3
Configuração Colaborativa de Linha de Produtos de Software

Tendo em vista os problemas e limitações apresentados na Seção 1.1 e Seção 2.2, propomos uma nova forma de abordar a configuração colaborativa de produtos. Essa abordagem permite que os *stakeholders* tomadores de decisão atuem de forma interativa e dinâmica, suportando os cenários de configurações discutidos, além dos com complexidade maiores, e garantindo a produção de um configuração final válida. O trabalho propõe uma maneira nova de tratar a coordenação da atividade de configuração colaborativa de linha de produtos de *software*, além provê suporte aos *stakeholders* através sugestões de ações de configuração durante todo o processo de configuração do produto.

3.1
Visão Geral da Abordagem

A abordagem proposta é dividida em três fase onde participam dois tipos distintos de atores: (i) o *gerente de produto*; e (ii) os *stakeholders configuradores*. O gerente de produto tem a responsabilidade sobre o produto final a ser gerado, identificando quem deve participar da configuração, definindo as responsabilidades desses *stakeholders* e verificando se a configuração final reflete os requisitos impostos sobre o produto. Os *stakeholders* são aqueles que possuem o conhecimento específico sobre algum subconjunto do domínio que envolve a família de produtos. São eles os responsáveis, de fato, por tomar decisões sobre a configuração do modelo de *features*.

Como base da solução, utilizamos os conceitos de *conjunto de decisão* e *papéis*, conforme descritos nas Definições 2.2 e 2.3. A Figura 3.1 mostra uma visão geral da abordagem proposta.

A primeira fase do processo de configuração é realizada pelo gerente de produto. Esse *stakeholder* é um profissional que tem contado direto com o cliente que está adquirindo um sistema da linha de produtos de software. Portanto, ele geralmente tem conhecimento sobre os requisitos principais desse cliente, além de ter um conhecimento geral da linha de produtos e do domínio onde o produto será aplicado. Dessa forma, com essa experiência, ele define grupos de *features* disjuntos (os conjuntos de decisão) conceitualmente ou

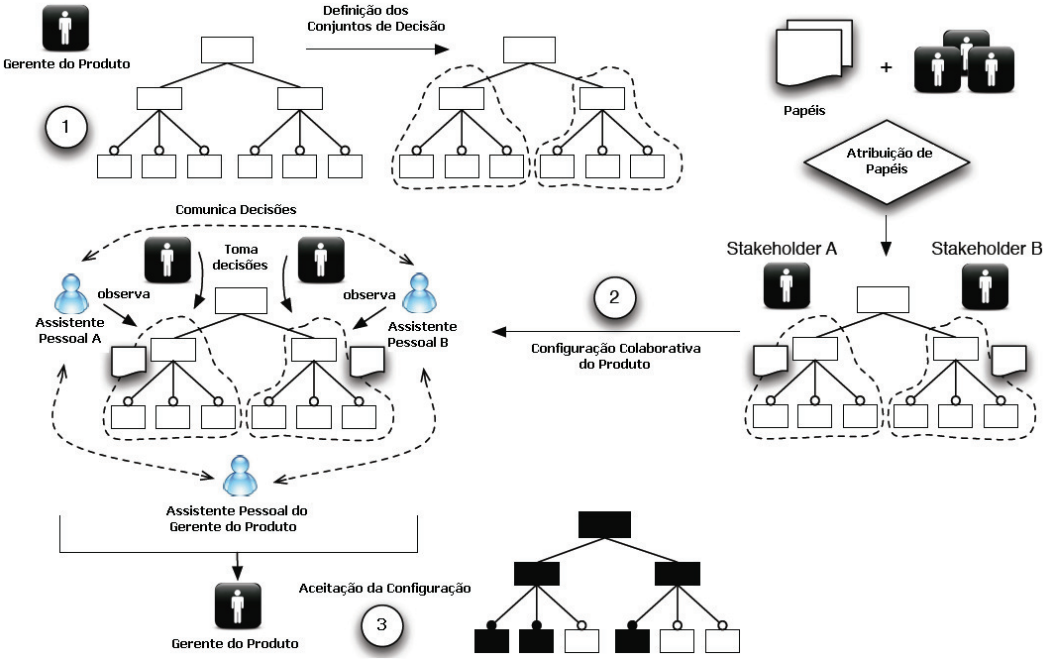

Figura 3.1: Visão geral da proposta de configuração dinâmica e distribuída de produtos.

logicamente relacionadas. Assumimos nesse trabalho que esse agrupamento se dá por áreas de conhecimento (ex. administração de redes, banco de dados, etc). Papéis de decisão são associados a esses conjuntos de decisão. Os papéis podem ser vistos como visões parciais sobre o domínio da família de sistemas e como uma autoridade sobre um conjunto de *features*. A união dessas perspectivas corresponde ao domínio completo. Atributos de prioridade podem ser atribuídos aos papéis. O último passo dessa primeira fase é a atribuição desses papéis a cada um dos *stakeholders*.

Na segunda fase ocorre, de fato, a configuração do produto. Os *stakeholders* são responsáveis por produzir uma configuração final que reflita os requisitos recebidos sobre o produto requerido. A cada *stakeholder* só é permitido realizar decisões sobre *features* pertencentes ao conjunto de decisão associados a seu papel. No entanto, todos eles podem visualizar a configuração completa do modelo de *features*. Como discutido em [35], esse desenho permite que os *stakeholders* se concentrem em suas decisões, evitando confusão e dispersão. Além disso, a atribuição de papéis permite maior controle sobre as *features* que cada ator pode configurar, restringindo a sua atuação a apenas aquelas que eles têm conhecimento e capacidade de decisão.

Nessa fase é utilizado um conceito bastante importante: os *assistentes pessoais de configuração*. Um assistente pessoal é um agente autônomo e inteligente que auxilia individualmente cada *stakeholder* durante a atividade de configuração. Entre suas responsabilidades estão a validação de decisões,

auxílio na atividade de recuperação da configuração em caso de conflitos, integração de decisões e, por conseguinte, garantia de consistência da configuração do produto. Para realizar tais tarefas, o assistente pessoal acompanha todas as decisões dos *stakeholders*, e verifica se elas estão produzindo uma configuração parcial válida ou inválida. Quando uma inconsistência é encontrada após uma decisão, ele é responsável por sugerir a seu *stakeholder* um conjunto de ações que possam ser seguidas, a fim de que ele/ela produza uma nova configuração válida. Caso contrário, o assistente pessoal comunica as decisões realizadas para os outros agentes, mantendo assim todas as cópias do modelo de *features* distribuído no mesmo estado de configuração e assegurando a coerência global da configuração. Detalhes de como é realizada a integração das decisões são apresentadas na Seção 3.4 e de como é feita a sugestão para correção da configuração na Seção 3.6.

Com base nesse desenho, a atividade de configuração do produto procede até que todos os *stakeholders* cheguem a um consenso e a configuração do modelo de *features* respeite todas as suas restrições. Esta atividade não precisa ser virtualmente organizada como um fluxo de trabalho dividido em uma série de passos, onde cada *stakeholder* é designado a tomar decisões de forma pré-definida, como proposto em [6, 12, 11]. Em vez disso, o resultado é uma atividade de configuração dinâmica e interativa.

A última fase se inicia após o término da atividade de configuração de todos os *stakeholders*. Nela ocorre a verificação se o produto final reflete os requisitos do cliente. A aceitação final é deferida pelo gerente de produto. Esse trabalho não aborda a validação de uma configuração em relação aos requisitos impostos. Nesse ponto pode haver uma validação em alto nível, de maneira visual, ou utilizando técnicas da teoria de lógica.

Além disso, o gerente de produto pode ficar a cargo de resolver conflitos que não puderam ser resolvidos pelos *stakeholders*. Por exemplo, se existir uma restrição que impeça que duas *features*, pertencentes a dois *stakeholders* diferentes, estejam presentes no produto final e eles não conseguirem chegar a um acordo de qual será removida, é do gerente de produto a responsabilidade de tomar a decisão final.

3.2 Um exemplo de execução

Para demonstrar as ideias acima, vamos utilizar um exemplo para facilitar o entendimento da abordagem. Utilizaremos um pequeno subconjunto do modelo de *features* apresentado na Seção 2.1.1 e representado pela Figura 3.2. Esse modelo de *features* reduzido possui onze *features* e quatro restrições –

três restrições estruturais do modelo de *features* (*File System*, *General Setup* e *Power Management Options*) e outra representada pela seta pontilhada.

