Sistema de Substituição: uma técnica reativa para auto-reparo e auto-diagnóstico de planos.

Claus de Castro Aranha

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Sistema de Substituição: uma técnica reativa para auto-reparo e auto-diagnóstico de planos.

Claus de Castro Aranha

Março de 2005

Banca Examinadora:

- Jacques Wainer (Orientador)
- Luiz Marcos Garcia Gonçalvez - UFRN
- Siome Klein Goldenstein
- Ariadne Maria Brito Rizzoni Carvalho (Suplente)

Substitua pela ficha catalográfica

Sistema de Substituição: uma técnica reativa para auto-reparo e auto-diagnóstico de planos.

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Claus de Castro Aranha e aprovada pela Banca Examinadora.

Campinas, 31 de março de 2005.

Jacques Wainer (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Substitua pela folha com a assinatura da banca

Prefácio

Um dispositivo planejador deve estar preparado para lidar com aspectos dinâmicos do sistema. Falta de conhecimento do mundo, mudanças de natureza dinâmica, ações com resultados não determinísticos e falhas podem fazer com que o ambiente no qual o planejador executa uma tarefa seja diferente daquele contemplado pelo plano original. Quando isso acontece, o plano pode se tornar inválido, ou seja, as pré-condições nas quais o plano se baseia não são mais verdadeiras, e os objetivos da tarefa não podem ser alcançados.

O trabalho de contornar esse problema, garantindo que um planejador saia de uma situação de plano inválido e alcance seus objetivos originais se chama de *reparo de planos*. Em geral as técnicas de reparos de planos podem ser classificadas como técnicas de *replanejamento* ou técnicas de *planejamento condicional/planejamento probabilístico*. Replanejamento consiste em fazer novamente o plano a partir do ponto onde ocorre a falha. Replanejadores são em geral robustos, mas possuem uma complexidade que pode dificultar sua utilização em tempo real. Planejadores probabilísticos/condicionais procuram gerar um plano antes da execução do mesmo que leve em conta as incertezas do sistema. Para sistemas muito complexos, eles podem não se mostrar capazes de resolver o problema proposto.

Neste trabalho propomos uma nova técnica de reparo de planos, baseada na substituição de ações. Uma aplicação computacional complexa em geral oferece várias maneiras de executar uma mesma ação, embora em geral apenas a mais eficiente dessas maneiras seja levada em consideração pelos planejadores. No nosso *sistema de substituição*, o planejador estuda os dispositivos disponíveis e seus relacionamentos para montar uma tabela de substituição, que lista subplanos que possam ser utilizados para substituir, com menor eficiência (degradação graciosa) ações normais do sistema.

Analisamos as características do relacionamento de uma ação e seus subplanos de substituição, e como montar a tabela de substituição necessária. Descrevemos então o algoritmo que implementa a técnica de uma maneira geral. Utilizamos então esse quadro geral para tecer comentários sobre a implementação do sistema de substituição em um robô simulado, um robô bípede, e sistemas de web services.

Abstract

A planner must be prepared to deal with dynamic characteristics of the system it acts upon. Lack of world knowledge, dynamic changes, actions with non-deterministic results and faults may put the environment in a state different from the one the planner was expecting while performing a task. When this happens, the pre-conditions required for the plan may become false, turning the plan itself invalid and unable to reach the task goals.

The problem of recovering from an invalid plan and achieving the task's original goals is called *plan repair*. Usually, plan repair techniques can be classified as either *replanning* techniques or *conditional/probabilistic planning*. Replanning consists of creating a new plan from the point of failure. Replanners are usually robust but too complex for using them in real-time applications. Probabilistic/conditional planners try to generate offline a plan that take into account the system's uncertainties. However, they might be unable to do so if the environment is too complex.

In this work we propose a new plan repair technique based on action replacement. Any complex application will usually offer many different ways to achieve the results of any given action, although usually only the most efficient one is taken into account by the planner. In our *replacement system*, the planner will study the available devices and their relationship to build a replacement table, which lists subplans that can be used to replace a regular action with lessened efficiency (gracious degradation).

We analyze the characteristics of the relationship between an action and its replacement subplans, and how to assemble the required table from this information. We describe the algorithm which implements the technique in a general context. We utilize this definition to apply the technique for simulated robots, a biped robot and web services.

