2
Revisão da Literatura

Este capítulo apresenta os principais assuntos relacionados ao trabalho descrito por essa dissertação. Na Seção 2.1 são apresentados os principais conceitos de linha de produtos de software e forma de representação. A Seção 2.2 apresenta os principais trabalhos relacionados ao processo de configuração colaborativo de produtos: (i) Configuração em Estágios; (ii) Fluxo de Atividades para Configuração de *Features*; e (iii) Coordenação em Configuração Colaborativa. Por fim, na Seção 2.3 apresentamos uma revisão sobre os conceitos de sistemas multiagentes, que são bastante explorados para construção da abordagem proposta.

2.1
Linha de Produtos de Software

Linha de Produtos de Software (LPS) é uma abordagem de desenvolvimento de sistemas que provê uma maneira inteligente de projetar e implementar famílias de software com objetivos e funcionalidades semelhantes e que atendam determinado segmento do mercado [19]. Essa abordagem explora os pontos em comum entre um conjunto de sistemas enquanto gerencia pontos de variação de maneira sistemática.

Na engenharia de linha de produtos de software são distinguidas duas atividades principais: *engenharia de domínio* e *engenharia de aplicação* [3]. A engenharia de domínio consiste em desenvolver um conjunto de artefatos que podem ser configurados e combinados para criar diferentes produtos de software. Por outro lado, a engenharia de aplicação tem como objetivo resolver progressivamente as variabilidades providas pela engenharia de domínio, através de um *processo de configuração*. É durante a engenharia de domínio que funcionalidades são escolhidas ou são descartadas.

O uso de LPS para desenvolvimento de sistemas facilita a atividade de geração de novos produtos porque é baseado na ideia de utilização de um conjunto de artefatos reutilizáveis compartilhados [20], tais como: arquitetura, componentes, modelos, processos, etc. A maior implicação do reúso dos diversos artefatos é a utilização mais eficiente dos recursos da fábrica de software,

tendo como consequência direta: (i) redução no custo de desenvolvimento e manutenção; (ii) diminuição dos custos de projeto e implementação; e (iii) maior qualidade dos produtos finais [3]. Além disso, outro benefício consequente da utilização do paradigma é uma maior facilidade de estimar preços. Isso se dá, principalmente, pela característica de LPS de definir as funcionalidades com escopos bem delimitados. Outra motivação para utilização das abordagens de LPS é o suporte ao desenvolvimento de uma arquitetura flexível e reusável com intuito de permitir uma rápida e fácil customização de diferentes produtos que pertencem à mesma família de sistemas.

Dentre as várias abordagens propostas para a representação das variabilidades de uma LPS [19, 21, 22], o método FODA (*Feature-Oriented Domain Analysis*), proposto por [5], é um dos mais conhecidos. Desenvolvida no Instituto de Engenharia de Software (SEI), essa metodologia destaca-se por propor o modelo de *features*, que é amplamente utilizado para representar uma LPS. Na seção seguinte descrevemos o modelo de *features*.

2.1.1 Modelo de Features

De acordo com [23], *feature* é uma propriedade do sistema que é relevante de alguma maneira para algum *stakeholder*, onde um *stakeholder* pode ser um desenvolvedor, analista, administrador ou arquiteto de sistemas, um grupo de pessoas ou até uma empresa. Uma *feature* é usada para capturar os pontos em comum ou as variabilidades entre sistemas que compõem uma mesma linha de produtos. Hoje em dia, existem várias classificações para *features* [5, 23]. Neste trabalho, utilizamos a categorização de *features* proposta por [5] e estendida por [6], onde as *features* podem ser classificadas como: *individual* ou *grupo de features*. As *features* individuais podem ser *opcionais* ou *obrigatórias*. Grupos de *features* são caracterizados como *or-feature*, *alternative-feature*, *and-feature* ou, de uma forma mais geral, baseados em cardinalidade. Na Figura 2.1 podemos ver a representação gráfica de cada um dos tipos de *features*. Além disso, a descrição de restrições mais complexas sobre o estado de seleção de um grupo de *features* também é permitida. Em geral, essas restrições são definidas por expressões booleanas.

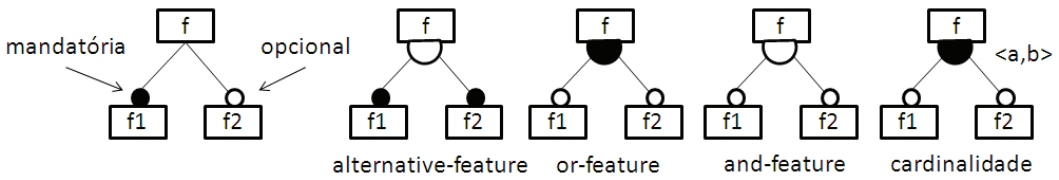

Figura 2.1: Tipos de *Features*.

Com essa representação, *features* são organizadas no formato de árvore em um modelo coerente, referenciado como diagrama de *features*, onde cada nível sucessivamente mais profundo na árvore corresponde a opções de configurações de granularidade mais finas. Modelos de *features* são diagramas de *feature* acrescidos de informações adicionais [6]. Os diagramas de *features* oferecem uma forma simples e intuitiva de se representar pontos de variação no sistema, além de representar restrições que *features* podem impor sobre outras. As *features* não precisam necessariamente ser mapeadas em artefatos de software concretos, podendo representar, também, funcionalidades transversais, que se estendem a diversos pontos do sistema – como *logging* ou sincronização – ou podendo ter propósitos puramente organizacionais e não ser mapeadas em nenhum tipo de artefato do sistema [24].