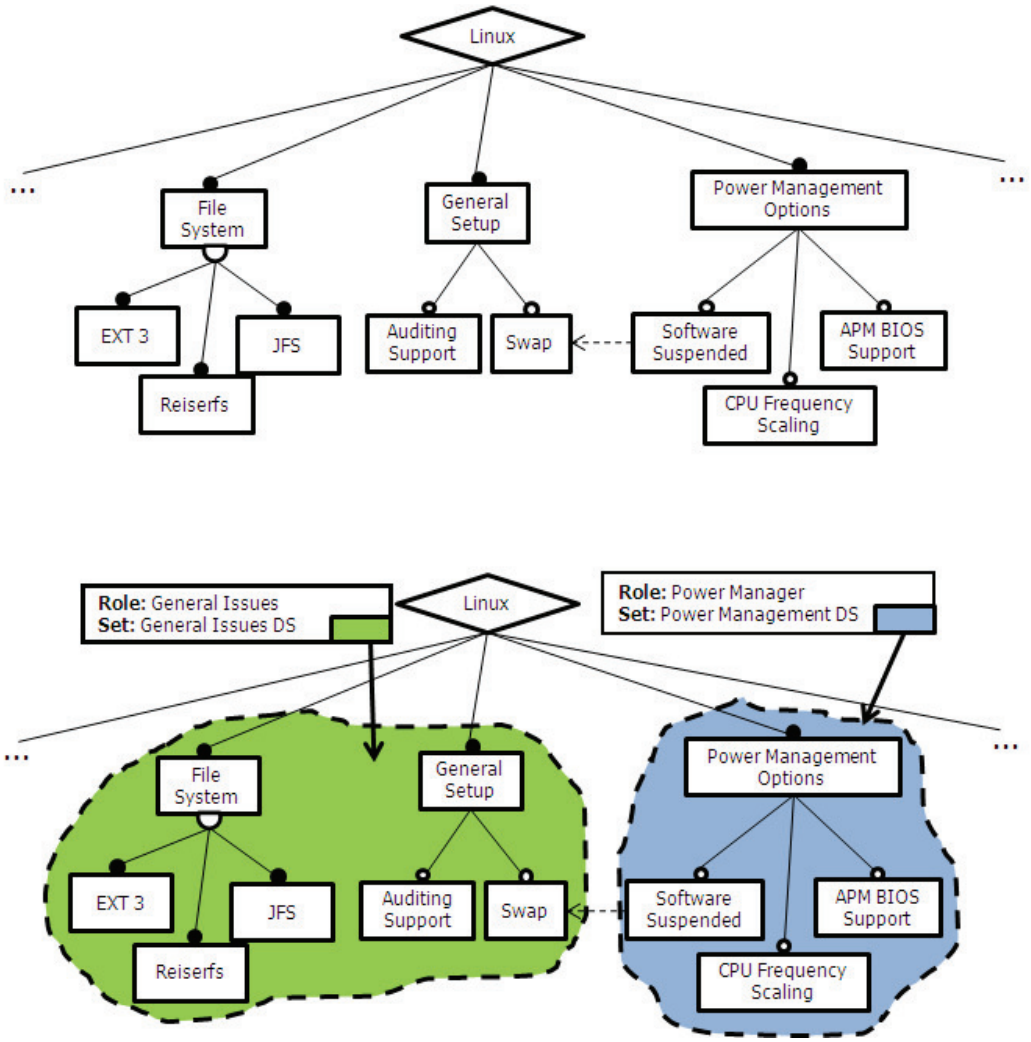

Figura 3.2: Modelo de *features* reduzido do kernel do Linux.

Na primeira fase da abordagem, o *stakeholder* gerente de produto é responsável por analisar o modelo de *features*. Dessa forma, por conhecimento do domínio, ele decide dividir esse modelo de *features* em dois conjuntos de decisão: *General Issues DS* e *Power Manager DS*. Associados a esses conjuntos foram definidos dois papéis: *General Issues Manager* e *Power Manager*, respectivamente. Após esses passos, o gerente deve indicar os *stakeholders* que participarão da atividade de configuração, atribuindo esses dois papéis, um para cada. Assim, o processo de configuração pode ser iniciado.

Na segunda fase, os *stakeholders* realizaram configurações de suas *features* na ordem em que desejarem. Importante relembrar que a um *stakeholder* só é permitido realizar decisões sobre *features* pertencentes ao conjunto de decisão associado a seu papel. No entanto, como o modelo de *features* possui restrições,

podem ocorrer situações que produzam conflitos. Distinguimos conflitos em dois tipos.

Conflitos dentro do conjunto de decisão. Suponha que o *stakeholder* interpretando o papel *General Issues Manager* selecione a *feature* "*File System*". No momento em que é realizada essa decisão, seu assistente pessoal aplica uma validação sobre a configuração do modelo de *features*, considerando as restrições. Como "*File System*"é uma *feature* alternativa e ela exige a seleção de pelo menos uma de suas *features*-filha ("*EXT3*", "*Reiserfs*"ou "*JSF*"), o assistente pessoal percebe que a configuração se tornou inválida. Neste caso, ele notifica o *stakeholder* sobre a inconsistência e sugere um conjunto de planos de recuperação que podem tornar a configuração do modelo de *features* válida novamente. A parte *A* da Figura 3.3 mostra os quatro diferentes planos de recuperação oferecidos pelo assistente pessoal que podem ser usados para corrigir a configuração do modelo de *features* nessa situação.

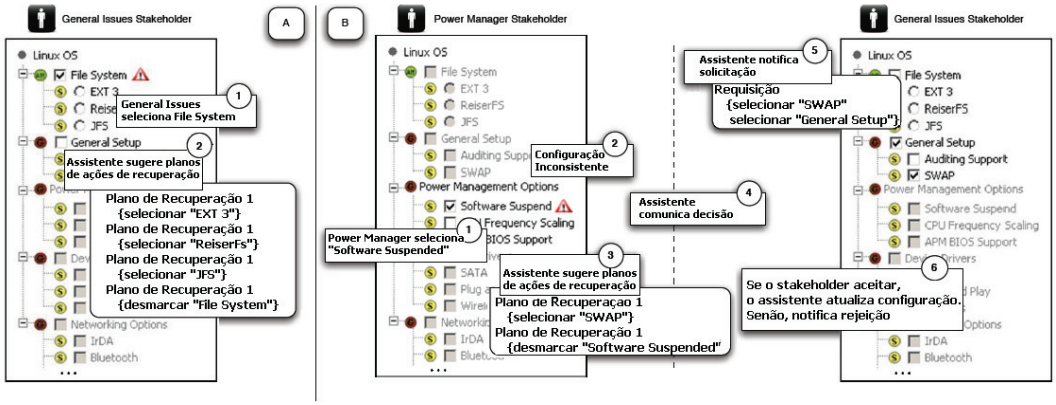

Figura 3.3: Exemplo.

Conflitos envolvendo *features* de dois ou mais conjuntos de decisão. Para ilustrar como os assistentes pessoais mantêm a coerência das decisões distribuídas, vamos tomar o exemplo mostrado pela parte *B* da Figura 3.3. Suponha a seleção da *feature* "*Software Suspended*". Note que essa decisão provoca um conflito porque uma restrição do modelo de *features* impõe que, se a *feature* "*Software Suspended*"estiver selecionada, "*Swap*"também deve estar. Portanto, quando o *stakeholder* interpretando o papel *Power Manager* seleciona essa *feature*, seu assistente pessoal percebe o conflito gerado e automaticamente o envia uma notificação. Além disso, dois planos são sugeridos: (i) uma recomendação a seleção da *feature* "*Swap*"e (ii) sugestão de desfazer a decisão recém-realizada. Se um *stakeholder* escolher um plano de recuperação que envolve a ação do outro *stakeholder*, seu assistente pessoal entra em contado com o respectivo assistente, enviando a solicitação. Esse segundo assistente se encarrega de solicitar ao seu *stakeholder* a realização

da decisão. Caso ele aceite, a configuração é atualizada e o conflito deixa de existir. Caso contrário, o *Power Manager* é notificado sobre a rejeição.

Note que consideramos *features* desmarcadas como decisões *dispensáveis*. Portanto, os *stakeholders* só precisam especificar as *features* que eles querem que façam parte da configuração, ao invés de especificar para cada *feature* se a querem ou não no produto final. Outro ponto é que o assistente pessoal se concentra na sugestão de soluções alternativas à decisão atual. Essa abordagem evita a produção de grandes quantidades de planos de recuperação que poderia fazer com que os *stakeholders* simplesmente os ignorassem. Por fim, os *stakeholders* têm a opção de não utilizar os planos de recuperação sugeridos e prosseguir com a configuração manualmente.

