-or-one-feature significa que é opcional selecionar uma das alternativas.
- Um grupo exactly-one-of indica que uma feature do grupo deve sempre ser selecionada.
- Um grupo at-least-one indica que um ou mais features do grupo pode(m) ser selecionada(s).
- Um grupo zero-or-more pode ser útil para agrupar features opcionais quando há dependência.


(a) Vehicle product line
«zero-or-one-of feature group» Roof Rack
{mutually exclusive feature}
«alternative feature» Basic Rack
«alternative feature» Ski Rack
«alternative feature» Bicycle Rack
(b) Microwave oven product line
«exactly-one-of feature group» Display Unit
{mutually exclusive feature}
«default feature» One-line Display
«alternative feature» Multi-line Display
(c) Hotel reservation product line
«at-least-one-of feature group» Hotel Reservations
«default feature» Single Booking Reservations
«optional feature» Block Tourist Reservations
«optional feature» Block Conference Reservations

Modelagem Avançada

- Em algumas linhas de produtos, é desejável ter grupos de features que consistem de outros grupos de features.
- Por exemplo, dois grupos de features podem ser mutuamente exclusivos.
- Ex. Hotéis residenciais com reservas por um mês são diferentes de hotéis convencionais
- Na notação, substitui-se o termo feature por feature group.

Modelagem avançada



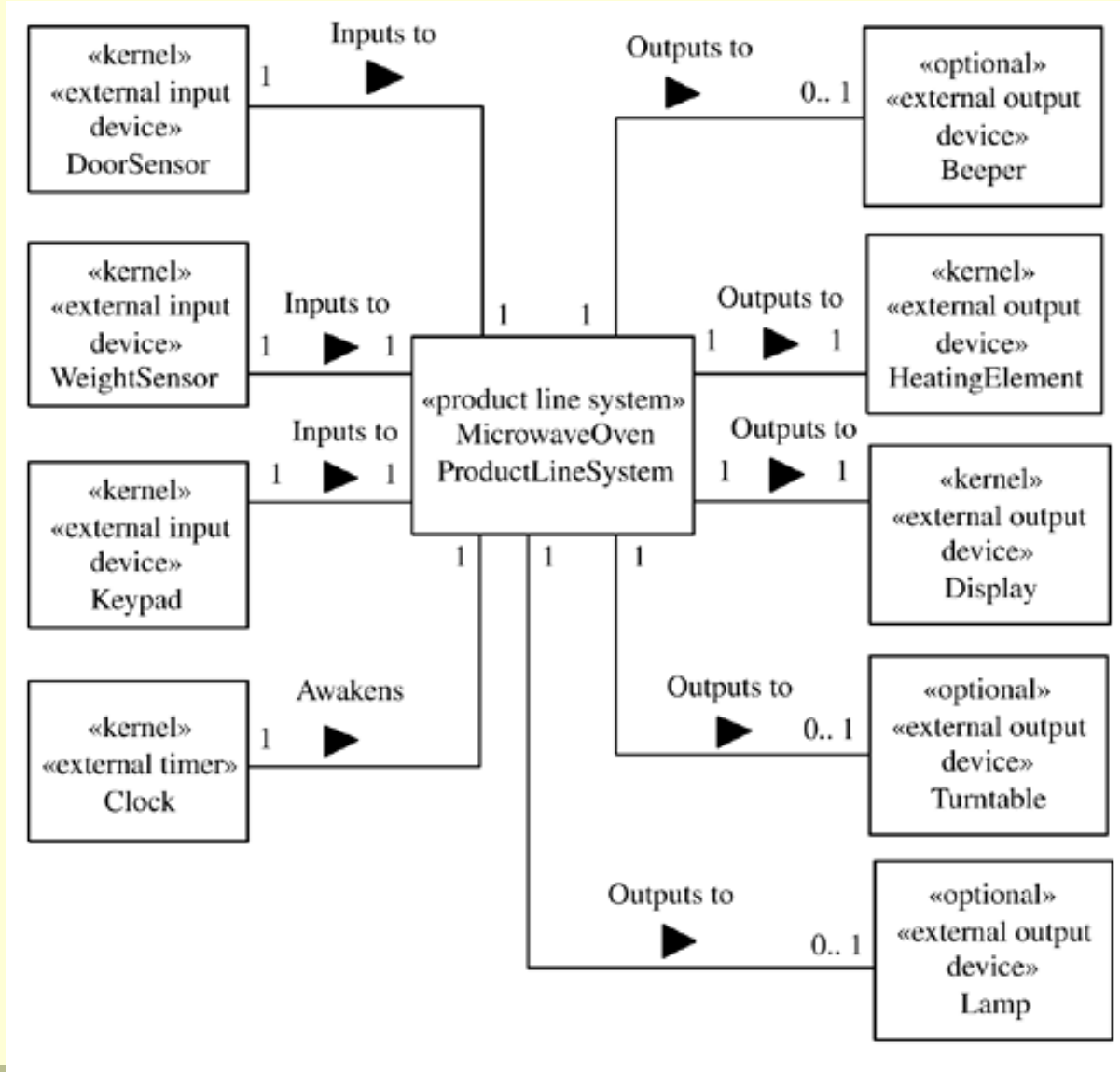
As Fases do PLUS

- Requisitos
  - Definição do Escopo
  - Modelagem dos Casos de Uso (similaridades e variabilidades)
  - Modelo de Características
- Modelo de Análise
  - Modelagem estática
  - Estruturação dos objetos
  - Modelagem Dinâmica
  - Modelagem por Máquina de Estados Finitos
  - Análise da dependência Características/Classes
- Modelo de Projeto
  - Projeto baseado em padrões de arquitetura
  - Projeto arquitetural da LPS
    - Arquitetura baseada em componentes