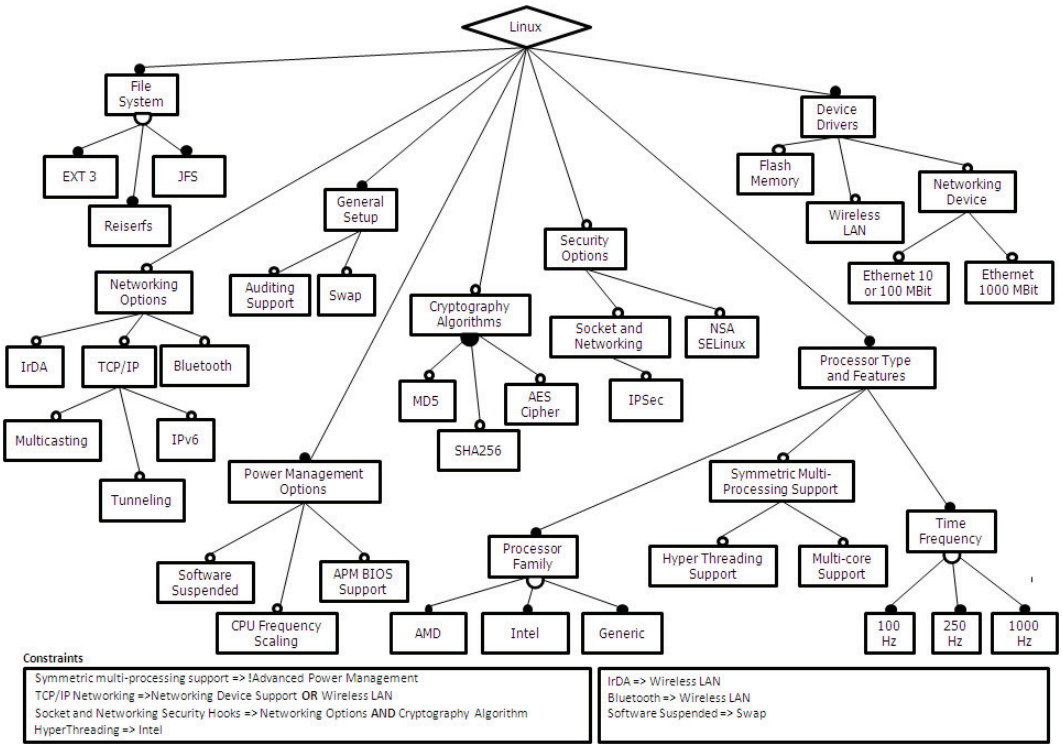

Figura 2.2: Modelo de *features* de um subconjunto do *kernel* do Linux.

Na Figura 2.2, temos como exemplo um modelo de *features* que representa um subconjunto do Kernel do Linux, seguindo as notações propostas em [5]. As *features* no Kernel do Linux são descritas usando a linguagem de configuração Kconfig. Utilizamos o mapeamento proposto em [25] para para transformar a notação Kconfig na notação proposta em [5]. Usaremos esse modelo de *features* durante diversas seções da dissertação, para exemplificar os conceitos que serão apresentados. Na raiz da árvore temos uma figura no formato de losango, que é chamado *nó conceito*. Os retângulos representam *features*. Os círculos na parte superior dos retângulos indicam se a *feature*

é obrigatória, ou não (círculo preenchido ou círculo vazio, respectivamente). "*Swap*"é um exemplo de *feature* opcional enquanto "*Processor Type and Features*"é de uma obrigatória. As *features* que possuem um círculo pela metade na sua parte inferior são chamadas de *features* de grupo. O círculo pela metade preenchido indica que pelo menos uma das *features* filhas deve está no produto final. Já o círculo vazio indica que apenas uma *feature* pode está presente na configuração. As restrições entre *features* são representadas por expressões lógicas.

2.2 Processo de Configuração Colaborativa de LPS

Em projetos industriais de grande porte, a configuração deve ser realizada de forma modular, com cada módulo focado em um assunto ou tarefa específica [12]. A identificação desses módulos é de grande interesse para a construção de uma configuração adequada. É comum um cenário em que nenhum dos *stakeholders* tem o conhecimento total para realizar a tarefa de configuração ou, até mesmo, um cenário em que o produto é desenvolvido por diferentes equipes [8, 6]. Portanto, o objetivo desse capítulo é prover uma visão geral dos principais trabalhos cujo foco é propor uma solução que suporte a realização da atividade de configuração de forma colaborativa.

2.2.1 Configuração em Estágios

Apresentado por Czarnecki *et al.* [24], o processo de *configuração em estágios* foi um dos primeiros trabalhos da literatura a propor uma solução para configuração de produtos realizada por um grupo de *stakeholders*. Esse trabalho parte do princípio básico de que um modelo de *features* descreve um espaço de configuração de uma família de sistemas e que um profissional pode configurar um sistema específico selecionando um conjunto de *features* que respeitem as restrições desse modelo. Czarnecki *et al.* [24] afirma que esse processo de especificação do membro de uma família de sistemas pode ser realizado em estágios, onde decisões realizadas em cada um desses eliminam decisões em estágios posteriores. Essa abordagem define que cada estágio recebe um modelo de *features* como entrada e produz outro modelo de *features* especializado como saída, onde o conjunto de sistemas descritos pelos segundo modelo é um subconjunto dos sistemas descritos pelo primeiro.

Dois conceitos muito importantes são definidos nessa abordagem: *especialização de modelo de features* e *configuração de modelo de features*. Segundo a terminologia descrita por [6], uma configuração consiste de um conjunto de

features que foram selecionadas e que respeitam as restrições de variabilidade definidas pelo diagrama de *features*. Podemos fazer uma analogia da relação entre diagrama de *features* e uma configuração com a relação entre classe e sua instância, do paradigma de orientação a objetos.

Por outro lado, o processo de especialização é definido como o processo de transformação que recebe um diagrama de *features* e o transforma em outro diagrama de *features*, de tal forma que o conjunto de configurações descritas pelo segundo diagrama é um subconjunto das configurações descritas pelo primeiro. É dito que o segundo diagrama é uma especialização do diagrama inicial. A especialização completa de um diagrama de *features* resulta em uma única configuração.