3.3 Ambiente de Configuração

Para realizar a configuração de um produto, um *ambiente de configuração* é atribuído a cada *stakeholder*. Um ambiente de configuração é um conjunto de componentes de que o *stakeholder* necessita para realizar a configuração colaborativa de um produto. Existem dois tipos de ambientes de configuração: *o ambiente do gerente do produto*; e (ii) *os ambientes dos* stakeholders. Ambos os ambientes são compostos por uma cópia do modelo de *features*, um assistente pessoal de configuração e uma interface gráfica, que é o ponto onde cada ator interage na configuração. A Figura 3.4 retrata um exemplo de configuração de um modelo de *features* com vários ambientes de configuração.

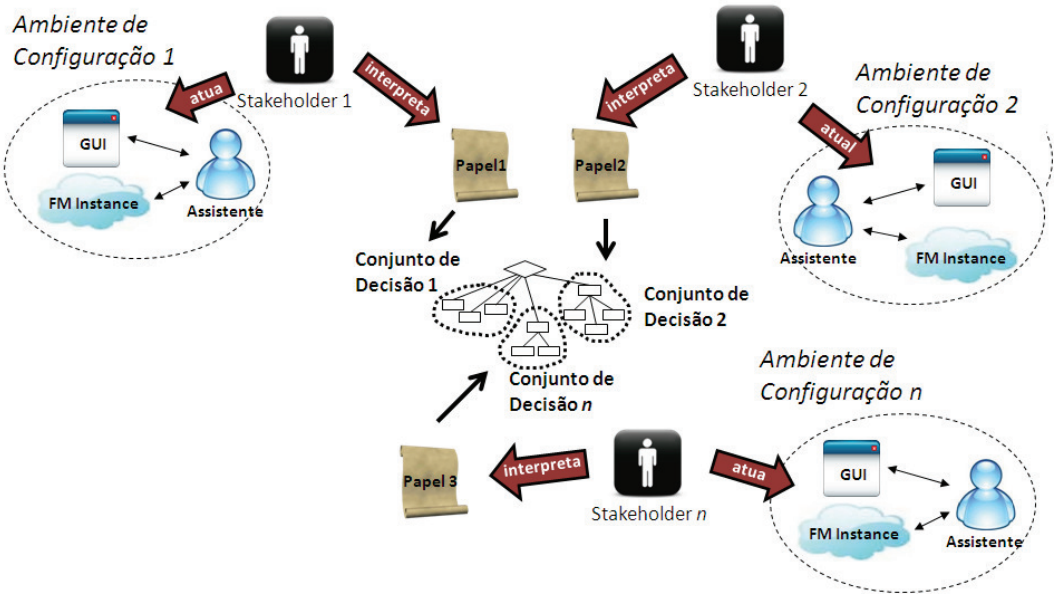

Figura 3.4: Ambientes de configuração.

O ambiente de configuração do gerente de produto é instanciado no início

do processo de configuração. Em seguida, o gerente do produto indica a seu assistente pessoal qual será o modelo de *features* a ser configurado, usando a interface gráfica. O agente, então, carrega esse modelo e inicia os demais componentes do ambiente. Após esse etapa, ele fica aguardando a solicitação de uma cópia desse modelo de *features* ou término da etapa de configuração realizada pelos demais *stakeholders*.

Ao iniciar a fase 2, os agentes pessoais são responsáveis por realizar a inicialização do seus respectivos ambientes de configuração. Para isso, cada um desses assistentes pessoais AP_i se comunica com o agente pessoal do gerente do produto AP_g, solicitando uma cópia do modelo de *features*. AP_g envia o artefato solicitado. Esse artefato possui tanto o modelo de *features* em si quanto os metadados com informações de conjuntos de decisão, papéis de decisão, etc. Com essas informações disponíveis, cada AP_i carrega o modelo de *features* no ambiente e constrói a interface gráfica. Em seguida, os assistentes ficam aguardando eventos produzidos pelo seus *stakeholders* ou a comunicação dos demais assistentes pessoais sobre eventos produzidos por outros *stakeholders*.

O ambiente de configuração é uma abstração que facilita a modularização na configuração. Em cada ambiente, o *stakeholder* só tem permissão de realizar decisões sobre um subconjunto de *features*, definido pelo papel que ele está interpretando, apesar de poder visualizar o estado de configuração das demais. A união de todos os ambientes de configuração representa espaço de configuração total, expresso pelo modelo de *features*. Os agentes que compõem cada ambiente de configuração provêem uma infraestrutura de comunicação entre os ambientes. Essa infraestrutura viabiliza a comunicação de decisões em ambientes distintos, além da permitir a negociação em caso de conflitos. A apresentação da atividade de integração de decisões nos ambientes de configuração é feita na próxima seção.

3.4 Integrando Decisões

Dada a natureza distribuída e dinâmica da nossa abordagem proposta, decisões realizadas sobre uma cópia do modelo de *features* em um dos ambientes de configuração podem provocar algum conflito com decisões realizadas em outros ambientes. Diferentes estados de configuração são consequência de decisões tomadas e que não são bem integradas. Existe um alto risco de introdução de inconsistências quando as decisões não estão bem integradas e todas as restrições não são verificadas no momento da configuração. Portanto, criar um mecanismo que garanta que decisões realizadas por qualquer *stakeholder* sejam difundidas corretamente por todos os ambientes de configuração

se torna mandatório.

A fim de manter sistematicamente todos os exemplares do modelo de *features* sincronizados ao longo da atividade de configuração, definimos uma estratégia de sincronização baseada no conceito de *configuração global*. Esse conceito é apresentado na Seção 3.4.1. Em seguida, a Seção 3.4.2 apresenta uma discussão de como a negociação de decisões produzidas nos diversos ambientes de configuração se desenvolve. Por fim, na Seção 3.4.3, é proposta uma solução para o tratamento de negociação entre *stakeholders* que possuem níveis de prioridades diferentes.

3.4.1 Configuração Global

Configuração Global (CG) é o estado de configuração que emerge das *configurações locais* de cada ambiente de configuração. A CG é uma configuração virtual, isto é, não existe um mapeamento físico específico desse conceito em nenhum ponto do processo de configuração. Essa abstração pode ser interpretada como se as atividades de configuração colaborativa, realizadas por diversos *stakeholders* em ambientes distribuídos, fosse feita por um único *stakeholder* principal – possivelmente o gerente de produto – ao longo de diversas interações pessoais com cada *stakeholder* com conhecimentos específicos. Com a configuração global é possível abstrair a característica de paralelismo da configuração colaborativa, da forma que é proposta por nossa abordagem, facilitando o desenvolvimento da metodologia.

Portanto, quando a atividade de configuração se inicia, todas as instâncias do modelo de configuração possuem a mesma configuração local, que é, portanto, a *configuração global inicial*, denotada por CG_0. Quando a atividade termina, todas as instâncias devem estar também com o mesmo estado de configuração local, que é a *configuração global final*, denotada por CG_f. Durante o processo de configuração, entre CG_0 e CG_f, podem existir várias configurações intermediárias. Dizemos que a configuração global evolui quando os *stakeholders* realizam um conjunto de decisões que leva a configuração GC_i, num estado válido, para uma outra configuração válida GC_{i+1}. Note que entre duas configurações globais podem existir estados de inconsistência. Essas inconsistências locais devem ser resolvidas antes de um estado evoluir para outro. O objetivo do nosso mecanismo de sincronização é manter todas as instâncias do modelo de *features* consistentes ao longo de todos os estágios intermediários da atividade de configuração do produto e garantir uma configuração global GC_f correta.

Na Figura 3.5 podemos visualizar como funciona a evolução do estado

de configuração global. O exemplo possui três ambientes de configuração e a configuração global está, inicialmente, em um estado válido qualquer GC_i. Como o processo de configuração se dá dinamicamente, qualquer um dos *stakeholders* atuando sobre cada um dos ambientes de configuração pode realizar alguma decisão. Essa decisão deve ser difundida por todos os demais ambientes. No entanto, como o modelo de *features* pode possuir restrições sobre os estados das *features*, essa decisão pode provocar um conflito dentro ou entre vários ambientes de configuração. No caso de um conflito, algumas outras decisões devem ser tomadas para sua resolução, além das demais decisões de configuração. No momento em que todos os conflitos são resolvidos a configuração global do modelo de *features* se torna válida. Dizemos então que a configuração evoluiu do estado GC_i para o estado GC_{i+1}.

