4
Libertas: Uma Ferramenta de suporte à Configuração Dinâmica e Distribuída de LPS

Nesta seção, descrevemos a implementação dos conceitos propostos por nosso trabalho. Essa implementação permite verificar a aplicabilidade da abordagem dinâmica e distribuída para configuração colaborativa de LPS. Referenciamos essa ferramenta por *Libertas*. Apresentamos uma visão geral da arquitetura na Seção 4.1. A descrição dos módulos utilizados pelos assistentes pessoais para raciocínio sobre o modelo de *features* é dada na Seção 4.2. A dinâmica da ferramenta é discutida na Seção 4.3. Na Seção 4.4 apresentamos um estudo sobre o desempenho na produção de soluções. Concluímos o capítulo na Seção 4.5.

4.1
Visão Geral da Arquitetura

A ferramentas *Libertas* possui duas versões distintas: uma para o gerente de produto; e outra para os *stakeholders* configuradores que participam da configuração. A versão de gerente de produto suporta o usuário interpretando esse papel nas tarefas de distribuição do modelo de *features* para os demais *stakeholders* e de aceitação do produto final, no término da configuração. A versão de *stakeholder* suporta os demais participantes durante toda a atividade de configuração, observando e validando suas ações, sugerindo planos de ações de recuperação e configurações ótimas, além de integrar decisões com os demais *stakeholders*.

A ferramenta foi desenvolvida sobre a plataforma Java[1] e sua arquitetura, conforme pode ser observada na Figura 4.1, é composta, para ambas as versões, por três elementos: (i) *interface gráfica de usuário*; (ii) *assistente pessoal*; (iii) *modelo de* features. A interface gráfica é o local por onde o usuário interage no processo de configuração e por onde o *stakeholder* é notificado sobre eventos gerados em respostas de suas próprias ações ou dos demais *stakeholders*. Além disso, cada instância da ferramenta possui uma cópia do modelo de *features*. Dessa instância o assistente pessoal obtém as informações

[1]http://www.oracle.com/technetwork/java/index.html

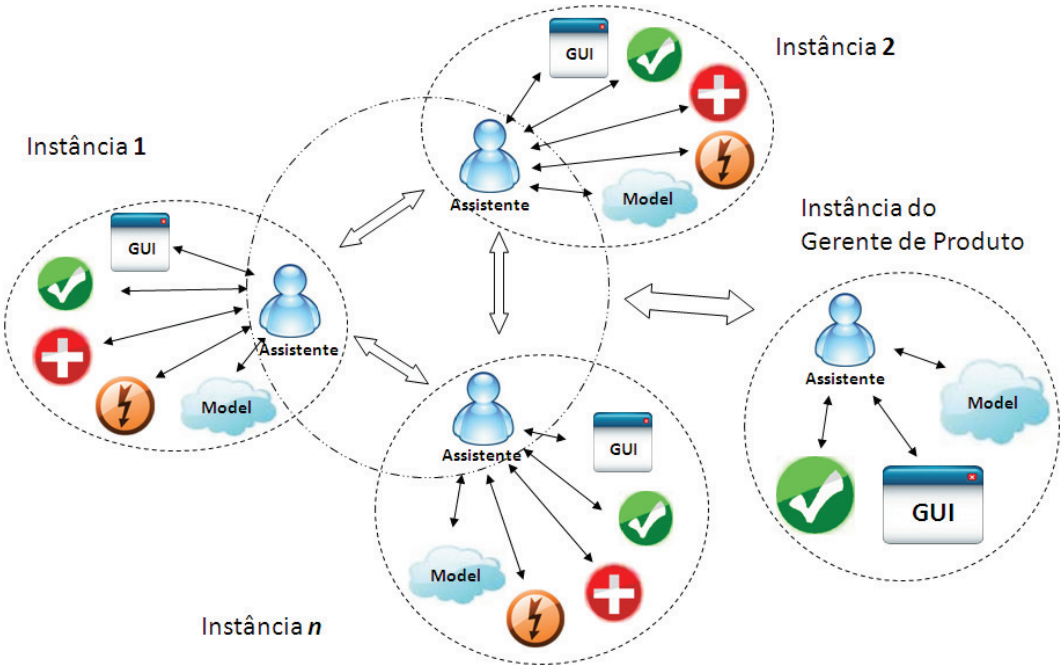

Figura 4.1: Arquitetura da ferramenta *Libertas*.

das *features*, conjunto de decisões, escopo de papéis e a configuração local. Por fim, o assistente pessoal é aquele que provê o suporte às atividades do *stakeholders*. Para poder realizar suas funções, esse agente computacional utiliza três módulos de raciocínio sobre o modelo de *features*: (i) *módulo de validação*; (ii) *módulo de criação de planos de ações de recuperação*; e (iii) *módulo de otimização*. Os três elementos que compõem a ferramenta, assim como os módulos de raciocínio são discutidos em detalhe nas próximas seções.