Agradecimentos

Devo este trabalho totalmente ao professor Wainer, meu orientador. Muitas vezes eu me perdi e me preocupei durante o desenvolvimento deste trabalho, e ele sempre esteve lá para me colocar de volta nos trilhos e me empurrar para frente novamente. Não tenho como agradecer todo o apoio e compreensão recebidos.

Outra pessoa importante para este trabalho é o professor José Jesus Perez, professor visitante no IC. Veio dele o chute inicial para idéia de aplicar o sistema de substituição a Web Services, casamento que se deu impressionantemente bem, além de várias dicas e orientações valiosas para o meu trabalho como um todo.

O Super Luiz Marcos foi quem me introduziu ao mundo dos robozinhos ("roubando-me" do grupo de estudos de criptografia do professor Dahab), e por pouco não foi meu orientador neste trabalho. Kudos para ele!

Agradecimentos a todo o pessoal do IC que esteve envolvido no rolo com a minha matrícula em dezembro/janeiro. Sem o trabalho deles, eu não seria mais um aluno regular, e não poderia defender esta tese.

Fora da academia, meu agradecimento especial vai para cada um dos Legonianos: Bicudo, Raffaels, Amauri, Danilo, Bell, Sábio do quarto do fundo e seu irmão, Sol, Ronaldo, Reis, Júlio, Covic, Lube, Obi, Beholder, Bixo, Planta e Rogério. Por conta dessas pessoas pude me sentir em casa nestes sete anos de UNICAMP.

Agradeço também meus pais, que serviram de rede durante este tempo todo em que eu aprendi a andar na corda bamba.

Fora os agradecimentos pessoais, tenho uma profunda dívida a tudo aquilo que faz da Unicamp um lugar especial para se viver: O observatório, o belo pôr do sol que se vê do IC, a barraquinha de sucos da BC, o Oficina Coral, o laboratório 302, o Bandex, o laguinho, a Padaria Alemã, as ruas transdimensionais de Barão Geraldo, o Chicão, o hot-dog do tiozinho, o muro de escalada da FEF, o universo das massas, o Camões, e outras coisas importantíssimas que com certeza estou esquecendo.

Nas últimas três semana em que escrevi minha tese, fui testado em meus limites físicos e psicológicos. A Eriça ficou do meu lado (virtualmente) todo esse tempo, segurando a minha mão, e me impedindo de cair. Kororo kara arigatou, itoshii no eriça.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Quando tratamos de sistemas que escolhem as ações a serem tomadas através do planejamento, uma preocupação importante é determinar o que fazer no caso do plano falhar durante a execução. Neste trabalho propomos uma nova técnica para recuperação e diagnóstico de falha de planos, a qual chamamos de *Sistema de Substituição*. A idéia básica é que podemos substituir uma ação qualquer por um conjunto de outras ações que cumpram os mesmos objetivos da ação original.

Essa idéia de substituição de ações já é encontrada em muitos sistemas naturais do nosso dia a dia. Animais quadrúpedes podem ser vistos frequentemente mancando com três de suas patas quando a quarta está machucada por algum motivo. Se uma pessoa está com um dente em um dos lados da boca doendo, tentará morder e mastigar alimentos com o outro lado. Se alguém estiver com as duas mãos ocupadas, tentará segurar um livro apertando-o debaixo dos seus braços, comprometendo o seu equilíbrio. Em todos esses exemplos, uma atividade realizada de uma maneira simples (andar com as quatro patas, mastigar com a boca inteira, carregar um livro com a mão), se torna impossível de ser realizada. Para poder alcançar o objetivo original, o indivíduo busca alternativas menos eficientes, trocando a ação em questão por outra equivalente.

Da mesma maneira, a idéia da substituição de ações pode ser usada para prover tolerância a falhas em um sistema computacional. Os subplanos equivalentes, ou de substituição, para cada ação são gerados a partir das redundâncias parciais entre os diferentes dispositivos de um sistema complexo. Ou seja, cada um dos dispositivos pode oferecer parte das capacidades do dispositivo original, de maneira que, quando agrupados, eles apresentam uma redundância de utilidade com esse dispositivo. Por exemplo, em um robô humanóide, uma manipulação com um braço direito pode ser substituída por uma manipulação com o braço esquerdo, acompanhada de um movimento apropriado do corpo.