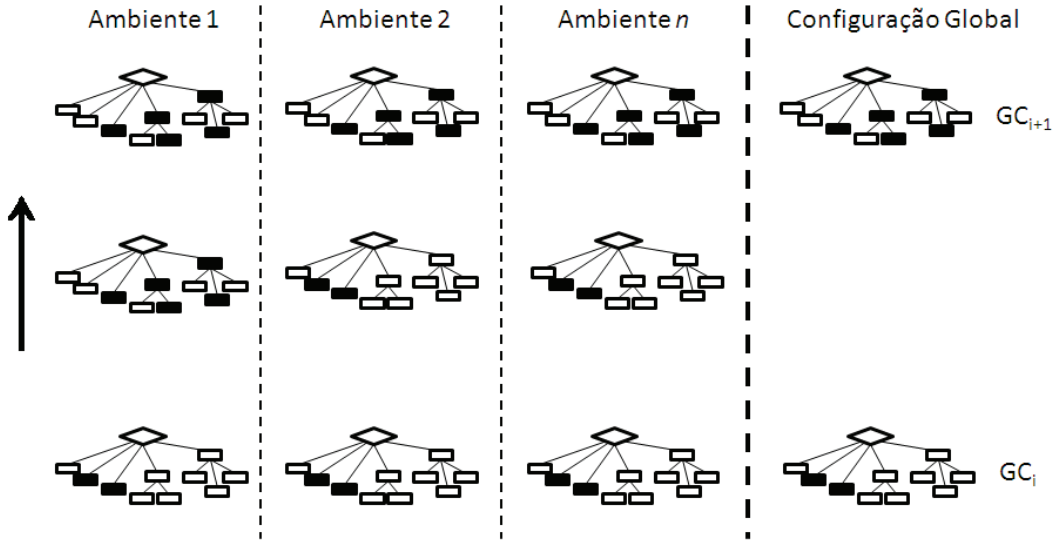

Figura 3.5: Evolução da configuração global de CG_i para CG_{i+1}.

3.4.2 Negociação de Decisões

Um dos *stakeholders* pode realizar decisões de configuração em qualquer momento dentro da Fase 2 do processo de configuração (ver Figura 3.1). No entanto, com esse relaxamento, é possível que surjam conflitos de decisões, isto é, é possível que seja produzido algum estado de configuração que desrespeite alguma restrição do modelo de *features*. Dessa maneira, a abordagem deve oferecer algum tipo de mecanismo que suporte a correção dessa configuração.

O comportamento da nossa estratégia se divide em três casos:

1. **A decisão do *stakeholder* resulta em uma configuração válida do modelo de *features* e não afeta outros conjuntos de decisão**.

Neste caso, o assistente envia uma notificação para os outros, que por sua vez atualizam as cópias de seus respectivos modelos de *features* locais.

2. **A decisão resulta em uma configuração inválida do modelo de *features*, porém não tem um impacto sobre outros conjuntos de decisão**. Nesse caso, o assistente raciocina sobre a configuração inválida e produz alguns planos de recuperação para a configuração tornar-se válida. O *stakeholder*, então, escolhe o plano que melhor atenda os seus requisitos. Em seguida, o assistente pessoal imediatamente envia uma notificação do novo estado de configuração de suas *features* para os demais, que por sua vez atualizam suas respectivas instâncias do modelo de *features*.

3. **A decisão tem dependência direta de inclusão ou exclusão de decisões em outros conjuntos de decisão**. Neste caso, todas as cópias do modelo de *features* estarão em um estado inconsistente. O assistente de configuração pessoal, responsável pelo *stakeholder* que provocou a decisão e denotado por ap_1, se comunica com os demais assistentes envolvidos para propor planos de ações de correção da configuração. Esses planos são apresentados a todos os *stakeholders*, que podem aceitar um ou rejeitar todos. Nesse ponto surge a necessidade de suporte à negociação de decisões.

Para os dois primeiros casos, após o tratamento das implicações da decisão, se nenhum outro conflito não resolvido está presente na configuração, a configuração global evolui. No terceiro caso, a evolução ocorre caso não haja mais conflitos e todos os *stakeholders* aceitem a proposta de ap_1.

Além de garantir a consistência da configuração através de sugestões de correção da configuração, nossa abordagem provê suporte a negociação de decisões. Negociação em um ambiente colaborativo, onde cada *stakeholder* pode ter perspectivas e expectativas diferentes ou, às vezes, divergentes, é um assunto bastante importante de ser tratado. A negociação é uma forma de lidar com restrições que por algum motivo podem ter sido ignoradas em fase de construção do modelo de *features* ou que só são aplicáveis no domínio onde será inserido o produto a ser gerado. Chamamos essas restrições de *restrições indiretas*. Infelizmente, as abordagens apresentadas na Seção 2.2 não provêem suporte à negociação de decisões.

A negociação é possível em nossa abordagem porque em nenhum momento da fase de configuração um *stakeholder* é restringido de realizar decisões na configuração como consequência de decisões tomadas por outros *stakeholders*. O cenário que exige negociação surge quando ocorre um conflito, geral-

mente envolvendo mais de um conjunto de decisão. Nesse caso, pelo menos um dos *stakeholders* deve abrir mão do estado de configuração corrente que o seu conjunto de decisão possui, desfazendo algumas decisões ou realizando outras, para que seja possível produzir uma configuração válida. Dependendo da definição das restrições do modelo de *features*, apenas uma decisão simples (selecionar ou excluir uma feature) é suficiente para resolver o conflito. No entanto, podem surgir situações para as quais a solução é mais complexa, exigindo ação de mais de um *stakeholder*. A complexidade do raciocínio sobre as restrições do modelo de *features* e o estado de configuração distribuído é abstraída dos *stakeholders* pelos assistentes pessoais, conforme será discutido na Seção 3.6.

Dessa forma, para visualizarmos como funciona a negociação de decisões, vamos supor um estado de configuração válido qualquer C_1, resultado de diversas decisões colaborativas realizadas previamente. Assim, para atender os requisitos impostos, um *stakeholder* realiza uma decisão d sobre C_1, produzindo a configuração C_2. No entanto, devido às restrições do modelo de *features*, vamos supor que d tenha provocado um conflito. Portanto, C_2 é uma configuração inválida. Assumindo que o *stakeholder* realmente deseje que d faça parte da configuração final, ele, como *gerador* do conflito, deve negociar com os demais *stakeholders* envolvidos um conjunto de decisões corretivas H^d que, quando aplicadas sobre C_2, produzam uma terceira configuração válida C_3. Portanto, assim que o assistente pessoal do *stakeholder* gerador percebe esse conflito em C_2, ele imediatamente envia uma notificação, além de sugerir um conjunto de sugestões HS^d de como resolvê-lo. Dessa forma, o *stakeholder* deve escolher um conjunto de decisões corretivas $H_i^d \in HS^d$ e enviá-lo aos *stakeholders* envolvidos para decidirem se aceitam ou não realizar essas decisões. Caso eles aceitem a solicitação, todas os ambientes de configuração são atualizados e uma nova configuração global, igual a C_3, é produzida. No entanto, note que, como em toda negociação, a solicitação pode ser rejeitada. Uma opção para o *stakeholder* gerador do conflito nessa situação é escolher outro conjunto de decisões corretivas $H_j^d \in HS^d$ que atenda os seus requisitos.

Na apresentação acima, assumimos que todos os *stakeholders* configuradores envolvidos no processo de configuração possuem o mesmo grau de prioridade. No entanto, existem situações em que alguns *stakeholders* possuem poder de decisão maior do que outros. Na próxima seção discutimos como é tratado nesse trabalho a questão da prioridades entre *stakeholders*.

3.4.3 Priorização de stakeholders

Priorização de *stakeholders* no processo de configuração é um assunto não abordado por nenhum dos trabalhos apresentados na Seção 2.2. No entanto, podem existir cenários em que alguns *stakeholders* possuem um nível de prioridade maior em relação a outros. Essas prioridades podem ser baseadas em questões técnicas ou até mesmo não-técnicas. Como um exemplo de cenário onde questões técnicas definem *stakeholders* com prioridade maior que outros, vamos utilizar o exemplo da família de sistemas do kernel do Linux. Supondo que o produto a ser configurado (um sistema operacional) será utilizado para controlar dispositivos móveis, é razoável aceitar que o *stakeholder* responsável por tomar decisões relativas ao gerenciamento de energia tenha prioridade sobre os demais *stakeholders*, considerando que esse assunto é crítico nesse domínio. Por outro lado, em determinadas situações, questões comerciais passam a ter importância maior do que as questões técnicas. Dessa forma, definir prioridade maior a um *stakeholder* hierarquicamente mais importante pode tornar o processo mais eficiente.

Os trabalhos estudados na Seção 2.2 não apresentam solução para definição de prioridades entre *stakeholders*. Na verdade, quando analisamos como o processo organizado por fluxo de atividades funciona, podemos notar que definições implícitas de prioridade de alguns *stakeholders* sobre outros são produzidas como efeito colateral desse tipo de coordenação. Conforme as três abordagens propõem, o modelo de *features* é especializado durante cada passo do fluxo de atividades. Dessa forma, decisões realizadas nos passos anteriores podem impedir que outras decisões sejam tomadas em passos posteriores, caracterizando prioridade maior de alguns *stakeholders*. No entanto, esse comportamento pode não ser o desejado num determinado cenário de configuração.