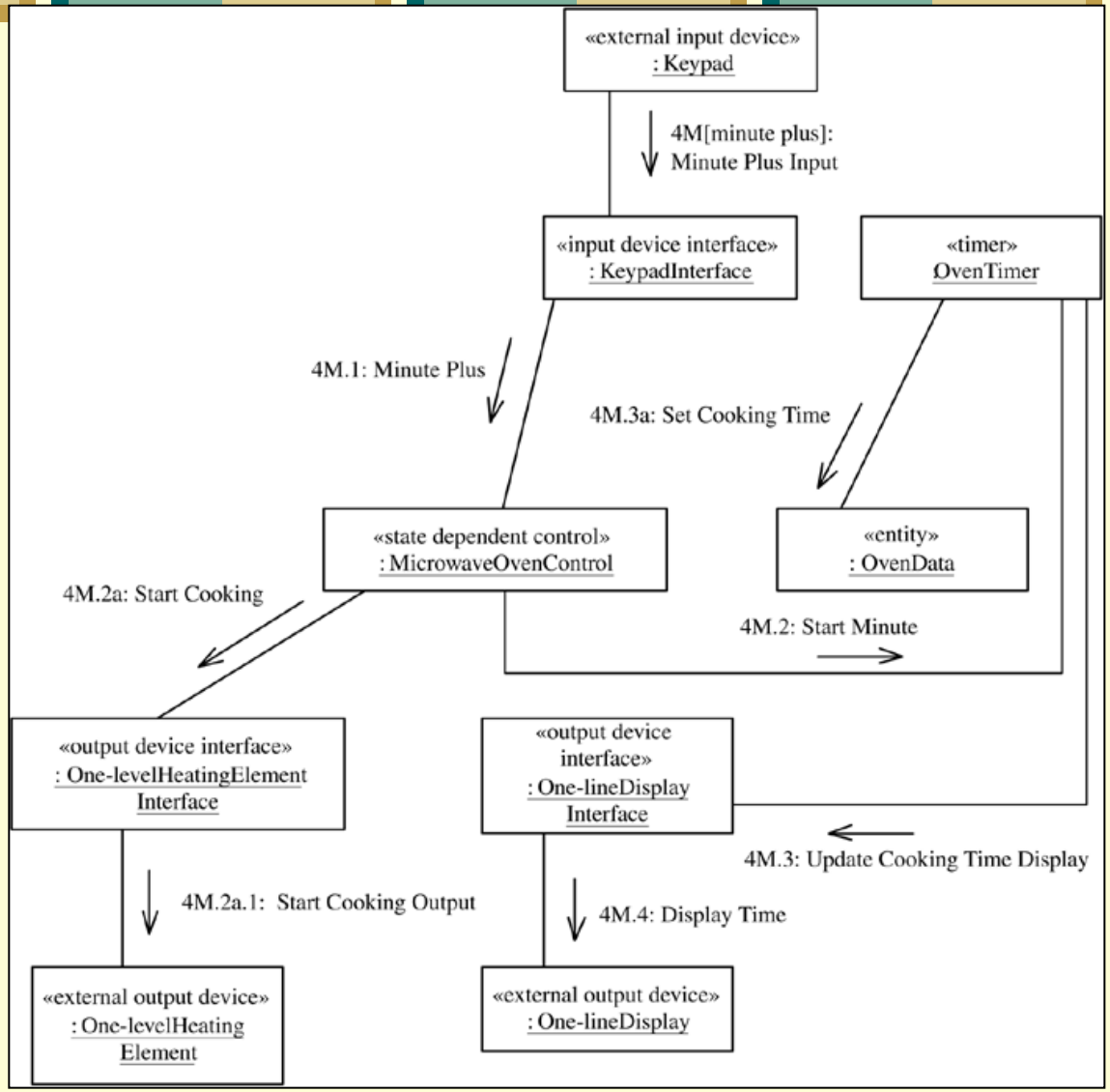
Modelo Estático

- Classe do Núcleo
  - Requerida por **todos** os membros da linha de produto
- Classe Opcional
  - Fornecida apenas por **alguns** membros das família
- Classe Variante
  - Versão de uma classe, que tem **variações** para diferentes membros
- Classe Default
  - Designada entre um grupo de classes variantes