Assim, a ação que será substituída é uma ação que não pode ser executada, seja por uma falha em um dos dispositivos que executam essa ação, seja por algum impedimento

decorrente do ambiente. Essa ação é então trocada por um sub-plano que não dependa desse dispositivo falho, ou que não seja bloqueado pelo ambiente. A utilização da substituição de ações para correção de planos se torna, então, uma técnica reativa, onde o sistema a cada passo "sente" a si mesmo e ao ambiente em busca de falhas ou impedimentos para as próximas ações, e substitui essas ações no plano conforme a necessidade.

Ao se substituir uma ação de um plano já montado por um subplano que não utiliza todos os componentes do sistema original, o resultado é que o novo plano, após a substituição, será menos eficiente que o original. Essa troca de eficiência pela capacidade de alcançar os objetivos é intencional, e conhecida na literatura de tolerância a falhas como *degradação graciosa* [39].

A idéia de substituição de ações deste trabalho foi inspirada no trabalho de Benso com degradação graciosa de eficiência em processadores [8]. Nesse trabalho é descrito um sistema de tolerância a falhas onde um "sistema de substituição" decompunha micro-instruções que requerem componentes defeituosos em sequências de instruções mais simples que utilizam apenas componentes não defeituosos. Por exemplo, uma instrução de multiplicação pode ser decomposta em uma ou mais instruções de soma, as quais, respectivamente, podem ser decompostas em instruções de negação e subtração, ou instruções de incrementação e XORs. Todos os relacionamentos de dependência entre as unidades, organizadas das mais complexas para as mais simples, é colocado em um diagrama de substituição, como ilustrado na figura 1.1.

Transpomos essa idéia para o ambiente mais genérico de sistemas computacionais baseados em planos. Ao fazer isso, perdemos a característica da hierarquia clara entre dispositivos do sistema, ou seja, não temos necessariamente uma ordem de dispositivos mais simples e menos propensos a falhas. Nessa situação, o grafo de substituição, para ser completo, precisa conter loops entre seus dispositivos. Isso cria a possibilidade de algumas situações críticas que serão discutidas mais à frente, como por exemplo deadlocks, onde ações A e B requerem ser substituídas uma pela outra.

Além disso, de acordo com o sistema em questão, o número de dispositivos e de relações entre eles podem aumentar drasticamente, especialmente quando se considera a atuação de mais de um dispositivo combinados. Enquanto um grupo pequeno e hierárquico de dispositivos pode ser ordenado manualmente pelo projetista do sistema para criar um grafo de substituição, os relacionamentos existentes nos sistemas que estudamos cresce facilmente a níveis em que humanos não são mais capazes de manipular [5], requerendo um processo de geração automática desses relacionamentos.

Na área de tolerância a falhas, o sistema de substituição é precedido pelas técnicas de replanejamento, e pelo planejamento condicional. As primeiras procuram criar um novo plano caso o plano em execução falhe por algum motivo. O replanejamento consegue, quando existe, buscar uma solução que atenda a todos os objetivos do plano anterior