Propomos nesse trabalho o conceito de *prioridade entre* stakeholders. Conforme vimos na Seção 3.4.2, quando um *stakeholder* produz uma decisão que provoca um conflito, esse *stakeholder* deve negociar alterações na configuração com os demais *stakeholders* envolvidos para que sua decisão possa ser inserida. Essa responsabilidade de negociação, por parte do gerador do conflito, se dá porque não há definição de prioridade entre eles. Portanto, deve haver um acordo mútuo.

No entanto, em um cenário com dois *stakeholders*, s_1 e s_2, em que s_1 tem prioridade sobre s_2, quando s_1 produz uma decisão que provoca um conflito com decisões de s_2, a responsabilidade de resolução é atribuída a s_2. Isso ocorre pois, por definição de prioridade, a decisão de s_1 possui uma importância maior

no domínio de que está sendo tratado. Importante ressaltar que prioridade não significa ausência de negociação. O que acontece de fato é transferência da responsabilidade de iniciar a negociação para os *stakeholders* com menor prioridade, fazendo com que o *stakeholder* prioritário tenha o poder de decidir aceitar ou não as possíveis sugestões. A responsabilidade de definição dos níveis de prioridade de cada *stakeholder* é atribuída ao gerente do produto.

Dependendo do cenário de configuração, podem existir *stakeholders* que são mais prioritários que uns e menos prioritários que outros. Dessa forma, o conceito de *nível de prioridade* se faz importante. Quando o gerente do produto está na fase de definição dos papéis dos *stakeholders*, ele pode definir qual o nível de prioridade que um papel terá em relação aos demais. O gerente do produto pode definir quantos níveis de prioridade ele desejar.

Assim, durante a configuração, quando um *stakeholder* s_1 realiza uma decisão que produz um conflito, seu assistente pessoal verifica seu nível de prioridade e o nível dos demais envolvidos. Para aqueles *stakeholders* que são mais prioritários, ou tenha o mesmo nível de prioridade, s_1 deve propor uma conjunto de ações corretivas – sugeridas pelo seu assistente pessoal. Por outro lado, os *stakeholders* menos prioritários devem se mobilizar para resolver os conflitos nas restrições que suas *features* estão envolvidas. Por exemplo, suponha uma configuração com três *stakeholders* s_1, s_2 e s_3, com prioridades p_1, p_2 e p_3, respectivamente, onde $p_1 > p_2 > p_3$. Se s_2 realizar uma decisão d que produza um conflito, seu *stakeholder* irá sugerir conjuntos soluções para resolver esse conflito. Suponha que s_2 escolha o conjunto de decisões de correção $H^d = H_1^d \cup H_2^d \cup H_3^d$, onde H_1^d é o conjunto de decisões de correção que envolvem *features* do conjunto de decisão de s_1. De maneira análoga, H_2^d possui decisões sobre *features* sob responsabilidade de s_2 e H_3^d de s_3. Portanto, como s_1 possui prioridade maior do que s_2, s_2 deve negociar com s_1 para que ele execute as decisões H_1^d. Por outro lado, como s_2 tem prioridade maior que s_3, s_3 deve executar as decisões H_3^d ou negociar com s_2 outra solução.

3.5 Abordagem Dinâmica vs. Coordenação por Fluxo de Atividades

A configuração coordenada por fluxos de atividades é uma boa solução para cenários de configuração que são, por construção, sequenciais. A Seção 2.2.1 apresenta o exemplo de cadeia de fornecimento de software, onde configuração em estágios é aplicável. Alguns cenários um pouco mais complexos, porém que ainda têm características de fluxo de atividades, são bem tratados por esse tipo de coordenação. No entanto, nos cenários onde os requisitos não são bem definidos e exceções ao fluxo podem ocorrer, essas abordagens

podem falhar. Vamos analisar o seguinte cenário. Suponha que, no exemplo apresentado na Seção 3.2, num estágio avançado da atividade de configuração, a manutenção da *feature* "*Swap*"no produto final tenha se tornado impraticável devido a ela requerer uma quantidade extra de memória, que encareceria o dispositivo móvel que está sendo construído. Esse é um exemplo de uma restrição indireta, citada na Seção 3.4.2. Uma implicação direta dessa alteração na configuração é que o *stakeholder* interpretando o papel *Power Manager* precisa desmarcar a *feature* "*Software Suspended*". Nesse caso, se a configuração estiver sendo coordenada por um fluxo de atividades previamente definido, a reconfiguração pode ficar travada, devido à possível inexistência de um fluxo que trate essa situação específica. Nesse exemplo didático, a solução é trivial. No entanto, no caso de cenários reais a reconfiguração pode se tornar uma atividade bem complexa, inserindo um alto risco introdução de inconsistências na configuração.

Por outro lado, esse cenário é facilmente tratado por nossa abordagem. Neste caso, assim quando a *feature* "*Swap*"é desmarcada pelo "*General Issues Manager*", seu assistente pessoal automaticamente notifica o conflito gerado, já propondo soluções de correção. Nesse caso específico é apresentada uma opção de desmarcar a *feature* "*Software Suspended*". Porém, como essa *feature* não pertence ao seu conjunto de decisão, automaticamente seu assistente entra em contato com o assistente do *stakeholder* "*Power Manager*", que é notificado da necessidade de alteração. Quando a negociação é encerrada, a configuração volta a um estado válido, garantindo a coerência global.

3.6 Recuperação de Estados Inconsistentes

Estados inconsistentes são resultado de decisões sobre a configuração do modelo de *features* que desrespeitam, pelo menos, uma de suas restrições. Um estado inconsistente da configuração impede a derivação de um produto a partir de uma LPS. Conforme vimos na Seção 3.4.1, entre duas configurações globais subsequentes, CG_i e CG_{i+1}, que por definição são configurações válidas, podem ser produzidos estados de inconsistência. Portanto, nesse ponto, podemos interpretar que o objetivo principal dos assistentes pessoais no processo de configuração é auxiliar os *stakeholders* a evoluir uma configuração GC_i para GC_{i+1}, sugerindo correções no caso de estados intermediários de inconsistência, até que seja produzida a configuração final. Importante lembrar que a responsabilidade de produzir uma configuração final que retrate os requisitos impostos sobre o produto é do *stakeholder*. Os assistentes pessoais têm como finalidade garantir a produção de configurações válidas, sugerindo ações corretivas caso

algum estado de inconsistência seja alcançado. Nesta seção, apresentamos os algoritmos incorporados pelos assistentes pessoais para validação de uma configuração e para geração automática de planos de ações de recuperação, que podem ser usados pelos *stakeholders* nesse cenário. Um *plano de ações de recuperação* é uma sequência de decisões que corrige a configuração do modelo de *features*. Chamamos essas decisões de *ações de correção*. Mais de um plano de ações de correção pode ser utilizado para atingir o mesmo objetivo: *uma configuração válida*. Cabe ao *stakeholder* decidir qual delas produzirá uma configuração que reflita os seus requisitos.

Na Seção 3.6.2 discutimos como é realizada a validação de uma configuração e na Seção 3.6.3 é apresentado o algoritmo de criação dos planos de ações de recuperação. Na próxima seção apresentamos uma estrutura de dados que é utilizada como base de nossos algoritmos: o *grafo de dependência*.

3.6.1 Grafo de Dependências

Durante o processo de configuração de um modelo de *features*, uma decisão sobre uma *feature* pode afetar outras, considerando a existência de restrições. Classificamos as restrições em dois tipos: (i) *restrições estruturais*; e (ii) *restrições de dependência*. Restrições estruturais são aquelas que limitam os possíveis estados de configuração de um grupo de *features* que possuem relações hierárquicas. Por exemplo, quando uma *or-feature* está selecionada, pelo menos uma de suas *features*-filha também deve estar. Por outro lado, se essa *feature* não estiver selecionada, nenhuma de suas filhas pode estar. As restrições estruturais são intrínsecas da estrutura do modelo de *features*. Já as restrições de dependência são aquelas definidas através de expressões booleanas. Essas restrições são escritas no padrão: $expr(F_1) \rightarrow expr(F_2)$, onde *expr* é uma expressão booleana formada pela combinação dos operadores *and*, *or*, *xor* ou *not*, sobre *features* $f_i \in F \subset FM$. Restrições de estados de configuração de um grupo de *features* que não podem ser representadas por restrições estruturais são especificadas através de restrições de dependência.

Portanto, para facilitar a verificação de dependências que *features* possuem em relação a outras, definimos uma estrutura de dados baseada em grafo. Referenciamos esse grafo como *grafo de dependências*. O grafo de dependências é uma abstração única para representar os dois tipos de restrições. A definição 3.1 formaliza o conceito.