Durante o processo de especialização de um diagrama de *features* – desde o recebimento do modelo inicial até que seja produzida uma configuração final –, as suas variabilidades são eliminadas gradualmente, através de uma sequência de passos. Chamamos esses passos de *passos de especialização*. Os principais tipos de passos de especialização são: (i) refinar a cardinalidade de uma feature; (ii) refinar a cardinalidade do grupo; (iii) remover uma *feature* do grupo e (iv) selecionar uma *feature* de um grupo. Uma ilustração dos três primeiros passos pode ser visto na Figura 2.3.

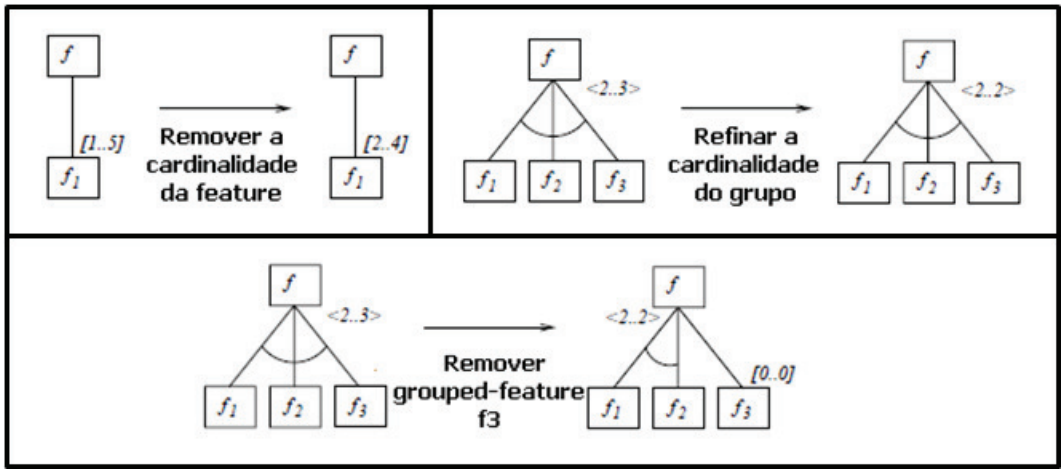

Figura 2.3: Passos de especialização na configuração em estágios.

Um contexto onde configuração em estágios é bastante aplicada é em cadeias de fornecimento de software (do inglês, *software supply chain*) [6]. Em uma cadeia de fornecimento de software, um fornecedor pode construir uma família de componentes de software, plataformas ou serviços especializados para diferentes integradores de sistemas. Essa família de produtos compartilha um conjunto de artefatos comum, além de oferecer toda a variabilidade demandada por cada membro da família. Nesse contexto, cada camada n que compõe a cadeia produtiva possui uma relação de fornecimento com a camada

$n + 1$. Essa estrutura da cadeia de fornecimento torna justificável a utilização de configuração em estágios [6], onde cada fornecedor e cada integrador de sistemas é capaz de eliminar um certo número de variabilidades, e produzir famílias especializadas de sistemas para a próxima camada.

Tendo base no modelo de *features* do kernel do Linux, representado pela Figura 2.2, podemos visualizar como funciona a abordagem de configuração em estágios através de um exemplo. Vamos utilizar um cenário de uma cadeia produtiva para construção de hardware embarcados para um determinado nicho de mercado de automação (observar Figura 2.4). Nesse exemplo, temos como *stakeholders*: (i) uma empresa vendedora de sistemas operacionais (VSO); (ii) uma empresa montadora de hardware embarcados (MHE); e (iii) um grupo de fábricas de componentes, que fornecem esses componentes à MHE.

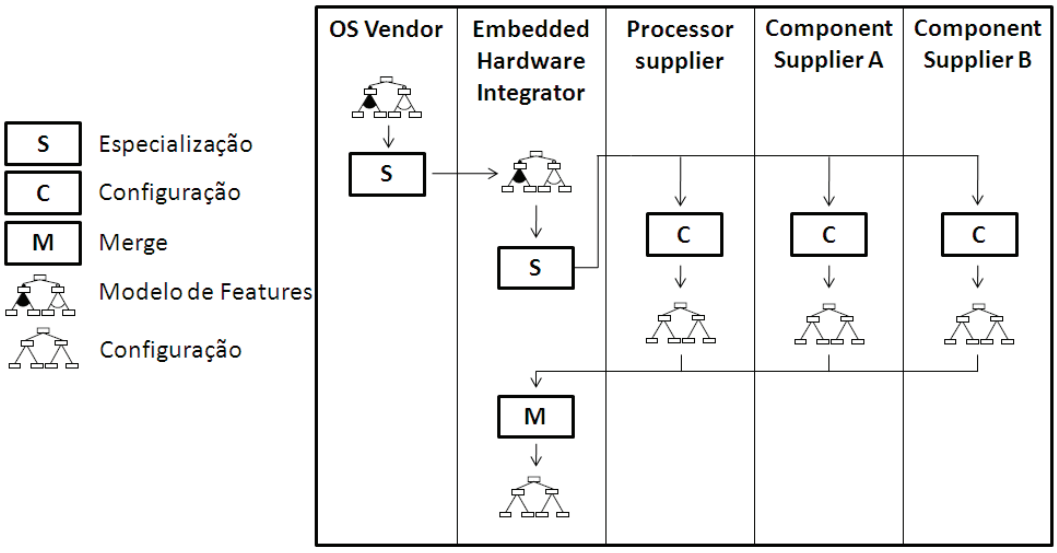

Figura 2.4: Configuração em estágios.