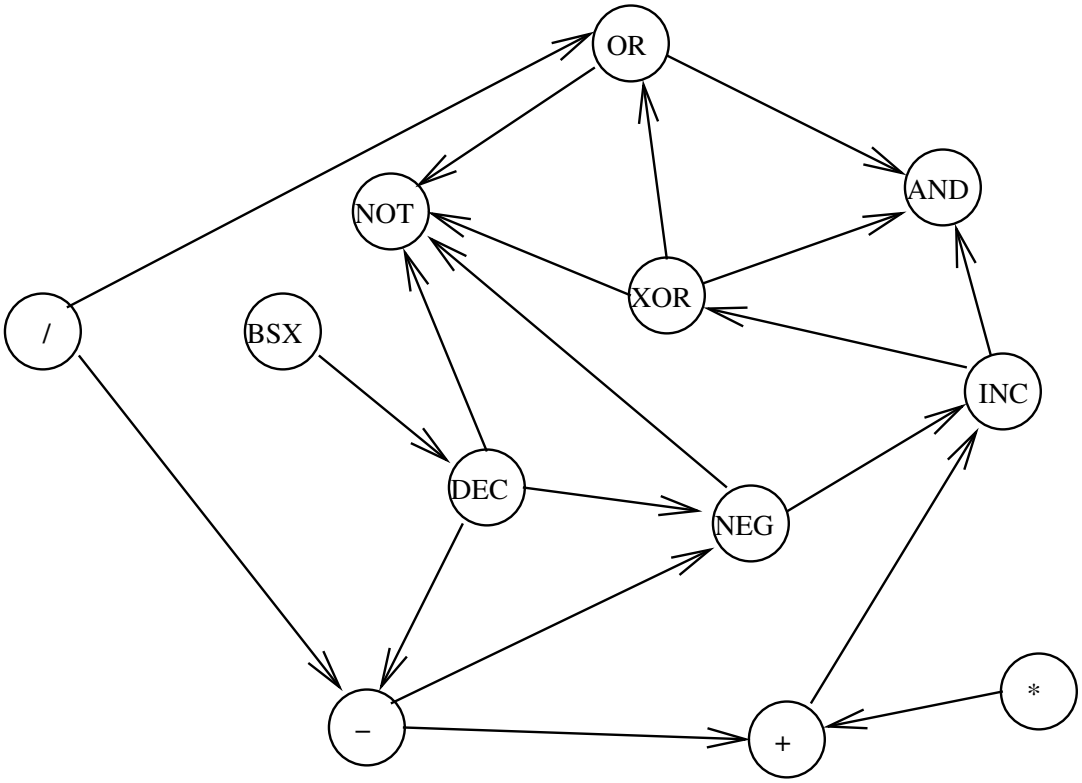

Figura 1.1: Grafo de substituição para microprocessador.

face às novas circunstâncias. Porém, os algoritmos de planejamento mais recentes ainda demoram alguns segundos para encontrar soluções que satisfaçam os objetivos de planos complexos [31].

Já no planejamento condicional, alternativas são criadas dentro de um plano para momentos nos quais o estado do sistema será incerto. Assim, nesses "pontos de quebra" o sistema verifica o seu verdadeiro estado atual, e escolhe qual alternativa de plano tomará a partir disso. O planejamento condicional não possui os problemas de custo de recálculo de plano do replanejamento. Porém, ele só é realmente útil para sistemas com um nível baixo de incerteza, pois o número de opções a ser criada pode explodir muito rapidamente para problemas complexos [34, 11].

Assim, buscamos desenvolver o sistema de substituição como um meio termo entre essas duas técnicas já existentes, com o objetivo de se integrar com o replanejamento para o desenvolvimento de sistemas robustos. Como o planejamento condicional, o sistema de substituição prepara, antes da execução, um conjunto de alternativas que possam ser utilizadas para casos de falha. Porém, diferentemente daquele, essas substituições são independentes de qualquer plano em particular e, na medida do possível, de instâncias

específicas do ambiente também.

O sistema de substituição pode, além dos usos para reparos de planos mencionados acima, também ser aplicado para auxiliar o diagnóstico de falhas em sistemas computacionais. No nosso trabalho descrevemos o *diagnóstico planejado*, uma técnica na qual executamos uma sequência de ações para isolar um dispositivo faltoso de um sistema, entre vários suspeitos. Essa técnica, apesar de barata no sentido de não querer nenhum componente de hardware extra para aumentar a capacidade de diagnóstico do sistema, requer que este pare o que estiver fazendo para "olhar para si mesmo", e isolar a falha. Propomos um modo de utilizar a substituição de ações para realizar um plano de diagnóstico sem interromper completamente o fluxo de execução do sistema.

Sistemas robóticos possuem várias características que fazem que a aplicação do sistema de substituição se encaixem naturalmente neles. Robôs humanóides, por exemplo, possuem vários dispositivos independentes que podem ser combinados para alcançar um mesmo efeito de diferentes maneiras (braços, pernas). Em robôs mais simples, a redundância pode ser distribuída entre os membros do grupo. Além disso, o path planning, tipo de planejamento normalmente utilizado para controle de robôs, é mais propício para o uso do sistema de substituição, uma vez que em geral as representações dos estados nesse problema podem ser somadas entre si, formando novos estados válidos (capítulo 4). Em junho de 2004, o robô Spirit, que está em missão em Marte, enfrentou falhas similares às que descrevemos no nosso modelo de robô simulado [22]. De uma maneira geral, aplicações de robótica no espaço requerem fortes desenvolvimentos nas áreas de automação, planejamento, auto-diagnóstico e auto-reparo para alcançarem o mesmo nível de autonomia que seus equivalentes terrestres [65].