4.1.1 Interface Gráfica de Usuário

A *Interface Gráfica de Usuário* (GUI) é onde o usuário realiza suas atividades de configuração do produto. Na Figura 4.2 é apresentado um exemplo da interface gráfica principal da ferramenta, pertencente a um *stakeholder* interpretando o papel *Hardware Manager*. A interface é carregada assim que o assistente obtém o modelo de *features* com o assistente do gerente do produto e papel que seu *stakeholder* irá interpretar é identificado. Note que, conforme é sugerido pela proposta, como um papel tem autoridade apenas sobre um subconjunto do modelo de *features*, então apenas essas *features* são passíveis de edição – serem selecionadas ou não pelo respectivo *stakeholder*. Portanto, as *features* que vemos na figura com a caixa de seleção e o nome na cor cinza são as *features* a que o *stakeholder* só tem acesso de leitura. Nas demais, o *stakeholder* pode realizar decisões conforme desejar. Ainda na figura da GUI, podemos observar um ícone de alerta vermelho ao lado da *feature* "*Hyper-*

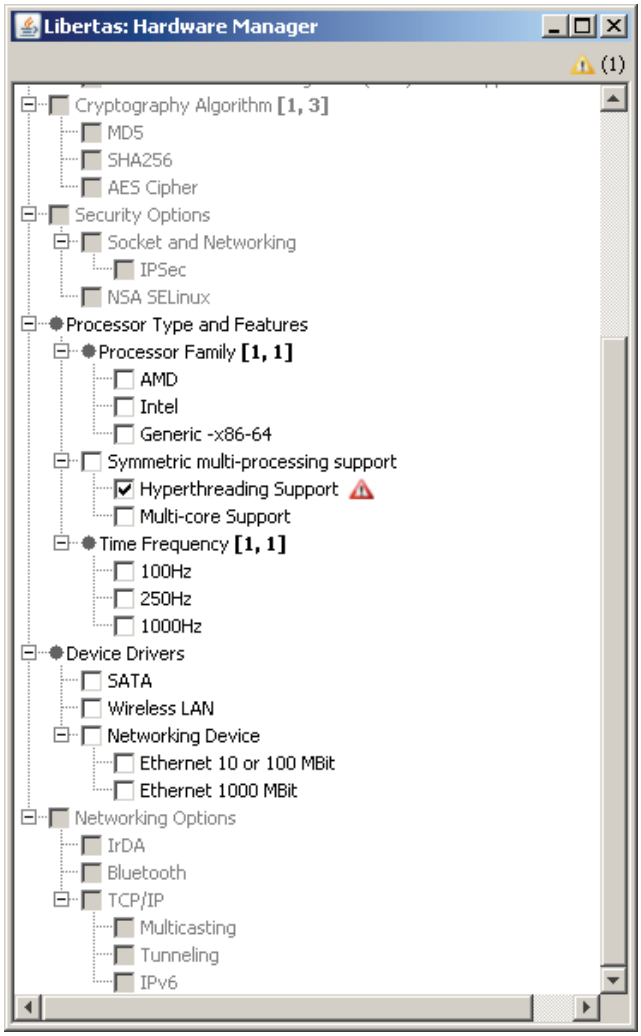

Figura 4.2: Interface principal da ferramenta Libertas.

Threading Support". Esse ícone é uma indicação visual de que a *feature* está envolvida em um conflito. O *stakeholder* pode clicar com o botão esquerdo do mouse nesse ícone para obter sugestões de como recuperar a configuração. Após essa ação, uma segunda janela é exibida (Figura 4.3), com opções de como resolver o conflito. Nesse exemplo, como a *feature* "*HyperThreading Support*" é uma feature-filha de "*Symmetric Multiprocessing Support*", além de depender a *feature* "*Intel*", é necessário que essas *features* sejam selecionadas ou, então, "*HyperThreading Support*" deve ser desmarcada.

Note também outro ícone de alerta amarelo no canto superior direito da Figura 4.2. Esse alerta indica que alguma solicitação foi realizada por outro *stakeholder*. O contador ao lado direito desse ícone mostra a quantidade de solicitações que estão em espera. Ao clicar nesse alerta, uma janela parecida com a da Figura 4.3 aparecerá, com detalhes da solicitação recebida. Um exemplo será mostrado na Seção 4.3.