A VSO possui uma linha de produtos de sistemas operacionais, que pode ser configurada com objetivo de gerar sistemas operacionais que atendam os requisitos específicos de hardware de diversos fornecedores. Portanto, quando a MHE contrata a fornecedora de sistemas operacionais, VSO realiza a primeira especialização do modelo de *features* – como a VSO conhece sua linha de produtos e conhece parte dos requisitos da MHE, ela pode realizar um conjunto de seleções/exclusões de *features* com objetivo de atender esses requisitos. No segundo passo, a MHE realiza mais uma etapa de especialização, tendo como base alguns requisitos que são de conhecimento interno. No passo seguinte, ele distribui o modelo de *features* resultado dessa especialização a todos seus fornecedores de componentes. Esses, por sua vez realizam, cada um, uma configuração que atenda as características de seus componentes. Por fim, a MHE obtém esse conjunto de configurações e aplica uma operação de

merge, para produzir uma configuração final. Após essa etapa final, o sistema operacional pode ser gerado e entregue para MHE instalar em seu hardware.

Dessa forma, o trabalho de configuração em estágios apresenta resultados bastante importantes para a área de configuração colaborativa. Essa abordagem é bastante aplicada para situações (como mostrada acima) onde uma sequência de passos de especialização é suficiente para produzir uma configuração desejada. No entanto, infelizmente, quando há a necessidade de interação de forma mais complexa entre os *stakeholders*, essa abordagem falha [12]. Nas próximas seções serão apresentados alguns desses cenários.

2.2.2 Fluxo de Atividades para Configuração de Features

Antes de discutirmos sobre a abordagem de fluxo de atividades para configuração de *features*, vamos analisar um cenário baseado no exemplo da Seção 2.2.1. Suponha que a vendedora VSO irá fornecer um sistema operacional para a montadora de hardware embarcado MHE. Para isso, MHE deverá entregar uma configuração para o modelo de *features* que VSO está fornecendo. Com esse intuito, MHE alocou um engenheiro de sistemas do seu corpo de funcionários para ser o responsável pela configuração. Além disso, foram colocados à disposição profissionais de três áreas da empresa: (i) área de redes e comunicação; (ii) área de hardware; e (iii) área de segurança. Como parte de seu processo de venda, VSO realiza uma especialização do modelo de *features*, com base nos requisitos que MHE a apresentou – como, por exemplo, relativos à plataforma de hardware. Além disso, questões como maturidade das *features* para um determinado hardware podem ser consideradas nesse passo. De posse desse modelo de *features* especializado, o engenheiro de sistemas, com conhecimento maior sobre produto que sua empresa fabrica, realiza decisões e produz um segundo modelo de *features*, mais refinado. Porém, como seu conhecimento não se estende a todos os detalhes do projeto de hardware, o engenheiro solicita aos profissionais especialistas das três áreas que realizem decisões sobre algumas *features*, conforme suas especialidades. Porém, por questões técnicas, as áreas podem requerer que o engenheiro de sistemas produza algumas seleções ou exclusões.

A abordagem de configuração em estágios, conforme definida em [24, 6] e apresentada na Seção 2.2.1, é muito restritiva para tratar um cenário complexos como esse [12]. Essa abordagem assume que o processo de configuração é puramente sequencial, mas esse não é o caso: (i) as áreas técnicas realizam a configuração em paralelo; e (ii) a configuração é interativa entre o engenheiro de sistemas e as áreas técnicas. Além disso, poderia existir a situação em

que a participação de um *stakeholder* seja opcional. Suponha que um ator representando um revendedor de VSO. Esse revendedor poderia realizar uma especialização do modelo, levando em consideração questões comerciais, caso seja de seu interesse.

Portanto, levando em conta essas limitações, Hubaux *et al.* [12] propõe uma abordagem que suporte a configuração em cenários mais complexos do que meras sequências de especializações. A ideia principal dessa abordagem é utilizar uma linguagem de modelagem de *workflows* que possa descrever o fluxo de especialização que um modelo de *features* sofre até que seja produzida uma configuração final. Nesse trabalho é adotada a linguagem de descrição de *workflow* YAWL (Yet Another Workflow Language) [26]. A Figura 2.5 mostra como o cenário de configuração descrito no início dessa seção é representado e organizado com essa abordagem.

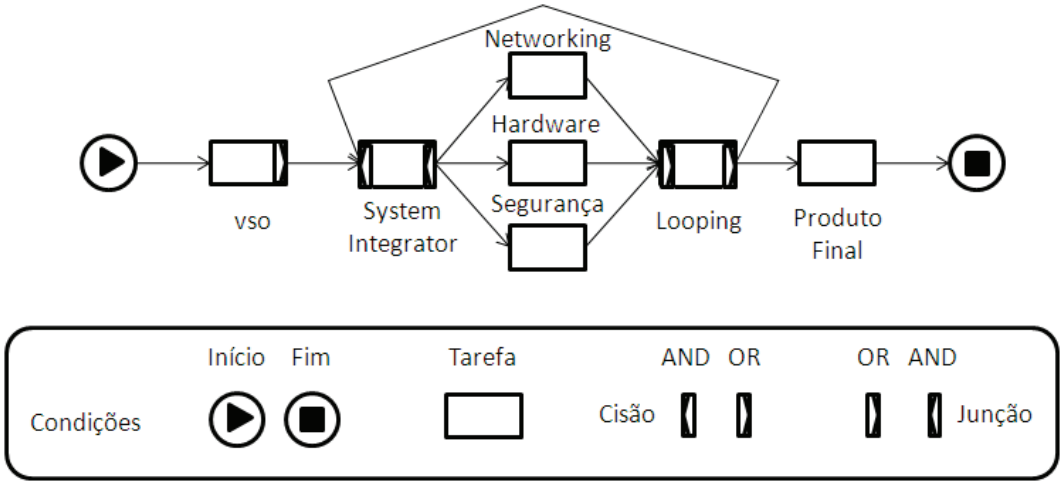

Figura 2.5: Fluxo de atividades para configuração de *features*.