Porém, além da aplicação em robótica sugerida, a idéia de substituição pode ser adaptada para qualquer outro sistema computacional baseado na utilização de planos. Por exemplo, os estudos em web services tem tido uma tendência recente de utilizar técnicas de planejamento para composição e execução de serviços [80]. A diversidade e heterogeneidade dos serviços web permitem planos de substituição variados e interessantes, úteis para serviços de grande escala.

Assim, a partir da idéia simples do sistema de substituição, desenvolvemos outros trabalhos relacionados à implementação e aplicação do mesmo. Um resumo desses tópicos estudados durante a realização do projeto pode ser visto na figura 1.2.

No capítulo 2, fazemos uma revisão dos conceitos relevantes para a compreensão do conteúdo desta dissertação. A seção 2.1 descreve melhor o problema do planejamento e da correção de planos (replanejamento, planejamento condicional e probabilístico). A seção 2.2 analisa técnicas de tolerância a falhas em robótica e de planejamento robótico, localizando nosso trabalho entre elas. A seção 2.3 explica o conceito de Web services, sua evolução recente, e como estes têm se utilizado cada vez mais técnicas de planejamento.

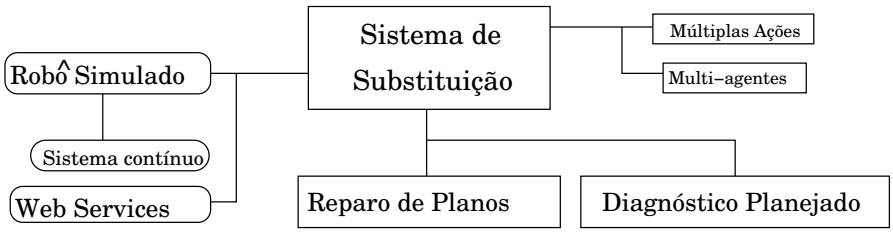

Figura 1.2: Resumo das linhas de idéias seguidas durante o projeto, e sua relação. Os balões com fonte menor indicam caminhos futuros.

No capítulo 3 descrevemos o sistema de substituição, seus componentes, e como estes podem ser utilizados para realizar o reparo de planos ou o diagnóstico planejado.

No capítulo 4 aplicamos os conceitos do capítulo 3 para plataformas robóticas. Descrevemos o robô explorador simulado que utilizamos durante nosso projeto, Dodói, e citamos as características do reparo de planos e do diagnóstico planejado nesse sistema. Depois disso, falamos sobre a aplicação de um sistema de substituição um pouco modificado para o WL-16, um robô real caminhante, comentando a diferença de implementação de um sistema discreto para um contínuo.

No capítulo 5 aplicamos os conceitos do capítulo 3 para plataformas não robóticas. é descrito um web service de transporte de cargas fictício, baseado em características comuns desse tipo de aplicação, e discutimos como implementar um sistema de substituição para ele, comparando-o com os sistemas robóticos.

No capítulo 6 discutimos caminhos futuros para o estudo de sistemas de substituição. Em particular discorremos sobre a substituição de múltiplas ações, e a representação resumida de planos de substituição.

Capítulo 2

Revisão da Literatura

Neste capítulo fazemos um breve estudo das principais áreas relacionadas com o trabalho desta tese. Citamos os trabalhos mais relevantes para um entendimento geral do problema, e como o sistema de substituição se relaciona com cada uma delas.

A primeira seção discorre rapidamente sobre o problema do planejamento, e avança para uma análise das diversas abordagens para a tolerância a falha em planos. Em seguida, analisamos a relação entre planejamento robótico, e sistemas de tolerância a falha em robôs. Finalmente, descrevemos a área de pesquisa dos web services, focando na geração automática de web services através de planejamento.