Definição 3.1 (Grafo de Dependências) *Grafo de dependências é um grafo não-direcionado* $G(V, E)$*, onde cada vértice* $v_i \in V$ *representa uma* fea-

ture $f_i \in FM$ *e cada aresta* $e_{ij} \in E$ *representa uma relação de dependência* $R(f_i, f_j)$ *entre* f_i *e* f_j*, onde* $f_j \in F \subset M$*.*

Para construir um grafo de dependências relativo a um modelo de *features* $FM(F, C)$, onde F é o conjunto de *features* do modelo e C é o conjunto de restrições, utilizamos no Algoritmo 1.

Algorithm 1 Algoritmo de construção de um Grafo de Dependências

Entrada: Um Modelo de *features* $FM(F, C)$
Saída: Grafo de dependências $g(V, E)$
{Adiciona cada *feature* do modelo como vértice do grafo}
for $f \in F$ **do**
  $v \leftarrow createVertex(f)$
  $g.add(v)$
end for
{Adiciona as relações de dependência entre *feature* como arestas do grafo}
for $c \in C$ **do**
  {Tratamento para restrições estruturais}
  if $isStructuralConstraint(c)$ **then**
    $f_p \leftarrow getParentFeature(c)$
    $F_c \leftarrow getChildren(c)$
    for $f_c \in F_c$ **do**
      $e \leftarrow createEdge(f_p, f_c, c)$
      $g.add(e)$
    end for
    {Tratamento para restrições de dependência}
  else
    if $isDependencyConstraint(c)$ **then**
      $expr1 \leftarrow getExpr1(c)$
      $expr2 \leftarrow getExpr2(c)$
      for $f_i \in getfeatures(expr1)$ **do**
        for $f_j \in getfeatures(expr2)$ **do**
          $e \leftarrow createEdge(f_i, f_j, c)$
          $g.add(e)$
        end for
      end for
    end if
  end if
end for

Como pode ser observado no algoritmo, o grafo vai sendo montado unindo, através de arestas, as *features* que têm relação direta de dependência. No caso de restrições estruturais, cada feature-pai é ligada a todas suas *features*-filha. Para cada restrição de dependência são construídas arestas que ligam todas as *features* de $expr1$ a cada *feature* de $expr2$. Note que em cada aresta é adicionada a restrição que relaciona essas *features*. Essa informação

será útil nos algoritmos que mostraremos a seguir. A Figura 3.6 mostra como seria o grafo de dependências do modelo de *features* do exemplo da Seção 3.2.

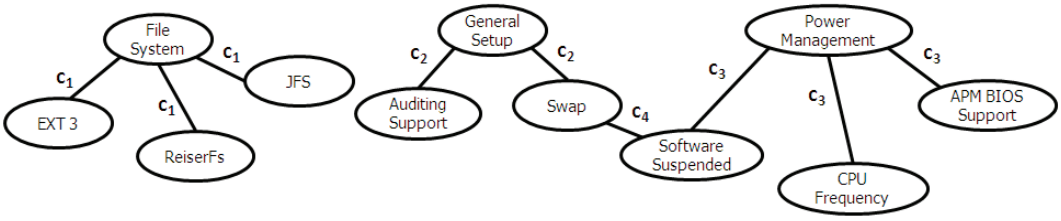

Figura 3.6: Grafo de dependências.

3.6.2 Validando Decisões

Quando uma decisão é tomada por um *stakeholder*, seu assistente pessoal deve verificar se essa decisão provocou alguma inconsistência. Para validar uma decisão, é necessário verificar se o estado da nova configuração desrespeita alguma das restrições do modelo de *features*. Assim, uma solução possível seria averiguar se cada restrição do modelo de *features* está sendo respeitada. O grafo de dependências, conforme construído na Seção 3.6.1, nos permite realizar a verificação de forma mais eficiente. Dessa forma, como as *features* tangentes à *feature* f, no grafo de dependência, por construção, são as *features* que podem ser afetadas com a mudança de estado de f, só é necessária a avaliação dessas restrições que as relacionam. Portanto, assim que o assistente percebe uma decisão, ele, primeiramente, obtém do grafo de dependências as arestas que estão ligadas à *feature* associada a essa decisão. Dessas arestas são extraídas as restrições que relacionam essas *features*. Por fim, o assistente verifica apenas esse subconjunto de restrições para descobrir se todas elas estão sendo respeitadas. Se a configuração estiver válida o assistente simplesmente envia aos demais assistentes o novo estado. Caso contrário, sua responsabilidade é produzir uma plano de ações de recuperação que auxilie o *stakeholder* a alcançar esse estado desejado.

3.6.3 Produzindo Planos de Ações de Recuperação

Nós abordamos o problema de produzir planos de ações de recuperação como um Problema de Satisfação de Restrições, do inglês *Constraint Satisfaction Problem* (CSP). Tsang [36] define CSP como:

Definição 3.2 (CSP) *O Problema de Satisfação de Restrições é uma tupla* (Z, D, C)*, onde* $Z = \{z_1, ..., z_n\}$ *é um conjunto finito de variáveis;* $D =$

$\{d_1, ..., d_n\}$ o conjunto domínio dessas variáveis; e $C = \{c_1, ..., c_n\}$ um conjunto finito de restrições sobre um subconjunto arbitrário das variáveis de Z.

Assim, a tarefa em um CSP é atribuir valores $d \in D$ a cada variável $z \in Z$ que simultaneamente satisfaçam todas as restrições $c \in C$.

Nossa abordagem utiliza as semelhanças entre um CSP e o problema de encontrar configurações válidas de um modelo de *features* para oferecer soluções de correção de estados inconsistentes. Assim como um CSP, um modelo de *features* consiste de um conjunto finito de variáveis (suas *features*) com um domínio finito bem definido (domínio com dois estados, isto é, selecionado ou desmarcado) e possui um conjunto de restrições sobre os estados dessas variáveis. Utilizamos o mapeamento de um modelo de *features* para um CSP conforme apresentado em [37]. Portanto, podemos encarar uma configuração válida de um modelo de *features* como uma solução γ_i de um CSP equivalente, onde cada variável v_{f_i} de γ_i tem seu valor definido como $v_{f_i} = 1$ quando a *feature* f_i associada está selecionada na configuração e, analogamente, v_{f_i} tem seu valor definido como $v_{f_i} = 0$ quando f_i está desmarcada.

No entanto, infelizmente, encontrar soluções para um CSP é um problema NP-Completo no pior caso [36]. Assim, para modelos de *features* relativamente grandes, utilizar essa abordagem de forma direta para calcular todas as soluções seria impraticável. Além disso, seriam produzidas grandes quantidades de soluções, tornando mais custoso para o *stakeholder* encontrar uma solução nesse conjunto do que raciocinar ele próprio sobre as restrições. Nesse contexto, o grafo de dependência vem nos auxiliar. Ao invés de construir um CSP a partir de um modelo de *features* completo, é criado um modelo de programação de restrições para encontrar soluções de configurações relacionadas apenas à decisão que queremos validar. Com o grafo de dependências podemos obter facilmente essas informações.

O primeiro passo do algoritmo é construir o modelo de programação de restrições. Assim, assumindo que uma decisão foi realizada sobre uma *feature* f_x, nosso CSP engloba: (i) um conjunto de variáveis F, representando *features* $f_i \in FR$, onde FR é o conjunto de *features* do modelo de *features* que possuem alguma relação de dependência com f_x, e (ii) um conjunto de restrições C, que são as restrições que envolvem f_x. O domínio de cada variável $v_f \in F$ é definida como booleana (ou seja, falso ou verdadeiro). Note que as *features* podem ser mandatórias ou opcionais. Sendo assim, quando o CSP é construído, as variáveis v_f cujas *features* são mandatórias são atribuídas com valores constantes $v_f = 1$.

A segunda etapa obtém as soluções desse CSP e as transforma em uma série de planos de ações de recuperação (RC). Para calcular o conjunto

de *features* que devem ser selecionadas e o conjunto de *features* que devem ser desmarcadas, cada solução γ_i do CSP é comparada a uma representação equivalente da configuração inválida corrente, denominada γ_w.