Modelo Estático Conceitual

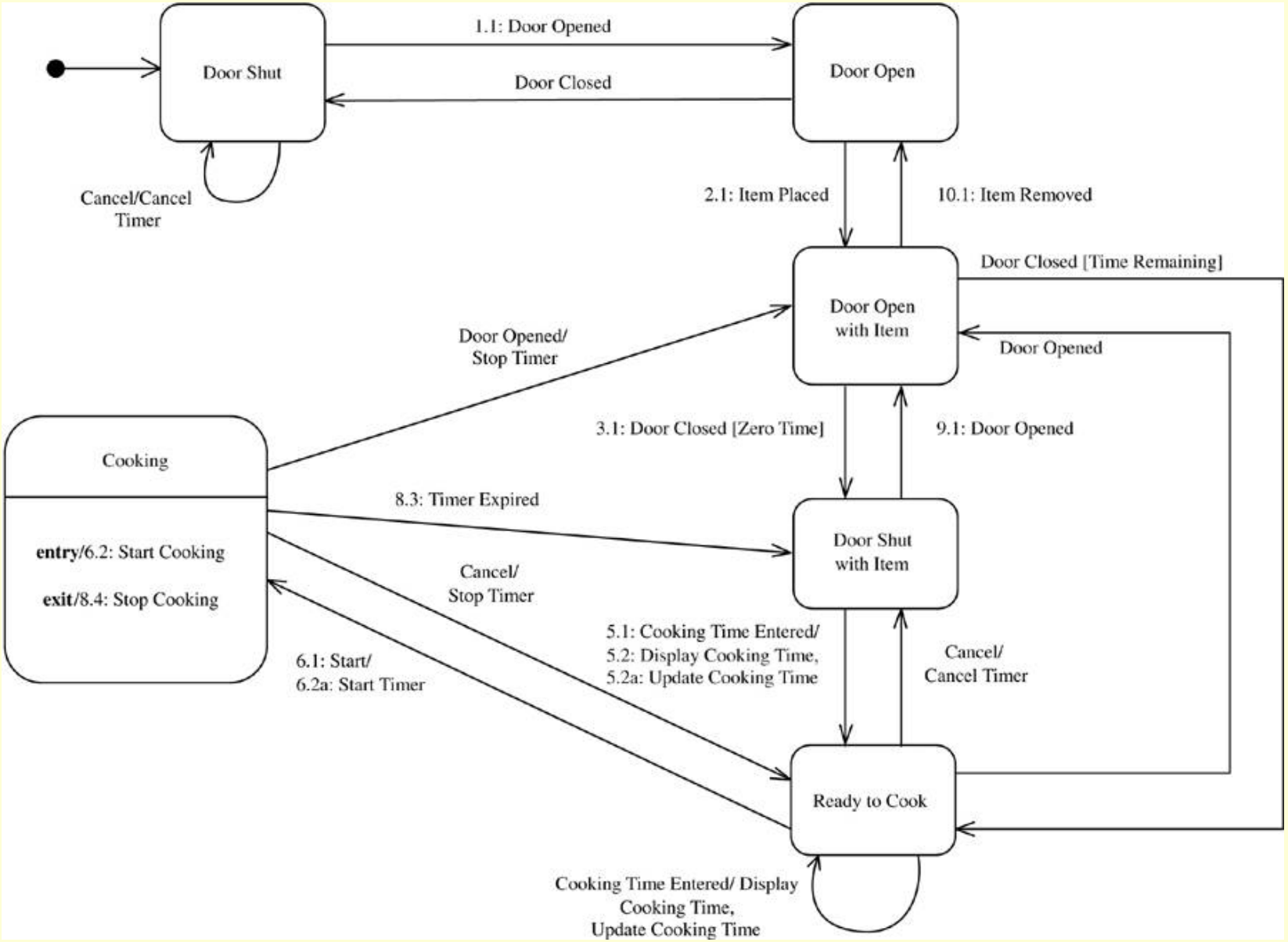


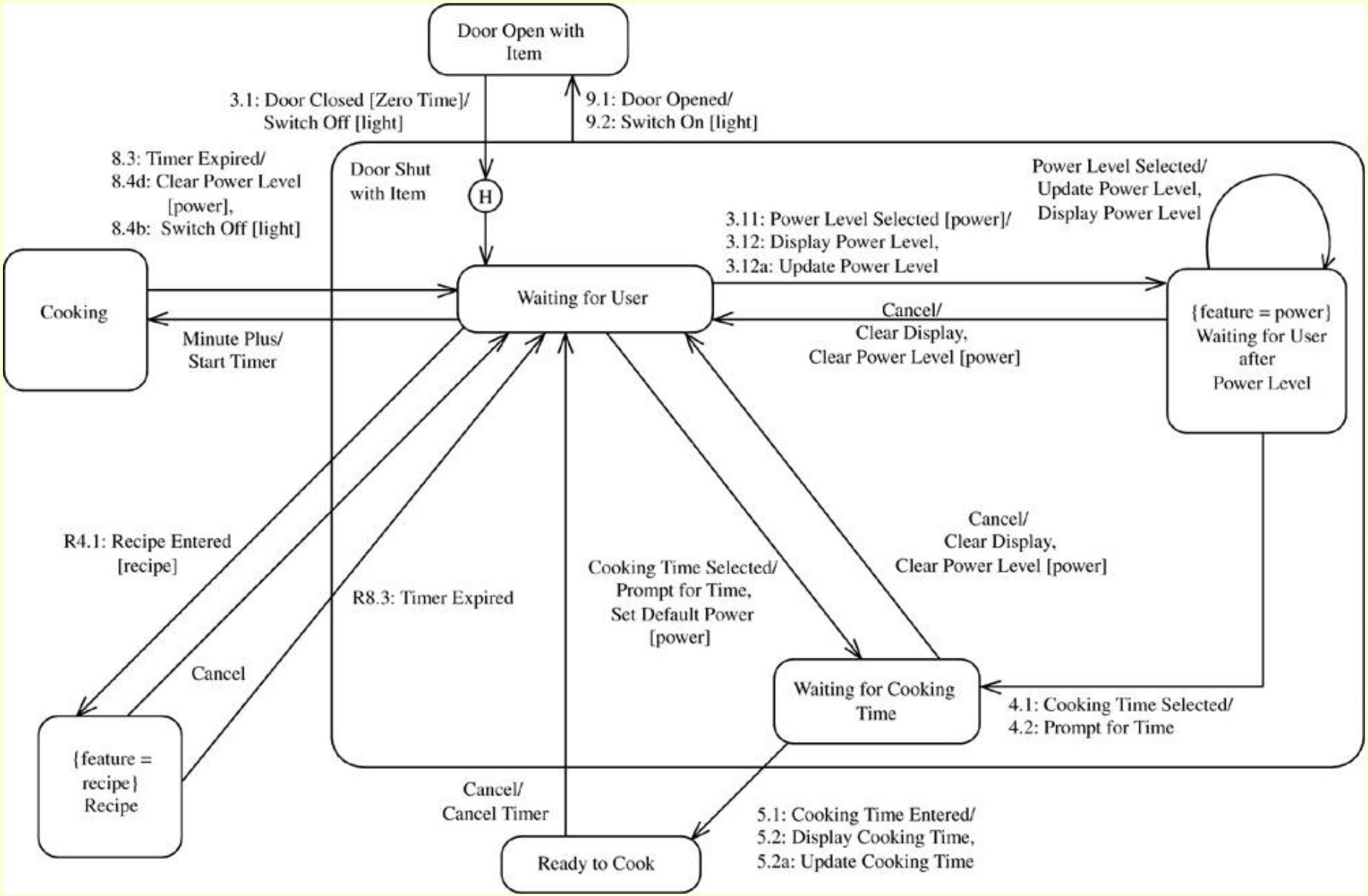
Modelagem Dinâmica



Feature alternativa - Pressionar a tecla Minute Plus antes de começar o cozimento.

Modelagem por Máquina de Estados Finitos


Door Shut
1.1: Door Opened
Door Closed
Door Open
Cancel/Cancel Timer
2.1: Item Placed
10.1: Item Removed
Door Closed [Time Remaining]
Door Open with Item
Door Opened/ Stop Timer
Door Opened
3.1: Door Closed [Zero Time]
9.1: Door Opened
Cooking
entry/6.2: Start Cooking
exit/8.4: Stop Cooking
8.3: Timer Expired
Door Shut with Item
Cancel/ Stop Timer
5.1: Cooking Time Entered/
5.2: Display Cooking Time,
5.2a: Update Cooking Time
Cancel/ Cancel Timer
6.1: Start/
6.2a: Start Timer
Ready to Cook
Cooking Time Entered/ Display Cooking Time, Update Cooking Time