2.1 Planejamento e reparo de planos

Planejamento pode ser definido da seguinte forma: Seja um conjunto de estados S, que descrevem as possíveis configurações do mundo onde o agente (que pode ser, por exemplo, um sistema robótico) atua, e um conjunto de *ações*, A, que permitem a esse agente afetar o estado do mundo atual. Para cada ação a_i em A, temos um conjunto de *pré-condições*, que definem quais características do estado atual devem ser verdadeiras para que a ação possa ser executada, e um conjunto de *pós-condições*, que definem que mudanças a ação vai operar sobre o estado atual, transformando-o em um outro estado. Seja um objetivo G definido por um estado final s_f em S, um plano que *satisfaz* G consiste na sequência de ações $(a_0, a_1, a_2...a_n)$ que levam o agente do estado inicial s_o até esse estado final.

De uma maneira geral, o problema do planejamento é considerado computacionalmente difícil. Para vários casos interessantes, foi determinado que o planejamento é NP-Completo [29, 17], ou indecidível [18]. Para contornar esse problema, foram propostas várias técnicas para diminuir o tamanho do espaço de busca. Entre elas estão os planejadores que utilizam heurísticas independentes do domínio para reduzir o custo da busca [32, 42, 10], e os planejadores nos quais o usuário declara regras de busca que

restringirão o espaço a ser procurado [24, 74, 83].

Para evitar completamente o cálculo de planos após o início da execução da tarefa é utilizada a abordagem reativa [16], na qual o planejamento é feito completamente offline. Assim, o sistema consulta uma tabela gerada pelo planejador, de acordo com a entrada em seus sensores a cada instante, para descobrir qual a próxima ação ou sequência de ações a tomar. Abordagens mistas de sistemas reativos com planejamento tem obtido bastante sucesso na área da robótica, com os sistemas reativos lidando com ações de curto prazo, e o planejador determinando os objetivos e estratégia a longo prazo do robô [15, 14].

Atualmente, a área de planejamento tem progredido em eficiência e sofisticação dos seus algoritmos e seu potencial para aplicação a problemas reais [52]. Tais técnicas são usadas: em robôs móveis [69, 53], processos de manufatura [59], planejamento de operações espaciais e de satélites [78], jogos de "bridge" [79], gerenciamento de situações de crise (evacuação urbana) [57], entre outros.

A utilização de sistemas controlados por planejadores em problemas do mundo real leva à necessidade de capacitar os planejadores a lidar com situações de falhas. Um plano pode falhar quando mudanças em um mundo dinâmico fazem com que o estado do sistema não seja o esperado em um dado ponto da execução da tarefa, de maneira que uma ação não possa ser executada. Por exemplo, um planejador para guiar um carro pelas ruas de uma cidade encontrará uma falha no plano caso fure o pneu do carro durante a execução da tarefa, ou um acidente bloqueie uma rua pela qual o carro iria passar. Outra possível causa para falhas em um plano é quando uma ação pode resultar em vários resultados possíveis e diferentes entre si, de maneira que o planejador não tem certeza de qual será o estado do mundo após a realização dessa ação. Voltando ao exemplo do carro, a certas velocidades, virá-lo pode resultar tanto em uma mudança de direção, quanto no carro capotando e explodindo em chamas.

Para tratar o problema das falhas em planos, vemos três tipos de abordagens principais na literatura. O *Planejamento Condicional* [91, 64, 68] procura levar em conta as possíveis variações do ambiente, e criar diferentes ramos no plano principal que possam reagir de maneira efetiva para cada passo do plano. Para isso, ele inclui ações de observação, que irão determinar o estado atual do mundo antes de efetuar uma decisão. O *planejamento probabilístico* [45], embora semelhante ao primeiro, se foca em como levar uma ação com mais de um possível resultado ao resultado desejado, através de repetições ou ações auxiliares que aumentem a probabilidade deste. Indo por outro caminho, o *replanejamento* procura analisar a nova situação existente a partir de um ponto de falha, e criar um novo plano, seja do zero, seja a partir de partes do plano anterior ou outros planos.

A seguir descreveremos com mais detalhes algumas técnicas de reparos de planos das três categorias acima.

2.1.1 Planejamento Probabilístico e Condicional

Sistemas de planejamento condicional são aqueles que geram planos que tem *ações de ramificação*, isto é, ações com vários resultados possíveis [91, 64, 68]. Quando uma ação é inicialmente inserida em um plano, um de seus resultados (o resultado desejado), será ligado ao passo seguinte no plano, enquanto os outros (resultados indesejados) não.