As principais construções da linguagem YAWL são *condições* e *tarefas*. As condições são representadas por círculos e as tarefas por retângulos. Existem duas condições especiais: *início* e *fim* do fluxo. Cada tarefa denota uma atividade de configuração e é anotada pelo nome do papel do *stakeholder* que executará a atividade. Além disso, são oferecidos mais dois construtores de definição de comportamento da tarefa: *cisão* e *junção*. Ambos construtores podem ser do tipo *XOR*, *OR* ou *AND*. Por exemplo, o fluxo em "*Engenheiro de Sistemas*"possui uma cisão do tipo *AND*, dividindo o fluxo para as três áreas técnicas, e uma junção do tipo *XOR*, recebendo o fluxo de VSO ou da interação com essas áreas. A cisão do tipo *AND*, saindo da tarefa "*Engenheiro de Sistemas*", significa que as três áreas realizarão suas tarefas em paralelo.

Portanto, a organização das atividades do processo de configuração é feita utilizando a abstração de *workflow*. Formalmente, um *workflow* de configuração é definido conforme a seguir.

Definição 2.1 (Workflow [26]) *Um* workflow *é definido como uma tupla* $(C, i, o, F, T, split, join)$ *onde* C *denota um conjunto de condições,* $i \in C$ *é a única condição de início e* $o \in C$ *a única de fim (único ponto de entrada e saída do fluxo de atividades).* F *é a relação de fluxo entre condições e tarefas e* T *denota o conjunto de tarefas.* $split$ *é uma função que determina o tipo do comportamento de cisão de uma tarefa (isto é, AND, OR ou XOR) e, analogamente,* $join$ *é a função que determina o tipo do comportamento de junção de uma tarefa.*

Assim, essa abordagem propõe uma solução mais genérica do que a suportada pela configuração em estágios. Com ela é possível a utilização de construtores que simulam repetições e seleção de fluxo (*AND*, *OR*, *XOR*). Uma implicação bastante importante disso é que os *stakeholders* não são necessariamente obrigados a realizar toda sua atividade de configuração em um único passo, permitindo uma melhor interação entre eles. No entanto, assim como na configuração em estágios, também é considerado que decisões realizadas em atividades anteriores restringem decisões nos modelos de *features* especializados nas atividade posteriores do fluxo.

Portanto, de forma resumida, nessa abordagem as atividades de configuração são coordenadas através de um *workflow*, onde cada uma dessas tarefas estão associadas à configuração de um conjunto de *features*. Conforme uma atividade é executada, as atividades de configuração seguintes dos próximos *stakeholders* são definidas pela avaliação das condições do *workflow*. Esse fluxo de atividades é executado até que a condição de *fim* seja alcançada, produzindo a configuração desejada.

Apesar de essa abordagem poder ser aplicada em cenários bem mais complexos que a proposta de configuração em estágios suporta, existem outros cenários em que sua utilização pode ser difícil ou, até mesmo, impraticável. Em situações com grande quantidades de *stakeholders*, a definição do fluxo de atividades pode ser uma tarefa tão complexa quanto à configuração em si. Além disso, são permitidas construções que possibilitam atividades em paralelo, porém esse relaxamento torna possível a produção de estados de inconsistência. Infelizmente, essa abordagem não apresenta como deve ser feita a integração dessas decisões nessa situação.

Outra situação que pode ocorrer é mudança de requisitos durante o processo de configuração. Nesse cenário, pode surgir a necessidade de revisão de configuração por parte de *stakeholders* que já realizaram suas atividades e o fluxo não previu – adicionando uma estrutura de repetição, por exemplo. De fato, descrever um fluxo de atividades visando situações de exceção pode não ser uma tarefa muito fácil. Nesse caso, haveria a necessidade da tomada de

decisão sobre próximos passos pelos próprios *stakeholders*, podendo conduzir a configuração para um estado de inconsistência. Como exemplo, podemos supor que durante uma atividade em paralelo, um *stakeholder* perceba uma situação de exceção e decida reenviar para um *stakeholder* de passos iniciais. Porém ele, por falta de conhecimento sobre o fluxo, não se preocupa com a operação de *merge* com as decisões de outros *stakeholders*.

2.2.3 Coordenação em Configuração Colaborativa de Produtos

Um trabalho bastante importante na área de engenharia de linha de produtos de software e, mais especificamente, em engenharia de aplicação é o de *Coordenação em Configuração Colaborativa de Produtos* [10, 11]. Nesse trabalho, Mendonça *et al.* propõe um abordagem baseada em processos que provê uma coordenação das equipes de trabalho tomadoras de decisão. Como parte da solução são definidos três conceitos principais: *conjunto de decisão*, *papel de decisão* e *conflito de decisões*.

Definição 2.2 (Conjunto de decisões) *Um conjunto de decisão engloba um grupo de* features *que serão configuradas por tomadores de decisão interpretando papéis específicos.*

Definição 2.3 (Papel de decisão) *Papéis de decisão são responsabilidades ou perspectivas sobre determinados assuntos relacionados com o processo de configuração.*

Definição 2.4 (Conflito de decisões) *É dito que um conflito entre decisões ocorre quando duas ou mais* features *(tipicamente atribuídas a diferentes tomadores de decisão) contêm dependências explícitas ou implícitas, e o estado de configuração dessas* features *desrespeita, pelo menos, uma dessas dependências.*

Os conjuntos de decisão são os componentes-chave da solução proposta por essa abordagem. Um conjunto de decisão válido deve conter pelo menos uma decisão em aberto. Um papel de decisão pode estar associado a um ou mais conjuntos de decisão. Além disso, uma pessoa interpretando um papel é responsável por realizar a configuração das *features* relacionadas aos conjuntos de decisão associados a esse papel. Note que um *stakeholder* só pode participar diretamente do processo de configuração se a ele for atribuído um papel de decisão. Um tipo especial de *stakeholder*, responsável pelo processo de configuração do produto, é definido por essa abordagem e referenciado como *Gerente de Produto*.