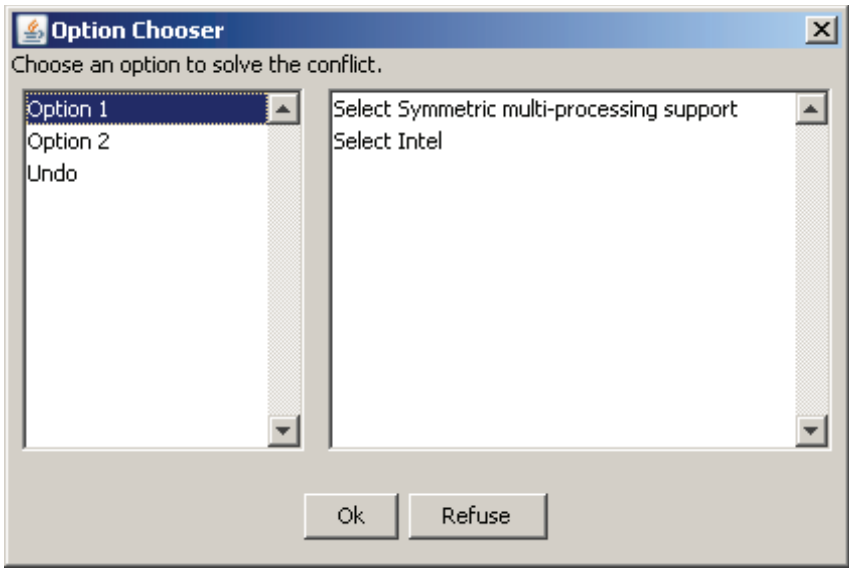

Figura 4.3: Interface principal da ferramenta Libertas.

4.1.2 Modelo de Features

Para representar o modelo de *features* dentro da ferramenta, definimos um *schema XML*. Esse *schema* possibilita a descrição de *features*, restrições, conjuntos de decisão e papéis. A utilização de arquivos *XML* facilita atividade de distribuição do modelo de *features* com os demais *stakeholders* pelo gerente do produto, em um ambiente distribuído, além de possibilitar a leitura e escrita dos dados, de forma fácil, através de um padrão aberto. Utilizamos a API *JAXB (Java API for XML Binding)*[2] para manipular essas informações no formato *XML* dentro do código Java.

A Figura 4.4 apresenta a estrutura do modelo de *features* descrito pelo *schema XML*, através de um diagrama de classes. Observe que para representar todos os tipos de *features*, conforme apresentado na Seção 2.1.1, utilizamos apenas duas classes: *Feature* e *GroupedFeature*. *feature* é a classe que define os atributos básicos de uma *feature*, que pode, inclusive, possuir *features*-filha. No entanto, essa relação de parentesco apenas introduz uma restrição de: se uma feature-filha estiver estiver selecionada, a feature-pai também deve estar. A segunda classe, *GroupedFeature*, possui dois atributos a mais, *min* e *max*, que indicam a quantidade mínima e máxima, inclusive e respectivamente, de *features*-filhas que devem ser selecionadas caso a feature-pai estiver. Com essa modelagem é possível representar todos os tipos de *features* citadas na Seção 2.1.1: *or-feature*, *and-feature*, *alternative-feature* e a *baseada em cardinalidade*.

[2]http://jaxb.java.net/

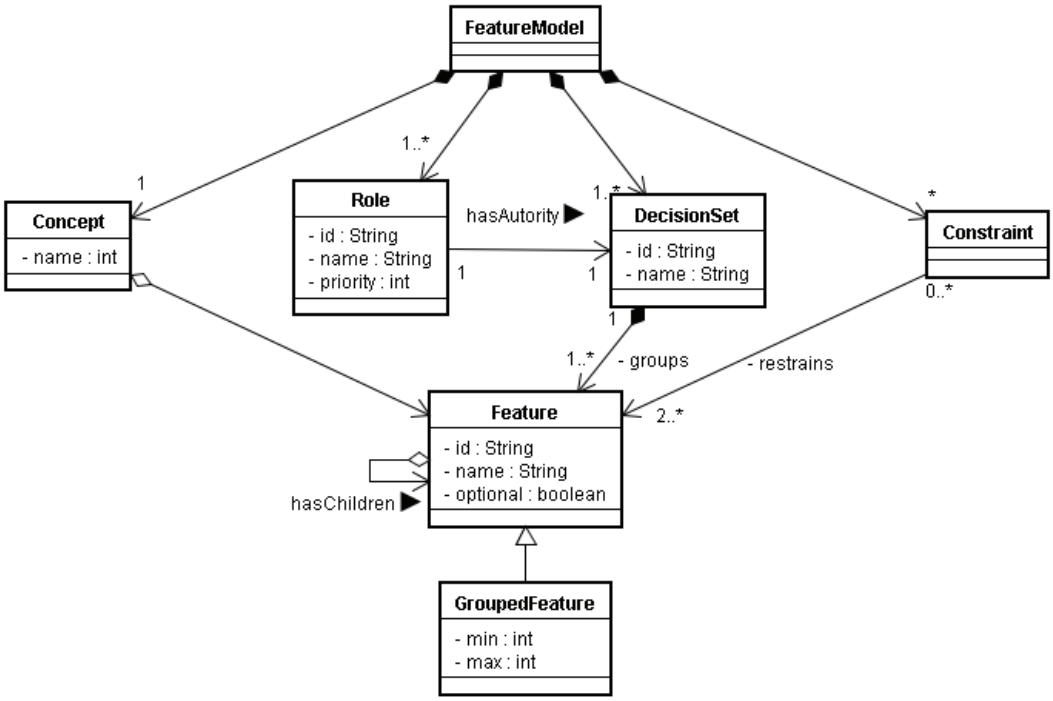

Figura 4.4: Diagrama de classes do modelo de *features* utilizado.