Primeiro, agrupamos as *features* em dois conjuntos distintos:

– sel_w engloba as *features* $f \in FR$ cujas respectivas variáveis v_f têm o valor definido como verdadeiro na configuração inválida atual γ_w.

$$sel_w = \{f | f \in FR \wedge v_f = 1 \wedge v_f \in \gamma_w \wedge sel_w \subseteq FR\}$$

– e para cada solução do CSP encontrada γ_i, o conjunto sel_i agrupa as *features* $f \in FR$ cujas respectivas variáveis v_f tem o valor de domínio como verdadeira.

$$sel_i = \{f | f \in FR \wedge v_f = 1 \wedge v_f \in \gamma_i \wedge sel_i \subseteq FR\}$$

Portanto, o conjunto de *features* de cada plano de ações de recuperação que devem ser selecionadas, denotado por sel_r, é produzido a partir da relação complementar de sel_i em sel_w.

$$sel_r = sel_i \backslash sel_w$$

Da mesma forma, o conjunto de *features* de cada plano de ações de recuperação que devem ser desmarcadas, denotado por $desel_r$, é produzido a partir da relação complementar de sel_w em sel_i.

$$desel_r = sel_w \backslash sel_i$$

Por fim, o assistente pessoal retorna os planos de ações de recuperação aos *stakeholders*, que estarão encarregados de escolher qual deles melhor se adapta as seus requisitos.

Para visualizarmos na prática como funciona esse algoritmo, vamos tomar o exemplo da Seção 3.2 novamente. Supondo que a configuração não tenha sido iniciada e que a *feature* "*Software Suspended*"(SS) tenha sido selecionada. Portanto, o primeiro passo do assistente pessoal é recuperar as restrições que estão associadas a essa *feature*. Verificando a grafo de dependências expresso na Figura 3.6, obtemos as seguintes *features* diretamente relacionadas: "*General Setup*"(GS) e "*Swap*"(SW). Além disso, duas restrições estão associadas a essas *features*: (i) a restrição que exige que a *feature* "*Swap*"seja selecionada; e (ii) a restrição estrutural de "*General Setup*".

Assim, portanto, as restrições $(SS \rightarrow SW)$ e $(SS \rightarrow GS)$ podemos ser representadas por:

$$(\neg SS \vee SW) \wedge (\neg SS \vee GS)$$

Como GS é uma *feature* mandatória, temos o seguinte CSP:

$$(\neg SS \vee SW) \wedge (\neg SS \vee true) \Rightarrow (\neg SS \vee SW)$$

Dessa forma, temos as seguintes possíveis combinações válidas

$$(1)\ SS = true \text{ e } SW = true$$
$$(2)\ SS = false \text{ e } SW = true$$
$$(3)\ SS = false \text{ e } SW = false$$

Assim, obtemos o conjunto $sel_w = \{SS\}$ e os conjuntos $sel_1 = \{SS, SW\}$, $sel_2 = \{SW\}$ e $sel_3 = \{\}$. Portanto, os conjuntos de ações para cada solução são:

$$(1)\ sel_r = \{SW\} \text{ e } desel_r = \{\emptyset\}$$
$$(2)\ sel_r = \{SW\} \text{ e } desel_r = \{SS\}$$
$$(3)\ sel_r = \{\emptyset\} \text{ e } desel_r = \{SS\}$$

que serão traduzidos nos seguintes planos de ações de recuperação:

(1) Selecionar "*Swap*"
(2) Selecionar "*Swap*;"e desmarcar "*Software Suspended*"
(3) Desmarcar "*Software Suspended*"

3.6.4 Filtrando Planos de Ações

Com a utilização do grafo de dependências é possível diminuir significativamente a quantidade de sugestões que são propostas ao *stakeholder*, facilitando sua escolha. No entanto, podemos perceber no exemplo da última seção que, mesmo assim, algumas sugestões são desnecessárias. Por exemplo, o plano de ações de recuperação (2) pede para selecionar "*Swap*"e desmarcar "*Software Suspended*", que foi a *feature* que o *stakeholder* acabou de selecionar. Isso ocorre porque o CSP produz todos os estados de configuração possíveis considerando as variáveis e as restrições especificadas. Dessa forma, um filtro final sobre as sugestões se faz necessário. Utilizamos duas técnicas para filtrar o número de planos de ações de recuperação.

A primeira técnica é modificar a própria construção do CSP. Assim, quando um *stakeholder* realiza uma decisão sobre uma *feature* f, nosso CSP

considera apenas os estados possíveis com essa *feature* fixada com seu novo estado. Portanto, a variável v_f, correspondente a f, é marcada como uma constante com o valor do seu novo estado ($v_f = 1$, se f foi selecionada ou $v_f = 0$, caso contrário). Como consequência, apenas sugestões com esse novo estado são produzidas. Por fim, é adicionada a sugestão de desfazer a decisão. No exemplo da seção anterior, portanto, só serão produzidos dois planos de ações de recuperação:

(1) Selecionar "*Swap*";

(2) Desmarcar "*Software Suspended*";

O segundo tratamento ocorre para o caso de decisões sobre *features* que estão associadas a várias outras e por diversas restrições. Como um CSP é um problema de encontrar combinações de estados e esse problema é exponencial, mesmo para pequenas quantidades de variáveis, podem ser produzidos muitos estados válidos que, por consequência, produziriam muitos planos. Portanto, um filtro sobre esse conjunto se faz necessário, para produção de sugestões que são realmente relevantes aos *stakeholders*. Dessa forma, aplicamos o Algoritmo 2 de filtro sobre o conjunto de planos já produzido inicialmente.

Inicialmente identificamos as restrições impostas sobre a *feature* f, que sofreu uma decisão, e que estão sendo desrespeitadas. Dessas restrições obtemos o conjunto de *features* envolvidas. No segundo passo, removemos dos planos de ações todas as ações sobre as *features* que não estão provocando conflito. Por fim, como com a remoção de algumas ações podem surgir planos semelhantes, isto é, planos com as mesmas sugestões de ações, removemos os planos duplicados. Dessa forma, reduzimos a quantidade de planos que não seriam úteis para o *stakeholder*. Note que nenhuma modificação na construção do CSP é realizada nessa segunda abordagem. Isso se dá porque, se ignorássemos as restrições que não estão sendo desrespeitadas no momento no CSP, poderiam ser produzidas sugestões de configurações inválidas.

Para exemplificar a importância desse filtro, vamos tomar o modelo de *features* da Figura 3.7. Dessa forma, para produzir sugestões para corrigir a configuração, tendo como base uma decisão realizada, por exemplo, sobre a *feature* f_2, é necessário construir um CSP com as restrições estruturais de f_1 e f_2, e a restrição de dependência $f_2 \rightarrow f_5$, envolvendo todas as *features* do modelo. Note que diversos estados são possíveis nesse cenário e, portanto, muitos planos de ações de recuperação serão produzidos. No entanto, o plano que é realmente importante para o *stakeholder* é aquele que lhe sugere selecionar f_5. Assim, com esse algoritmo, são removidas as ações sobre as *features* f_3 e f_4, porque a restrição estrutural de f_2 está sendo respeitada, e sobre a *feature* f_1, por sua restrição estrutural também estar sendo respeitada.

Algorithm 2 Algoritmo de filtro sobre os planos de ações de recuperação

Entrada:
O conjunto de planos de ações de recuperação P
O grafo de dependências g
feature f sobre a qual foi realizada a decisão
Saída:
Um conjunto de $P_f \subset P$
{Obtém as *features* das restrições que não estão sendo respeitadas}
$F_c \leftarrow \emptyset$
$C \leftarrow g.getNeighborConstraints(f)$
for $c \in C$ **do**
  $valid \leftarrow isValid(c)$
  if $\neg valid$ **then**
    $F_c \leftarrow F_c \cup getfeatures(c)$
  end if
end for
{Remove as ações sobre *features* que não estão envolvidas em conflito}
for $p \in P$ **do**
  $A \leftarrow getRecoveryActions(p)$
  for $a \in A$ **do**
    $f_a \leftarrow getFeature(a)$
    if $\neg contains(f_a, F_c)$ **then**
      $remove(a, p)$
    end if
  end for
end for
{Remove possíveis planos duplicados devido à remoção de ações}
$removeDuplications(P)$

Como a decisão foi realizada por f_2, ela é fixada como selecionada. Então, apenas sobra a opção de selecionar f_5, ou desfazer a decisão de selecionar f_2, conforme definido pelo tratamento anterior. Assim, com esse algoritmo de filtragem, conseguimos reduzir as sugestões para apenas aquelas que são realmente relevantes ao *stakeholder* no momento.

3.7 Sugestão de Configurações Ótimas

Quando estudamos as linhas de produtos de software, observamos que inúmeros sistemas podem ser derivados através de combinação das *features*, desde que as restrições sejam respeitadas. Alguns trabalhos estudam e propõem algoritmos para determinar a quantidade de produtos que são possíveis de serem derivados de um LPS [14]. Portanto, dada essa grande quantidade de possibilidades, é natural desejar a melhor dessas configurações. Em [14], Benavides *et al.* propõe uma extensão a modelo de *features* proposto por

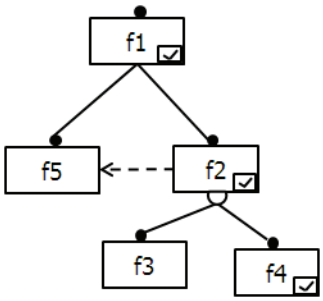

Figura 3.7: Modelo de *features* para exemplificar filtros de ações de correção.