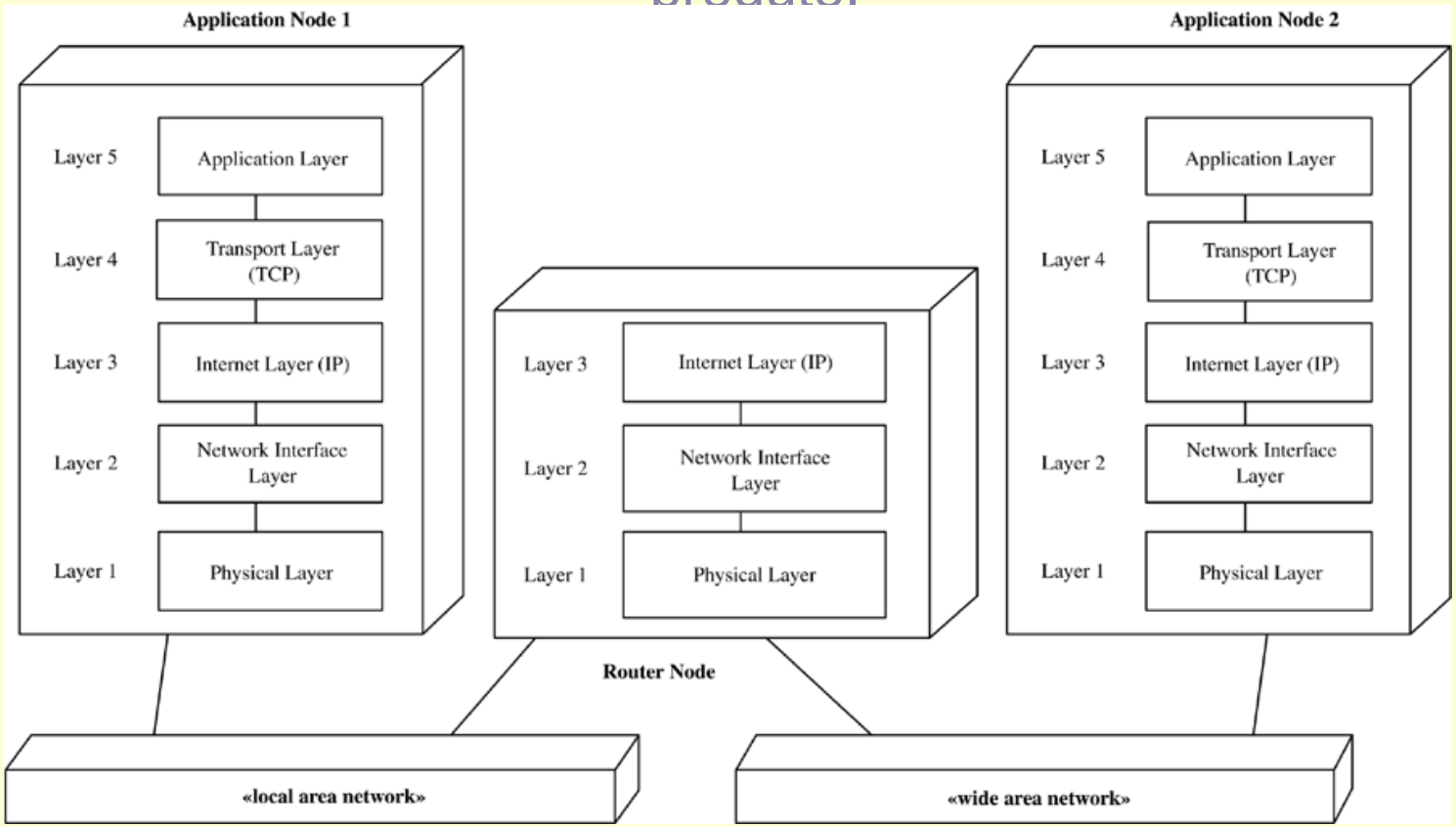

Door Open with Item
3.1: Door Closed [Zero Time]/ Switch Off [light]
9.1: Door Opened/ 9.2: Switch On [light]
8.3: Timer Expired/ 8.4d: Clear Power Level [power], 8.4b: Switch Off [light]
Door Shut with Item
H
Power Level Selected/ Update Power Level, Display Power Level
3.11: Power Level Selected [power]/ 3.12: Display Power Level, 3.12a: Update Power Level
Cooking
Waiting for User
{feature = power} Waiting for User after Power Level
Minute Plus/ Start Timer
Cancel/ Clear Display, Clear Power Level [power]
R4.1: Recipe Entered [recipe]
R8.3: Timer Expired
Cooking Time Selected/ Prompt for Time, Set Default Power [power]
Cancel/ Clear Display, Clear Power Level [power]
Cancel
Waiting for Cooking Time
4.1: Cooking Time Selected/ 4.2: Prompt for Time
{feature = recipe} Recipe
Cancel/ Cancel Timer
5.1: Cooking Time Entered/ 5.2: Display Cooking Time, 5.2a: Update Cooking Time
Ready to Cook