Uma entidade importante que criamos dentro da ferramenta *Libertas* é o gerenciador do modelo de *features*. O gerenciador do modelo de *features* é um classe que provê acesso à instância do modelo, abstraindo os detalhes de representação. Essa classe tem a função de carregar o modelo de *features* de um arquivo e transformá-la em uma representação interna da ferramenta, que o assistente pessoal do *stakeholder* conheça. Além disso, ela concentra métodos de consulta das informações mais detalhadas do modelo de *features*.

4.1.3 Assistente Pessoal

O *assistente pessoal* é o um agente computacional que compõe a ferramenta e cujo objetivo principal é assistir o usuário durante o processo de configuração de um produto. Esse agente de usuário pode ser um *agente de gerente de produto* (AGP) ou um *agente de stakeholder* (ASH), dependendo do usuário alvo. O assistente é responsável por representar o *stakeholder* durante o processo de configuração, abstraindo dele as complexidades dessa atividade. Para o desenvolvimento do agente de usuário, seja ele um AGP ou um ASH, utilizamos os conceitos de sistemas multiagentes, conforme discutido na Seção 2.3. A implementação utiliza o framework Jade[3] [38]. Jade é uma plataforma que simplifica o desenvolvimento de sistemas multiagentes, atendendo as especificações padrão. Essa plataforma permite que os agentes estejam distribuídos sobre

[3]http://jade.tilab.com/index.html

várias máquinas, provendo um API para a comunicação entre eles, através de mensagens. Nossa ferramenta se beneficia dessa plataforma utilizando essa infraestrutura de mensageria para comunicação entre diferentes ambientes de configuração. Além disso, a abordagem orientada a comportamentos, suportada pelo Jade, facilitou a implementação das funcionalidades dos assistentes pessoais.

Destacamos os três principais comportamentos dos assistentes pessoais: (i) *ExecuteDecisionBehavior*; (ii) *DecisionReceivedBehavior*; e (iii) *ActionsRequestedBehavior*.

O primeiro comportamento é executado assim que o *stakeholder* produz alguma ação na interface gráfica. Nesse comportamento o assistente atualiza configuração local do modelo de *features*, informa os demais assistentes desse novo estado e, por fim, constrói planos de ações de recuperação, caso um conflito tenha sido gerado. Além disso, os alertas na interface são atualizados.

O segundo comportamento citado, *DecisionReceivedBehavior*, é invocado quando o assistente recebe uma comunicação de que alguma decisão foi produzida em outro conjunto de decisão. Esse comportamento atualiza a configuração, verificando se essa decisão resolveu ou produziu outro conflito. Assim, os alertas da interface são atualizados. Note que a execução do comportamento *DecisionReceivedBehavior* é uma resposta a execução do comportamento *ExecuteDecisionBehavior* em outro ambiente de configuração.

Por fim, *ActionsRequestedBehavior* é o comportamento inicializado quando outro *stakeholder* participante da configuração solicita ações para corrigir algum conflito que ele tenha produzido. Nesse comportamento o assistente converte a mensagem de solicitação em uma representação conhecida internamente, adicionando um alerta de requisição na interface gráfica – o ícone de alerta amarelo. Quando o *stakeholder* clicar nesse alerta, as opções de ações solicitadas serão exibidas e o *stakeholder* poderá escolher uma ou rejeitar todas.

4.2 Módulos de Raciocínio

Nesta seção descrevemos os módulos principais utilizados pelos assistentes pessoais para raciocinar sobre o modelo de *features*. Essa inteligência dos agentes permite diminuir o trabalho dos *stakeholders* durante a atividade de configuração. É exigido dos *stakeholders* apenas conhecimento dos requisitos, para poderem decidir se uma *feature* estará presente ou não na configuração final e, em um caso de conflito, escolherem uma das opções que melhor contorne essa situação. São descritos nas seções seguintes os módulos de: (i) *validação*; (ii) *produção de planos de ações de recuperação*; e (iii) de *otimização*.