Czarnecki *et al.* [6]. Assim, atributos com valores são associados às *features*. Por exemplo, a cada *feature* do exemplo da Figura 2.2 poderiam estar associados dois atributos: (i) um atributo de preço; e (ii) um atributo de grau de importância que essa *feature* possui na configuração dado o contexto onde será aplicado o produto, por exemplo. Dessa forma, Benavides *et al.* define conjunto de configuração ótima.

Definição 3.3 (Configuração Ótima [14]) *Seja FM um modelo de* features *estendido, ψ um problema CSP equivalente e O uma função objetivo, então o conjunto de produtos ótimos é dados por:*

$$\max(FM, O) = \max(\psi, O)$$
$$\min(FM, O) = \min(\psi, O)$$

Com essa abordagem, para encontrar uma configuração ótima deve ser construído um CSP ψ equivalente à FM e encontradas as soluções γ_i de ψ. Em seguida, a melhor ou o conjunto das melhores configurações, dada a função objetivo O, pode ser obtido mapeando cada γ_i em uma configuração.

No entanto, podem existir situações em que restrições adicionais são impostas no contexto de configuração. Tomando o exemplo do modelo de *features* estendido do kernel do Linux, uma configuração ótima em relação ao atributo de importância pode custar um valor x. No entanto, o cliente pode possuir um orçamento $o < x$. Dessa forma, esse cliente tem o requisito de adquirir a melhor configuração de um produto, cujo custo total seja, no máximo, igual o, por exemplo.

Portanto, estendemos a definição de uma configuração ótima conforme a Definição 3.4

Definição 3.4 (Configuração Ótima) *Seja FM um modelo de* features *estendido, ψ o problema CSP equivalente, O uma função objetivo e R um conjunto de restrições sobre os atributos de FM, então, o conjunto de produtos ótimos é dados por:*

$$\max(FM, O, R) = \max(\psi, O, R)$$
$$\min(FM, O, R) = \min(\psi, O, R)$$

Atribuímos ao gerente de produtos a responsabilidade de definição de atributos, na Fase 1 do processo de configuração (Figura 3.1). A configuração ótima pode ser utilizada inicialmente como ponto de partida da configuração e, a partir daí, os *stakeholders* realizariam uma revisão para ver se, por questões técnicas ou de negócio, por exemplo, será necessária alguma modificação. O segundo aproveitamento dessas informações pode ser feito através da utilização da função objetivo para classificação de sugestões, permitindo ao assistente pessoal oferecer ao seu *stakeholder*, primeiramente, aquelas que possam produzir melhores configurações.

3.7.1 Cálculo de uma configuração ótima

Sejam $A = \{a_{f_1}, a_{f_2}, ..., a_{f_n}\}$ e $B = \{b_{f_1}, b_{f_2}, ..., b_{f_n}\}$ dois conjuntos de atributos associados às *features* de um modelo de *features* $FM(F, C)$, com n *features*, e cada atributo $a_{f_i} \in A$ e $b_{f_i} \in B$ atributos associados à *feature* $f_i^m \in F$. Seja também r_b uma restrição sobre B, e regida por uma relação R, e f_i variáveis inteiras representando as *features* do modelo, onde $f_i \in \{0, 1\}$. Podemos encontrar uma configuração ótima utilizando os modelos matemáticos 3-1 e 3-2, para maximização ou minimização, respectivamente, de uma função objetivo.

$$\begin{aligned} \text{maximize} \quad & f_1 a_{f_1} + f_2 a_{f_2} + ... + f_n a_{f_n} \\ \text{subject to} \quad & b_{f_1} f_1 + b_{f_2} f_2 + ... + b_{f_n} f_n \leq r_b. \\ & C. \\ & f_1, f_2, ..., f_n \in \{0, 1\} \\ & a_{f_1}, a_{f_2}, ..., a_{f_n}, b_{f_1}, b_{f_2}, ..., b_{f_n} \in \mathbb{R} \end{aligned} \tag{3-1}$$

$$\begin{aligned} \text{minimize} \quad & f_1 a_{f_1} + f_2 a_{f_2} + ... + f_n a_{f_n} \\ \text{subject to} \quad & b_{f_1} f_1 + b_{f_2} f_2 + ... + b_{f_n} f_n \geq r_b. \\ & C. \\ & f_1, f_2, ..., f_n \in \{0, 1\} \\ & a_{f_1}, a_{f_2}, ..., a_{f_n}, b_{f_1}, b_{f_2}, ..., b_{f_n} \in \mathbb{R} \end{aligned} \tag{3-2}$$

Portanto, a solução desses programas lineares (PL) produz um conjunto de configurações ótimas que satisfazem tanto as restrições do modelo de *features* (conjunto C) – aquelas que restringem os estados das *features* do

modelo – quanto as restrições sobre recursos do cliente que está adquirindo esse produto.

Dada uma solução γ_i desses programas lineares, obtemos a configuração ótima selecionando as *features* $f_i^m \in F$ cuja variável $f_i = 1$ e, analogamente, desmarcando as *features* $f_i^m \in F$ cuja variável $f_i = 0$.

3.7.2 Classificando Sugestões

Outro benefício das informações adicionais dos atributos nas *features* e a definição de uma função objetivo é a possibilidade de classificação das sugestões de correção da configuração. Conforme discutido, quando o *stakeholder* realiza uma decisão e seu assistente pessoal verifica que um conflito foi produzido, diversos planos de ações de recuperação podem ser construídos. Assim, determinar qual desses planos produzirá a melhor configuração é bastante interessante. Para isso, com esse conjunto de planos de ações de recuperação em mãos, o assistente pessoal constrói, para cada um desses planos, todos os estados de configuração S_i que seriam produzidos, caso fossem escolhidos. Em seguida, o agente aplica a função objetivo $O(S_i)$, definida pelo gerente de produto, sobre cada um desses estados. Por fim, o assistente ordena essas sugestões e as oferece a seu stakeholder.

3.7.3 Atualização dos Algoritmos de Produção de Planos de Ações de Recuperação

Note que, com a introdução de atributos nas *features* e restrições extra sobre a configuração, é necessário que os assistentes pessoais também as considere, tanto na validação, quanto na produção de sugestões de ações corretivas. Portanto, após cada decisão que um *stakeholder* produzir, seu assistente, além de validar a configuração levando em conta as restrições do modelo de *features*, conforme explicado na Seção 3.6.2, ele aplica a função objetivo sobre a nova configuração, a fim de verificar se os novos requisitos também estão sendo respeitados. Caso eles estejam, a atividade continua conforme já discutido. No entanto, quando essas restrições adicionais são desrespeitadas, um raciocínio adicional deve ser realizado.

Para o algoritmo de construção de planos de ações de recuperação, esse deve ser atualizado, incluindo as restrições adicionais no modelo do CSP. Note que uma modificação a mais deve ser realizada. Como o algoritmo, assim como o de validação, usa apenas as restrições vizinhas de f, com essa nova restrição, pode ser o caso de nenhuma solução ser encontrada nesse subconjunto. Isso

ocorre porque essas restrições, em geral, são sobre todas as *features* do modelo. No exemplo da seção anterior, o cliente pode ter um orçamento o que a configuração corrente já tenha ultrapassado. Quando o algoritmo é construído apenas considerando as *features* vizinhas de f, no grafo de dependências, a remoção de todas elas pode ainda produzir uma configuração com valor x maior que o. Dessa forma, o algoritmo é atualizado para tentar obter uma solução com as restrições à distância 1 de f, do grafo de dependências, que é a versão atual. Caso não sejam encontradas soluções, o algoritmo tenta utilizando as restrições à distância 1 e 2. Essa nova versão do algoritmo procede até que seja encontrada pelo menos uma solução para correção da configuração, que será, no pior caso, quando todo o modelo de *features* é considerado.

3.8 Conclusão

Propomos neste capítulo uma maneira nova de abordar o problema de configuração colaborativa de um produto de software a partir de uma LPS. Através dessa abordagem os *stakeholders* podem configurar um produto de maneira dinâmica. Dessa maneira, conseguimos tratar diversos cenários de configurações em que as abordagens atuais falham, além de propor uma alternativa viável e eficiente para ser utilizada em cenários nos quais as demais já são aplicáveis. Utilizamos para isso técnicas da área de sistema multiagentes, que são fundamentais para a nossa abordagem e propomos alguns conceitos como, por exemplo, *estado de configuração global*, *planos de ações de correção* e *grafo de dependências*. Nossa abordagem também provê suporte à negociação de decisões entre *stakeholders* e garante que apenas configurações válidas sejam produzidas ao final do processo.