Continuando o exemplo do carro, imagine que para atravessar um cruzamento, o sinal deve estar verde. O sistema deve executar uma ação para verificar o estado do sinal quando chegar ao cruzamento. O conhecimento de que o sinal está verde é o resultado desejado da ação de reconhecimento. Verificar que o sinal está vermelho é o resultado indesejado. O plano terá sucesso se para todas as ações de observação ocorrerem os resultados desejados. Se não, não há garantia de sucesso.

Uma maneira de melhorar tal plano é descobrir o que fazer no caso de ocorrer um resultado indesejado em alguma das ações. A isto chamamos de *Reparo Corretivo*, pois envolve descobrir que ações devem ser realizadas para corrigir a situação resultante do resultado indesejado. No exemplo acima, um reparo corretivo possível é parar o veículo, esperar alguns segundos e verificar novamente a situação do cruzamento. Para implementar reparos corretivos na prática, planejadores condicionais criam duplicatas do estado final, e tentam criar planos que alcancem os estados duplicados nos casos em que as ações de ramificação terminam em resultados indesejados.

Planejadores probabilísticos utilizam uma outra abordagem para resolver esse problema. Os planejadores condicionais assumem que os agentes não tem nenhuma informação a respeito das probabilidades dos vários resultados de uma ação, mas são capazes de observar o estado do mundo durante a execução. Já os planejadores probabilísticos, como Buridan [45], assumem exatamente o contrário: que o agente conhece as probabilidades de cada resultado possível de uma ação, mas não é capaz de observar o estado atual do ambiente. Normalmente, planejadores probabilísticos modelam ações através de um conjunto finito de tuplas $< t_i, p_{i,j}, e_{i,j} >$, onde t_i é um conjunto de *condições*, exaustivas e mutuamente exclusivas, e $p_{i,j}$ representa a probabilidade que a ação vai resultar no efeito $e_{i,j}$ se t_i for verdadeira no momento em que a ação for executada.

Por exemplo, o nosso carro pode querer fazer uma curva, mas ele pode capotar de acordo com a velocidade atual. Os resultados possíveis de "fazer curva" são "mudar de direção" e "capotar", sendo que a condição para "mudar de direção" seja "velocidade baixa". Se a velocidade não estiver baixa o suficiente, o carro pode capotar. Para evitar esse resultado, o planejador deve garantir que a probabilidade da velocidade ser baixa o suficiente seja alta. Para isso, ele pode, por exemplo, adicionar algumas ações de "reduzir a velocidade" antes da ação "fazer curva". A isto chamamos de *Reparo preventivo*, pois consiste de adicionar ações que ajudam a prevenir o resultado indesejado.

É possível também combinar as idéias desses dois tipos de planejamento. O primeiro

sistema Condicional/Probabilístico de planejamento foi o C-Buridan [27]. Apesar do C-Buridan utilizar reparo preventivo para aumentar a probabilidade de sucesso, a criação de ramificações do plano não é feita por reparo corretivo, mas de uma maneira mais indireta. Ao realizar o reparo preventivo, o planejador pode adicionar ao plano uma ação que entre em conflito com outra ação já existente. Para resolver esse conflito, o planejador ramifica o plano em duas alternativas, colocando cada ação conflitante em uma delas.

Um outro sistema que combina planejamento condicional e probabilístico é o Weaver [12]. Ele procura raciocinar sobre quais ações escolher para aumentar mais rapidamente a probabilidade de sucesso. Além disso, ele utiliza tanto o reparo preventivo como o corretivo. Diferente da maioria dos outros planejadores, ele também inclui mecanismos explícitos para lidar com eventos externos. "Conformant planning" é uma outra abordagem para planejamento com incertezas, e envolve a geração de planos que alcançam os objetivos em todos os casos possíveis, sem utilizar ações de observação [35]. O trabalho em planejadores baseados em processos de decisão Markovianos (MDP) se foca em encontrar funções de estados para ações (políticas). Para fazer isso de uma maneira eficiente, planejadores baseados em MDP utilizam programação dinâmica e técnicas de abstração [26].

2.1.2 Re-planejamento

O sistema ROGUE [38] integra um planejador com um sistema de execução para uso em um robô. Quando falhas no plano ocorrem, ROGUE pede para que o planejador crie novos operadores para circundar a falha e trazer o plano de volta ao curso. O sistema foi desenhado primariamente para lidar com falhas simples; se uma falha requer uma quantidade significativa de busca, ele pode ficar preso por um bom tempo gerando um novo plano.