4.2.1 Módulo de Validação

O *módulo de validação* oferece uma API que valida tanto uma configuração inteira do modelo de *features* quanto a configuração de um subconjunto desse modelo. Esse módulo utiliza-se dos métodos de acesso ao modelo de *features* oferecidos pelo gerenciador do modelo de *features* para obter a configuração e suas restrições, e, então, realizar sua tarefa de verificação se todas essas restrições estão sendo satisfeitas pela configuração corrente ou não. Para validação utilizamos o algoritmo apresentado na Seção 3.6.2.

Assim, é obtido do grafo de dependências um conjunto de restrições C adjacentes à *feature* que sofreu a decisão e, então, uma expressão booleana é construída, concatenando cada expressão $expr_c$, associada à restrição $c \in C$, através do operador *and*. Em seguida, a cada variável correspondente às *features* envolvidas é atribuído o valor *true* ou *false*, conforme seu estado de configuração. Por fim, a expressão construída é avaliada. Se o valor apurado for *true*, então a decisão não produziu conflito ou resolveu algum conflito existente. Caso contrário, o assistente inicia o procedimento de sugestão, invocando o módulo de produção de planos de ações de recuperação.

No caso em que a questão de otimização está sendo considerada na configuração, uma validação extra das restrições sobre os atributos é realizada. Por exemplo, se foi definida uma restrição que somatório dos atributos a de cada *feature* selecionada tem que ser menor ou igual a uma constante r, então essa verificação é executada após a avaliação das restrições do modelo de *features*.

4.2.2 Módulo de Produção de Planos de Ações de Recuperação

O *módulo de produção de planos de ações de recuperação* é utilizado pelos assistentes pessoais para ajudar os *stakeholders* a se recuperarem configurações inválidas. Esse módulo oferece uma API para análise das restrições, dado um estado de configuração do modelo de *features*. Para realizar esse raciocínio sobre essas informações, o módulo utiliza o grafo de dependências, que é criado quando o ambiente de configuração é carregado. Como base da implementação do módulo, utilizamos o resolvedor de problemas de restrições *Choco Constraint Solver*[4]. Choco é uma biblioteca implementada em Java bastante eficiente para resolução de problemas de restrições.

Portanto, em resposta a uma decisão que tenha provocado um estado de inconsistência, o assistente pessoal obtém do grafo de dependência o conjunto

[4]http://www.emn.fr/z-info/choco-solver/index.html

de restrições adjacentes à *feature* que sofreu a decisão, assim como todas as *features* relacionadas. Com esses dois conjuntos (*features* e restrições) é construído um CSP, traduzindo a representação de nossa ferramenta para as classes oferecidas pelo Choco. Em seguida, esse modelo de restrição é passado para uma instância da classe *CPSolver*, que retorna todos os possíveis estados após invocar o método *CPSolver.solve()*. O próximo passo é traduzir novamente essas soluções para a representação da ferramenta, produzindo o conjunto de sugestões usando o algoritmo apresentado na Seção 3.6.

4.2.3 Módulo de Otimização

O *módulo de otimização* oferece uma API para raciocínio sobre a estrutura do modelo de *features* (*features*, restrições e valores dos atributos) e os requisitos do cliente (restrições sobre recursos ou necessidades mínimas) com objetivo produzir uma configuração ótima. Assim, antes de iniciar a configuração, o gerente de produtos tem a opção de calcular configuração ótima. Essa configuração pode ser utilizada como estado de configuração global inicial. Para calcular uma configuração que maximize ou minimize algum atributo do modelo de *features*, considerando restrições sobre outro atributo, utilizamos também a biblioteca Choco. Assim, quando o gerente de produto solicita essa configuração, ele informa o modelo de *features*, o atributo que será otimizado e o atributo que possui alguma restrição associada. Utilizamos o modelo matemático da Seção 3.7.1 para construir o programa linear equivalente ao modelo de *features* e às restrições.

4.3 Dinâmica da Ferramenta

Para começar o processo de configuração, o gerente do produto deve inicializar sua instância da ferramenta, indicando o arquivo com modelo de *features* que será utilizado para derivar um produto. Dessa forma, após carregar esse arquivo, o assistente pessoal, com auxílio do gerenciador do modelo de *features*, transforma o modelo de *features* em uma representação conhecida dentro do ambiente de configuração. Em seguida, o assistente fica à disposição dos demais *stakeholders* que participarão da configuração, para fornecê-lhes esse modelo.

A partir desse momento, cada *stakeholder* configurador pode inicializar sua respectiva instância de configuração. Assim, seu assistente pessoal ap_i é automaticamente carregado e sua primeira tarefa é procurar o assistente ap_g do gerente do produto no ambiente. Em seguida, ap_i envia uma mensagem a

ap_g, solicitando uma cópia do modelo de *features*. Com o modelo em mãos, o ap_i identifica os papéis de decisão e solicita a seu *stakeholder* indicar qual será interpretado por ele. Após a escolha, ap_1 envia uma segunda mensagem, informando o papel escolhido. A partir desse momento, o *stakeholder* está pronto para realizar a configuração, apenas esperando todos os envolvidos realizarem o mesmo processo. Quando todos os *stakeholders* estão com seus ambientes corretamente carregados e os papéis devidamente distribuídos, a configuração pode ser iniciada.