Utilizando o nosso modelo de *features* do kernel do Linux, a Figura 2.6 mostra um exemplo que facilita o entendimento desses três conceitos. O modelo de *features* foi dividido em sete conjuntos de decisão: *Ln*, *Fs*, *Dv*, *Gs*, *Pm*, *Pr* e *No*. Além disso, associados a cada um desses conjuntos de decisão estão papéis de decisão. Por exemplo, para o conjunto *No* pode ser criado um papel chamado "*Administrador de Redes*", cuja responsabilidade é escolher as *features* relacionadas aos protocolos de comunicação que estarão presente no sistema operacional final. Um conflito de decisões ocorreria se o *stakeholder* interpretando o papel relacionado ao conjunto de decisão *Pr* selecionar a *feature* "*Symmetric Multi-Processing Support*"e um segundo *stakeholder*, interpretando o papel relacionado à *Pm*, selecionar a *feature* "*APM BIOS Support*". Isso se daria devido à restrição imposta pelo modelo de features: *Symmetric multi-processing support –> !Advanced Power Management (APM) BIOS Support.*

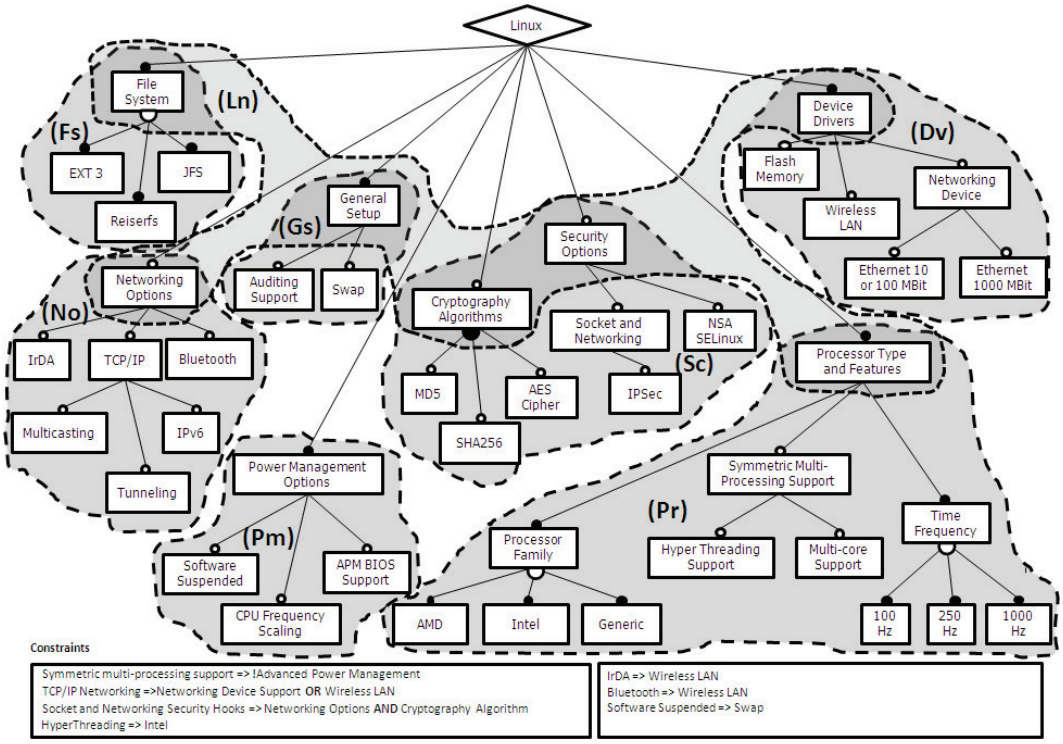

Figura 2.6: Modelo de *features* dividido em conjuntos de decisão.

Baseado nesses conceitos, o processo de configuração colaborativa consiste de duas fases. A Figura 2.7 apresenta uma representação gráfica da abordagem. Na fase 1, o objetivo é produzir planos de coordenação das atividades de configuração. O gerente de produto conduz essa fase, devido ao seu conhecimento privilegiado de quem deve fazer parte do processo de configuração, quais grupos podem trabalhar juntos, além de potenciais conflitos.

O primeiro passo da fase 1 é chamada de *particionamento*. Nesse passo ocorre a divisão do universo de decisões de configurações e o agrupamento em

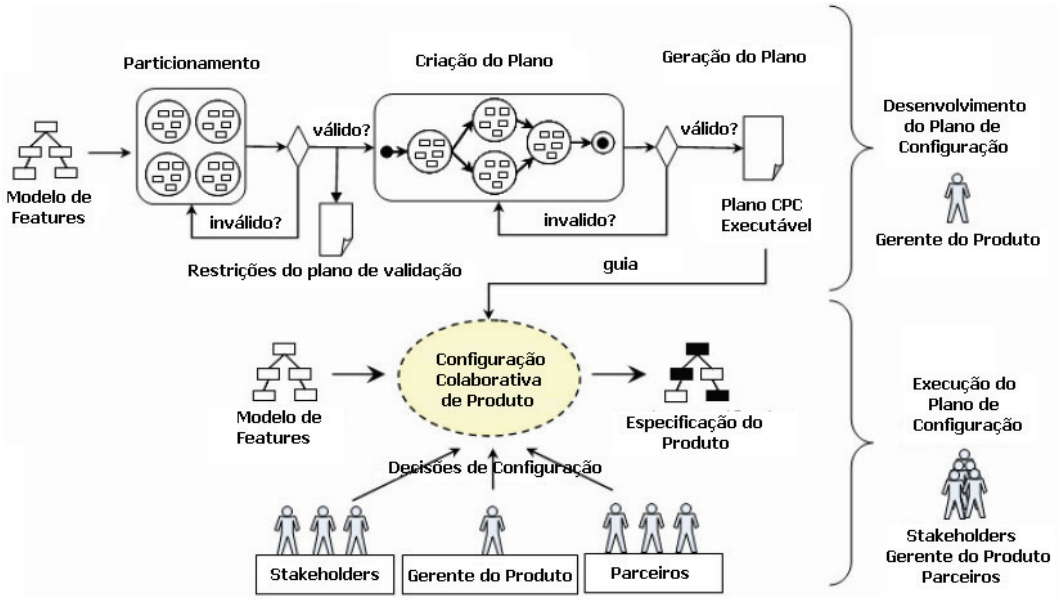

Figura 2.7: Coordenação em configuração colaborativa de produtos.

unidades menores – os conjuntos de decisão – baseado critérios particulares (ex. domínio de conhecimento). Além disso, nesse passo também é realizada a identificação dos papéis que serão responsáveis por esses conjuntos de decisão. O resultado desse passo deve ser garantir que a união desses conjuntos cubra todo o modelo de *features*, isto é, todo o espaço de decisão. A Figura 2.6 é um exemplo do resultado do passo de particionamento. Note que existem *features* que pertencem a dois conjuntos de decisão. Elas são chamadas de *pontos de junção*. A abordagem define que essas *features* devem ser atribuídas a um dos conjuntos de decisão na fase de configuração.