As Fases do PLUS

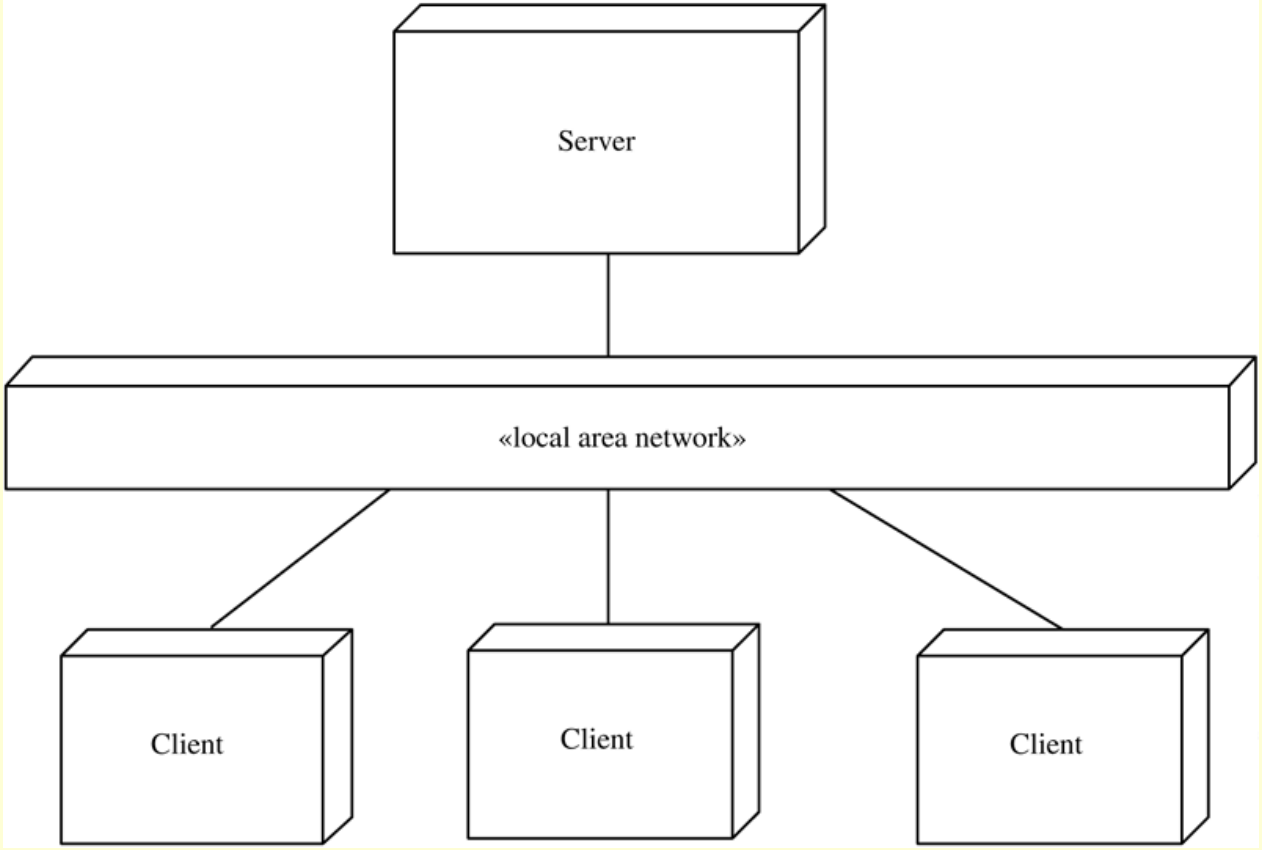
- Requisitos
  - Definição do Escopo
  - Modelagem dos Casos de Uso (similaridades e variabilidades)
  - Modelo de Características
- Modelo de Análise
  - Modelagem estática
  - Estruturação dos objetos
  - Modelagem Dinâmica
  - Modelagem por Máquina de Estados Finitos
  - Análise da dependência Características/Classes
- Modelo de Projeto
  - Projeto baseado em padrões de arquitetura
  - Projeto arquitetural da LPS
    - Arquitetura baseada em componentes

Padrão Camadas de Abstração

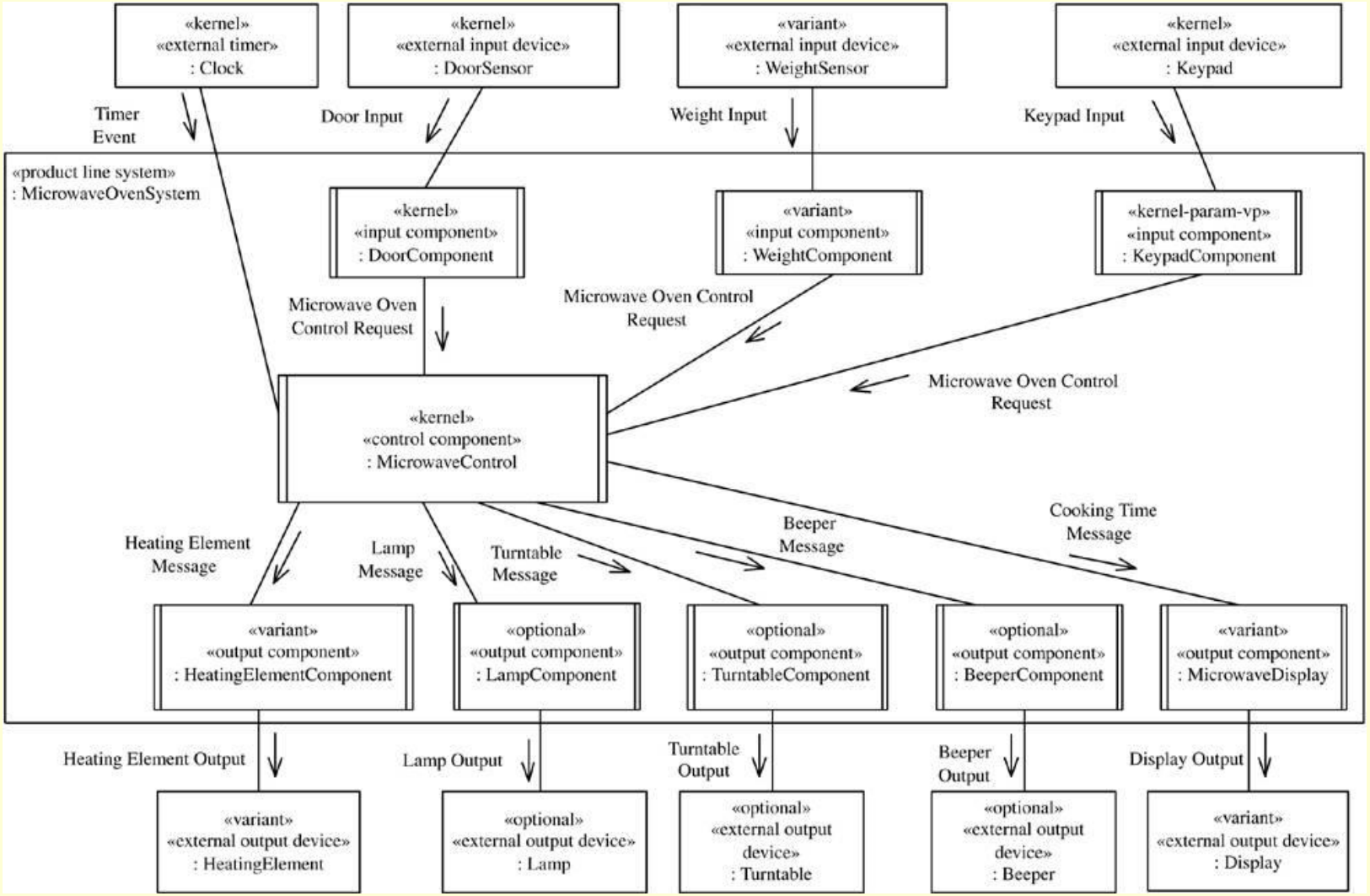
Os nós podem ser considerados membros de uma linha de produto.