Durante a fase de configuração do produto podem surgir três situações distintas, conforme identificado na Seção 3.4.2:

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

2. **A decisão resulta em uma configuração inválida do modelo de *features*, porém não tem um impacto sobre outros conjuntos de decisão**;

3. **A decisão tem dependência direta de inclusão ou exclusão de decisões em outros conjuntos de decisão**;

O Item 1 é o caso base. Por exemplo, na situação retratada pela Figura 4.2, supondo que o *stakeholder* tinha como requisito que a o sistema operacional que está sendo configurado será utilizado para controlar uma plataforma móvel que possuirá um processador que opera na frequência de 250Hz, então ele deverá selecionar a *feature "250Hz"*. Como pela definição do modelo de *features* não existe nenhuma restrição sobre essa *feature*, então não é produzido um conflito. Assim, todos os outros assistentes pessoais são informados, fazendo com que todas as configurações locais sejam atualizadas.

Um exemplo para a situação do Item 2 é apresentada pela Seção 4.1.1. Como existe uma restrição sobre a *feature "HyperThreading Support"* e ela não está sendo respeitada, o assistente pessoal tem a responsabilidade de alertá-lo. No entanto, como todas as *features* envolvidas nesse conflito pertencem ao mesmo conjunto de decisão, então a responsabilidade de resolvê-lo é do próprio *stakeholder*. Portanto, são oferecidas opções de correção e após a escolha de uma delas, os outros ambientes de configuração são atualizados. Os demais *stakeholders* não ficam sabendo desse conflito.

Por fim, o caso mais complexo é descrito pelo Item 3. Vamos supor a situação em que um *stakeholder* interpretando o papel *Networking Manager* escolha a *feature "TCP/IP"*, conforme na parte *A* da Figura 4.5. Note que duas