Como os conjuntos de decisão podem ser vistos como agrupamentos de *features*, e essas *features* podem possuir dependências entre si, esse conceito de dependência pode ser estendido aos conjuntos de decisão. Mendonça *et al.* define dois tipos de dependências entre conjuntos de decisão: *forte* e *fraca*.

Definição 2.5 (Dependência forte) *Um conjunto de decisão DS_a é dito fortemente dependente de um conjunto de decisão DS_b quando uma única decisão em DS_b pode impactar todas as decisões em DS_a.*

Definição 2.6 (Dependência fraca) *Dois conjuntos de decisão DS_a e DS_b são ditos fracamente dependentes quando alguma decisão em DS_a pode impactar alguma decisão em DS_b, e vice-versa.*

Um conjunto de decisão que possui todas suas *features* sendo filhas de uma *feature* que está em outro conjunto de decisão é um exemplo de dependência forte. Na Figura 2.6 podemos verificar que o conjunto de decisão *No*

é fortemente dependente do conjunto *Ln* porque quando a *feature* "*Networking Options*"é excluída da configuração, todas as *features* de *No* são excluídas. Por outro lado, *No* e *Dv* são fracamente dependentes por causa da restrição *IrDA –> Wireless LAN*. A Figura 2.8 mostra uma visão gráfica das dependências fortes e fracas entre todos os conjuntos de decisão do modelo de *features* do kernel do Linux.

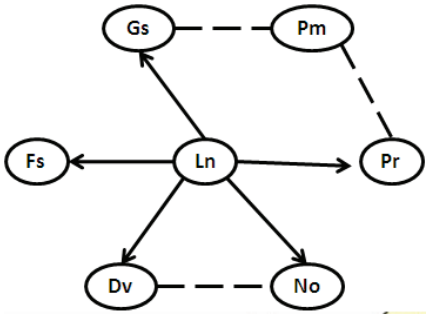

Figura 2.8: Dependência entre conjuntos de decisão.

Em seguida, o segundo passo da fase 1 é a *criação de planos*. A criação de planos é uma atividade também realizada pelo gerente de produtos. Esse *stakeholder* elabora um plano com base no resultado do passo anterior e no seu conhecimento sobre o modelo de *features*. Esse plano define *sessões de configuração* e a ordem de suas execuções. Em uma sessão de configuração podem existir um ou um agrupamento de conjuntos de decisão, permitindo que os *stakeholders* realizem suas decisões como um time. Para garantir que não sejam produzidas configurações inválidas, o plano de configuração é submetido a uma validação automática. A validação de um plano é realizada avaliando se as seguintes regras estão sendo respeitadas:

- **Regra 1.** Se um conjunto de decisão DS_b é fortemente dependente de DS_a, então DS_a deve preceder DS_b.
- **Regra 2.** Se dois conjuntos de decisão DS_a e DS_b são fracamente dependentes, eles podem ser arranjados em (i) sequência ou (ii) em paralelo, desde que imediatamente seguidos por uma sessão de *merge*.

Assim, um plano de configuração colaborativa de produto é um estrutura que agrupa conjuntos de decisão de forma sequencial ou paralela, similar a um *workflow* e construída pelo gerente de produto. A Figura 2.9 mostra dois planos de configuração *A* e *B* para a linha de produtos de kernel do Linux. O plano *A* é inválido por dois motivos: (i) *Fs* é fortemente dependente de *Ln*, portanto, segundo a *regra 1*, *Ln* deveria precedê-la; (ii) os conjuntos de decisão *Pm* e *Pr* estão desrespeitando a *regra 2*, pois são fracamente dependentes e estão

em paralelo, não seguidas de uma operação de *merge*. O plano B é um plano válido que corrige os problemas de A.

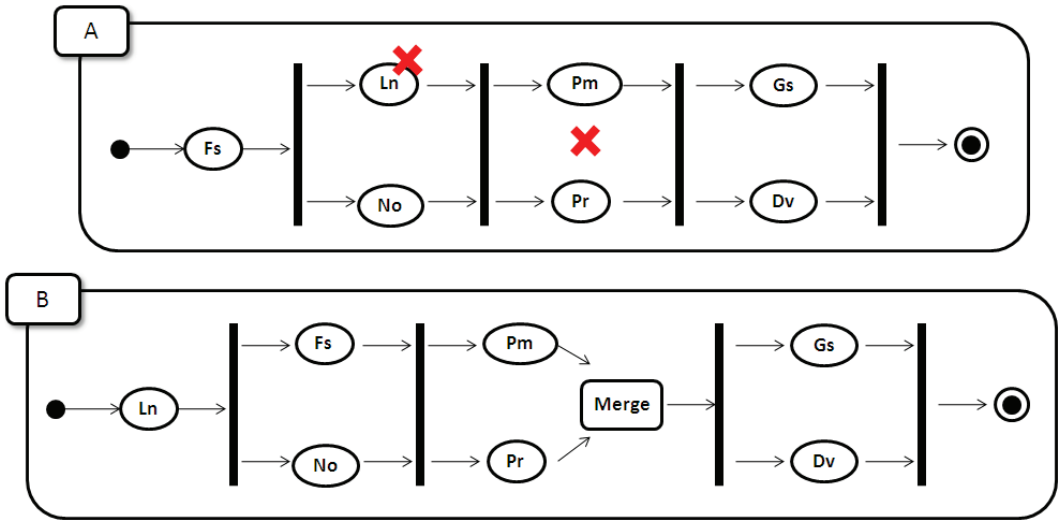

Figura 2.9: Planos de configuração colaborativa de produtos.