Padrão Cliente/Servidor

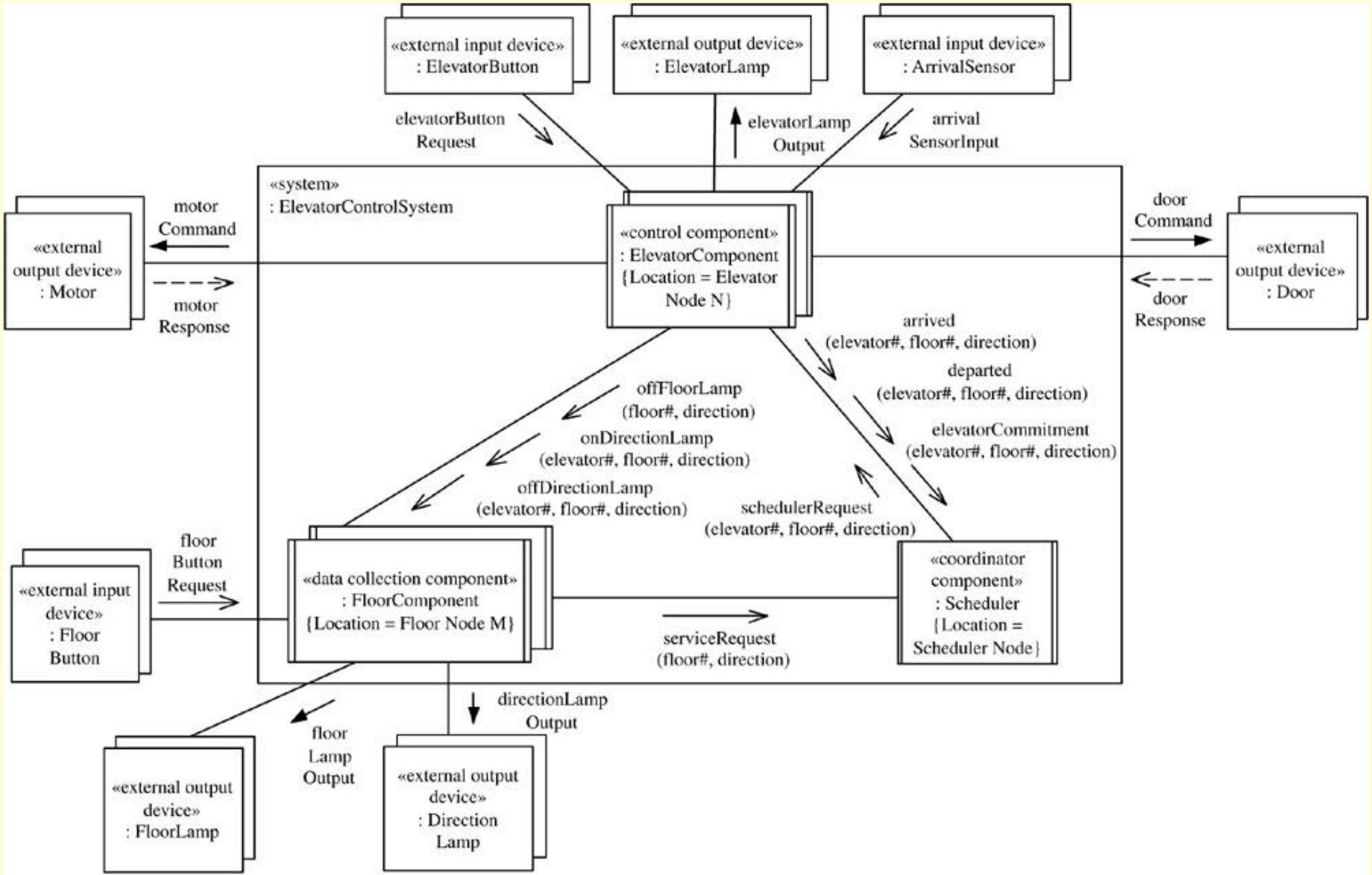


Padrão de Controle Centralizado

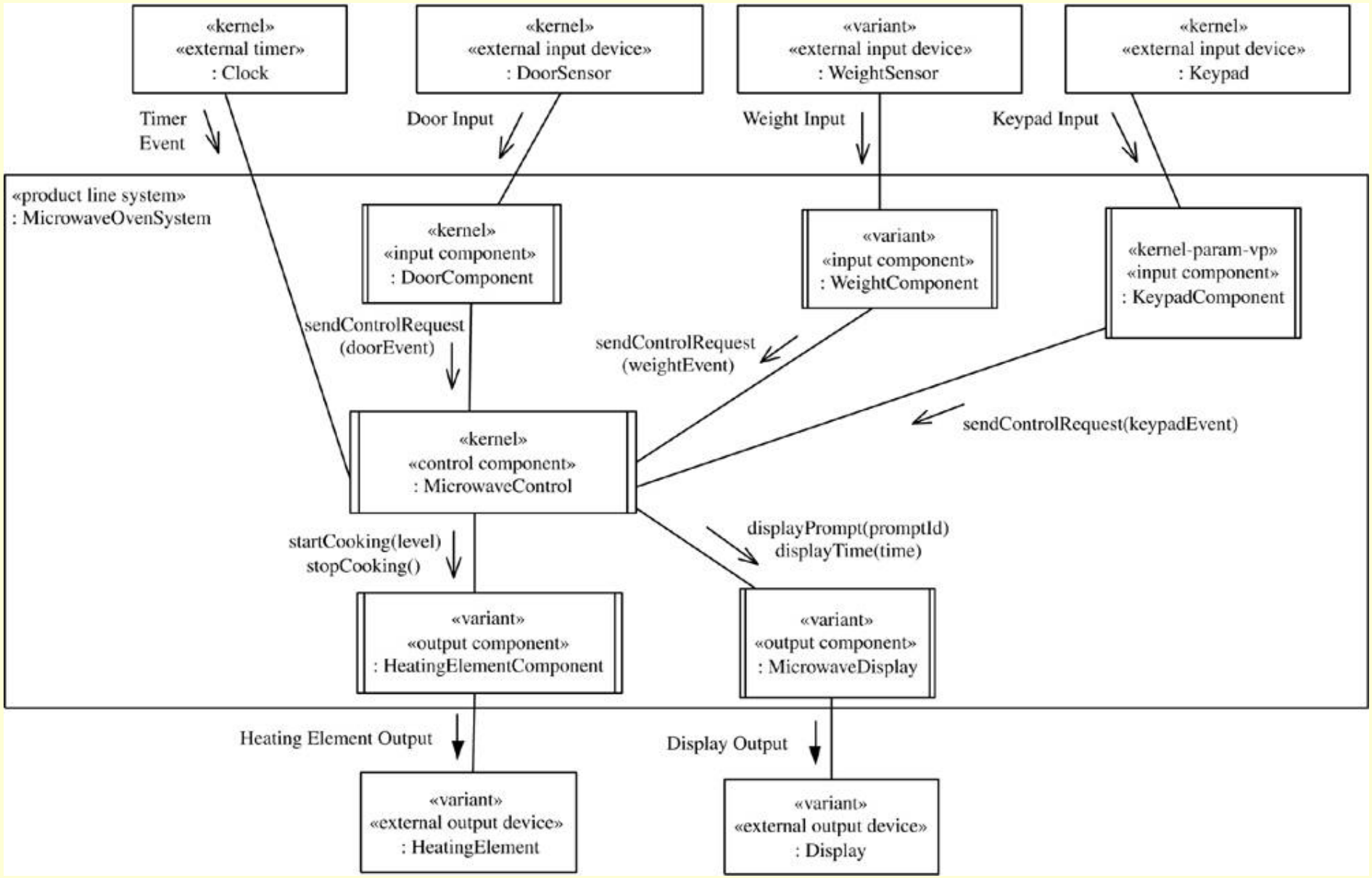


- Projeto arquitetural da LPS
  - Arquitetura baseada em componentes

Critérios de Estruturação de Componentes



Critérios de Estruturação de Componentes



Interfaces de Componentes: Fornecidas e Requeridas