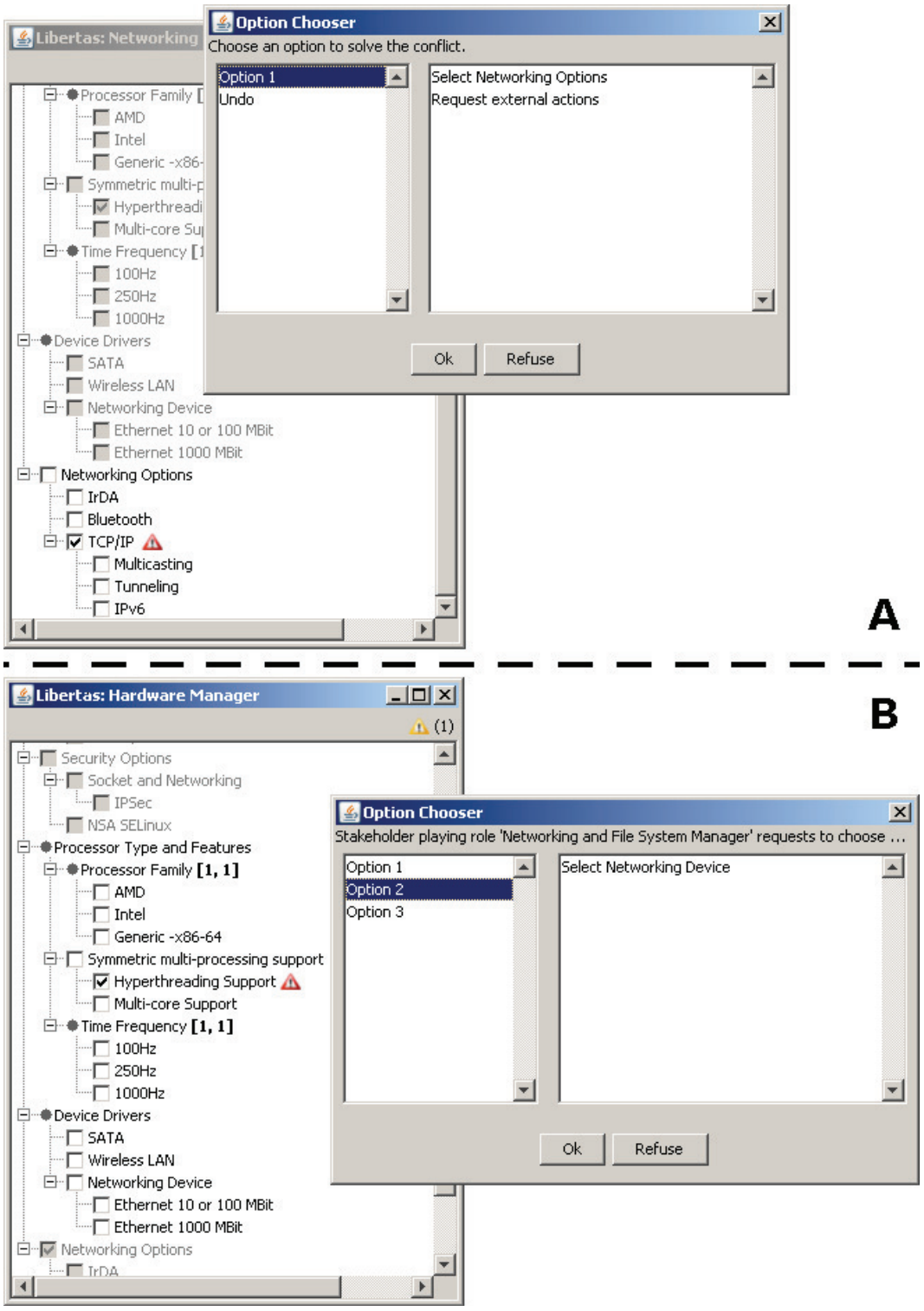

Figura 4.5: Exemplo de solicitação de ações.

restrições são impostas sobre essa feature: (i) de parentesco com "*Networking Options*"; e (ii) *TCP/IP* → *Networking Device Support OR Wireless LAN*. Assim, o assistente pessoal percebe que o estado de configuração corrente está produzindo um conflito e que, para resolvê-lo, é necessário negociar com outro *stakeholder*. Então, o assistente adiciona um alerta ao lado da *feature* "*TCP/IP*", para avisar seu *stakeholder*. Ao clicar nesse alerta, são oferecidas duas opções: (i) selecionar a *feature* "*Networking Options*"e solicitar que os *stakeholders* envolvidos o ajudem, realizando um conjunto de ações; ou (ii) desfazer a decisão de seleção da *feature*. Ao escolher a segunda opção, a configuração volta ao estado de configuração anterior. No entanto, ao escolher a Opção 1, uma mensagem é enviada aos assistentes dos *stakeholders* envolvidos no conflito, solicitando ajuda na sua resolução.

No exemplo da Figura 4.5, apenas *features* pertencentes ao *Hardware Manager* estão envolvidas no conflito, além daquelas próprias do *Networking Manager*. Assim, ao receber essa solicitação, o assistente pessoal do *Hardware Manager*, automaticamente, adiciona um alerta de solicitações na interface gráfica ou aumenta o contador. Esse *stakeholder*, ao perceber essa notificação, pode clicar no alerta, sendo mostrada uma janela com detalhes da solicitação, conforme mostrado na parte *B* da Figura 4.5. Assim, são oferecidas três opções que resolvem o conflito gerado pelo *Networking Manager*: (i) selecionar a *feature Wireless LAN*; (ii) selecionar *Networking Device*; ou (iii) selecionar as duas *features*. O *Hardware Manager* pode escolher a mais adequada às suas restrições, ou rejeitá-las. Note que somente foram solicitadas ações sobre *features* que estão sob responsabilidade desse *stakeholder*. Os assistentes pessoais têm inteligência suficiente para distribuir as solicitações conforme a autoridade de cada *stakeholder*, para realizar a negociação do novo estado de configuração, que resolva o conflito.

Outro ponto interessante de ser destacado é que a negociação de ações sobre um conjunto de *features* é independente de outros conflitos que as demais *features* já possam ter. Por exemplo, quando o *stakeholder Networking Manager* solicitou ajuda para resolver o conflito que ele tinha produzido, o *Hardware Manager* já possuía outro conflito em seu ambiente de configuração (*feature HyperThreading Support*). As ações para correção dos conflitos podem ocorrer em qualquer ordem.

4.4 Avaliação de Desempenho

Nesta seção apresentamos uma avaliação do desempenho da ferramenta para produção de planos de ações de recuperação. Conforme discutimos na

Seção 3.6.3, utilizamos o CSP para produzirmos esses planos. No entanto, a aplicação direta do CSP sobre todo o modelo de *features* para calcular todas suas possíveis configurações, nos levaria a um algoritmo $O(2^n)$, pois estaríamos tentando encontrar combinações de valores binários (selecionado ou desmarcado) para n variáveis. Dessa forma, essa abordagem para encontrar soluções para grandes modelos poderia se tornar impraticável.

Assim, propusemos na Seção 3.6.1 o grafo de dependências. Dessa forma, para calcularmos as soluções de recuperação de uma configuração, utilizamos apenas um subconjunto de variáveis e restrições do contexto total. Da forma como o grafo de dependências é construído, unindo através de arestas as *features* que possuem alguma restrição entre si, podemos obter uma solução de forma mais eficiente para o problema. Assim, portanto, ao invés de termos um algoritmo exponencial sobre a quantidade de *features* do modelo, obtemos um algoritmo que cresce exponencialmente conforme a densidade do grafo de dependências. Definimos densidade como a razão entre a quantidade de restrições e a quantidade de *features* do modelo.