O último passo da fase 1 é a geração de uma representação executável do plano de configuração. Essa representação é capaz de indicar os passos seguintes de configuração. Por exemplo, supondo que durante sua etapa de configuração, o *stakeholder* interpretando o papel responsável pelo conjunto de decisão *Ln* decidir por não selecionar "*Networking Options*", não será necessário a sessão de configuração de *No*. Esse passo é a compilação do fluxo de configuração para poder ser suportado por uma ferramenta.

Dessa forma, uma vez que os três passos da fase 1 foram completados e corretamente validados, a fase 2 pode ser inicializada. A fase 2 é onde ocorre, de fato, a configuração do produto, através da execução dos planos de atividades. O resultado final é uma configuração de um produto onde diversos *stakeholders* tiveram participação direta de decisão.

A abordagem de coordenação em configuração colaborativa de produtos, apesar de abordar o problema de forma similar às abordagens anteriores, utilizando fluxos de atividades e permitindo construções mais complexas, apresenta conceitos bastante importantes que facilitam a construção desses fluxos. Os conjuntos de decisão, por exemplo, são abstrações que facilitam a modularidade e o desenho desses fluxos de atividades. No entanto, essa abordagem não apresenta soluções para os problemas citados em 1.1.

2.3 Engenharia de Software Orientada a Agentes

A Engenharia de Software Orientada a Agentes aborda a especificação e projeto de sistemas complexos e distribuído, utilizando os conceitos de agentes,

objetivos, planos, ambiente, comunicação, entre outros [27]. O conceito de agente pode ser definido como uma entidade situada em um ambiente, capaz de desempenhar suas tarefas e comunicar com outros agentes para atingir seus objetivos [17]. De forma geral, os agentes de software são uma abstração que possuem as seguintes propriedades [28]:

- **Autonomia.** Os agentes operam sem nenhuma forma de intervenção humana direta ou de outras entidades; e tem algum tipo de controle sobre suas ações e sobre seu estado interno;
- **Habilidade Social.** Os agentes interagem com outros agentes (e possivelmente com outros humanos) através de alguma linguagem de comunicação pré-definida;
- **Reatividade.** Os agentes percebem o ambiente em que estão inseridos e podem responder às mudanças que ocorrem nele;
- **Pró-atividade.** Os agentes não agem simplesmente em resposta a mudanças do ambiente, mas eles são capazes de exigir um comportamento orientado a objetivos, tomando a iniciativa;

Um sistema multiagentes pode ser definido como uma rede com baixa acoplagem de resolvedores de problemas que interagem entre si para resolver problemas que estão além da capacidade ou do conhecimento de cada resolvedor [29]. Esses resolvedores de problemas, também chamados de agentes, são autônomos e podem ser heterogêneos. As características dos sistemas multiagentes são [30]: (i) cada agente tem informações ou capacidades incompletas para resolver o problema e, portanto, tem perspectivas limitadas; (ii) não existe um controle global do sistema; (iii) os dados estão descentralizados; e (iv) a computação é assíncrona.

Um ponto bastante estudado dentro do tema sistemas multiagentes é a *negociação* [31, 32, 33]. Negociação é uma forma chave de interação que permite que um grupo de agentes alcance um acordo mútuo, independentemente de suas crenças, objetivos ou planos [34]. A área de negociação é extensa e aplicável em diversos cenários [17]. Agentes tipicamente representam entidades com interesses próprios e a forma principal de interação entre eles é a negociação.

Além de sistemas que demandam interação entre agentes, uma aplicação bastante comum e importante de agentes computacionais é para interação com seres humanos, principalmente quando atuando como auxiliadores. Esse conceito é chamado de *assistência pessoal* [18]. Nessa visão, um agente autônomo tem como objetivo principal reduzir trabalhos e a sobrecarga de informações.

Os agentes podem atuar em atividades repetitivas e automatizáveis que, de outra forma, exigiriam muito esforço humano.

Portanto, utilizando uma abordagem baseada em agentes, problemas complexos podem ser decompostos em agentes autônomos, pró-ativos e reativos, com habilidades sociais. Nesse trabalho, utilizamos fortemente os conceitos e os benefícios da utilização de agentes. Os agentes guiam os *stakeholders* por toda a atividade de configuração e os suportam nas interações dinâmicas. Agentes são aplicados para retirar do *stakeholder* a complexidade de raciocínio sobre as restrições do modelo de *features*; para prover uma infraestrutura de comunicação entre *stakeholders*; na validação da configuração; e resolução de conflitos.

2.4 Conclusão

Esse capítulo apresentou uma revisão sobre os principais conceitos utilizados para o desenvolvimento da abordagem de configuração dinâmica e colaborativa que é proposta por essa dissertação. Apresentamos uma visão geral sobre linha de produtos de software e detalhamos uma das mais utilizadas técnicas para sua representação: o modelo de *features*. São discutidos também os principais trabalhos relacionados que propõem soluções para o suporte à configuração colaborativa de produtos de uma família de sistemas. Detalhamos a visão de cada abordagem e seus conceitos principais, como configuração e especialização, passos de configuração, conjunto de decisão, fluxo de configuração, entre outros. Apresentamos os pontos fortes e fracos de cada abordagem. Por fim, uma discussão sobre engenharia de software orientada a agentes é apresentada. Conceitos como autonomia, habilidade social, reatividade e pró-atividade são brevemente discutidos, assim como negociação e um conceito bastante explorado nesse trabalho: *assistência pessoal*.