Realizamos um experimento para avaliar e verificar essa complexidade. Na Tabela 4.1 apresentamos os resultados. Como entrada do experimento, utilizamos cinco tipos de modelos de *features* gerados automaticamente, que variam em quantidade de *features*, quantidade de restrições e densidade, e que foram obtidos do site *SPLOT (Software Product Lines Online Tools)* [5]. Nesses modelos gerados automaticamente, a probabilidade de cada *feature* ser mandatória, opcional, *or-feature* e *alternative-feature* é a mesma. O fator do número de filhas que uma *feature* pode ter varia entre 1 e 6. Todas as restrições são definidas na forma 3-CNF (Forma Normal Conjuntiva).

Para cada tipo de modelo de *features*, executamos o algoritmo sobre dez modelos. Assim, na Tabela 4.1, temos o tempo gasto para encontrar a solução, assim como a média para cada tipo de modelo de *features*. Atribuímos configurações aleatórias para cada modelo e calculamos os planos de recuperação para cada *feature* em estado de conflito. Em cada modelo, F significa o número de *features*, C o número de restrições e D a densidade das cláusulas 3-CNF. Através do gráfico da Figura 4.6, podemos notar que, apesar de alguns modelos de *features* possuírem quantidade de *features* muito maiores, o que mais influencia o crescimento do tempo de execução do algoritmo é o grau de dependência entre as *features*, em vez do tamanho do modelo.

[5] http://www.splot-research.org/

Tabela 4.1: Resultados da aplicação do algoritmo de produção de ações de recuperação.

Modelo	**F:500-C:50-D:0,10**	**F:1000-C:100-D:0,10**	**F:2000-C:100-D:0,05**
1	1,102	1,205	0,733
2	1,105	1,116	0,706
3	1,123	1,189	0,713
4	1,179	1,132	0,718
5	1,205	1,154	0,715
6	1,270	1,190	0,703
7	1,284	1,073	0,705
8	1,171	1,114	0,717
9	1,178	1,269	0,716
10	1,135	1,092	0,729
Média	1,171	1,149	0,707

Modelo	**F:5000-C:150-D:0,03**	**F:10000-C:100-D:0,01**
1	0,633	0,599
2	0,633	0,605
3	0,655	0,601
4	0,636	0,602
5	0,630	0,593
6	0,655	0,595
7	0,641	0,609
8	0,647	0,596
9	0,630	0,608
10	0,639	0,607
Média	0,633	0,595

4.5 Conclusão

Apresentamos neste capítulo a ferramenta Libertas, construída para verificar a aplicabilidade da abordagem de configuração dinâmica e colaborativa proposta por esse trabalho. Essa ferramenta foi implementada com a linguagem Java, utilizando o framework Jade, para os agentes computacionais, e a API *JAXB*, para manipulação de arquivos *XML* dentro código do código Java. Apresentamos uma visão geral da arquitetura, assim como o detalhamento dos três componentes principais que compõem cada instância de configuração de Libertas: *interface gráfica*, *assistentes pessoais* e *modelo de* features. Além disso, foram discutidos os principais módulos de raciocínio implementados pelos assistentes pessoais e um exemplo da dinâmica da ferramenta. Por fim, um experimento sobre o desempenho do algoritmo de produção de ações de recuperação é apresentado, mostrando que ele é eficiente, mesmo para grandes modelos de *features*.

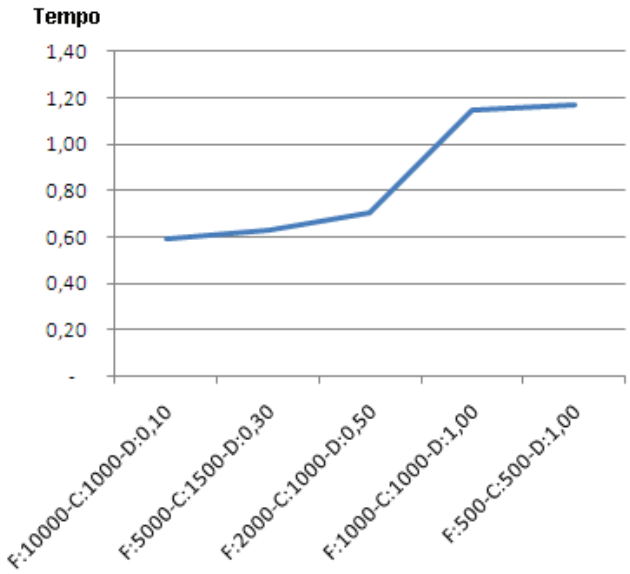

Figura 4.6: Análise do algoritmo de produção de ações de recuperação sobre cinco tipos de modelos de *features*.