No Cypress [92] o replanejamento ocorre de maneira assíncrona quando uma falha no plano é detectada. A execução é feita por um sistema reativo. Se esse sistema não conseguir resolver uma situação, ele chama o planejador para refazer o componente do plano que falhou. Como esse replanejador gera planos de ordem parcial, esses componentes substitutíveis de planos são sub-grafos selecionados com a intenção de minimizar o impacto da mudança em outras partes do plano. Enquanto isso, o sistema reativo continua executando operadores que não tenham interdependências com o sub-grafo que falhou. Não existe um limite explícito no tempo necessário para substituição.

O sistema CASPER [19] utiliza o planejamento contínuo como abordagem para o reparo de planos. No planejamento contínuo, "planejamento" e "reparo de planos" são atividades indistintas entre si. O planejador atualiza o plano continuamente para refletir mudanças no ambiente que sejam detectadas. Assim, o Casper é um representante da classe de replanejadores "anytime", caracterizada por ter um plano sempre disponível

a qualquer momento. A cada iteração, o planejador tenta resolver conflitos e alcançar objetivos. Algoritmos especialistas são utilizados para criar planos para tipos específicos de objetivos.

Um primeiro exemplo de "anytime-replanner" é o algoritmo de planejamento de Elkan [28]. A busca pelo plano é realizada de maneira tradicional, tentando encaixar ações a partir do estado tradicional até o estado inicial, de trás para frente. A diferença chave entre o método de busca deste algoritmo e o do Prolog é que Elkan utilizou busca em profundidade iterativa ao invés de busca em largura. O comportamento anytime do algoritmo é caracterizado por ele considerar que objetivos e sub-objetivos negados são verdadeiros, quando seu valor é desconhecido. Assim, ele já procura planos para essas condições, e, conforme o tempo passa, verifica a validade delas. Se os planos se demonstram não condizentes com a realidade, novos planos são gerados levando em conta o conhecimento modificado. Assim, o algoritmo encontra planos iniciais em tempo polinomial, embora esses não sejam necessariamente válidos.

Um trabalho mais recente na linha de replanejamento "anytime" é o Local Subplan Replacement [31]. Quando a falha é detectada, o planejador procura substituí-la por subplanos simples baseados nas alternativas armazenadas durante o planejamento, e ir refinando o novo plano através de substituições mais preparadas. Outra característica desse trabalho são as restrições do replanejador para garantir que o plano disponível sempre será válido. A técnica consegue reparos de planos válidos em um curto período de tempo, minimizando o tempo de processamento perdido a cada falha.

2.2 Planejamento e Tolerância a falhas em robótica

Com o desenvolvimento de técnicas em automação, robótica e computação, o uso de robôs tem crescido em diversas áreas de nossa sociedade. Além dos campos tradicionais de aplicação de robôs, como produção industrial e exploração de ambientes hostis [65], podemos encontrar robôs auxiliando médicos, e realizando operações [41]; robôs realizando serviços domésticos [67]; entre outros.

Com a variação dos usos dos robôs no dia a dia, aumenta a sua necessidade de autonomia. Apesar da grande variedade de robôs e aplicações, o conceito básico para todos é o mesmo: planejar as ações do robô de maneira a realizar uma dada tarefa.

O problema do Path Planning é encontrar uma sequência de configurações que mova um robô de uma configuração inicial até uma configuração final (objetivo), sem que ele colida com obstáculos no ambiente. No baixo nível, podemos considerar todas as ações de um robô como uma sequência de movimentos de seus atuadores, reduzindo o problema do planejamento robótico ao problema de *path planning* para esses atuadores, com as apropriadas restrições [13]. Path planners automáticos se tornam, então, essenciais para

sistemas robóticos autônomos, aumentando a eficiência e aplicabilidade de sistemas de programação off-line. Assim, nas últimas décadas, esse problema foi estudado extensivamente, e diversas abordagens foram propostas [37, 40, 47].

Uma maneira de ver o problema do path planning é considerar a configuração inicial como ponto de partida, e então tentar alcançar um mínimo (ou máximo) global localizado na configuração final. Em robótica, tais técnicas são conhecidas como métodos de campos potenciais. Mas ao contrário da otimização tradicional, o interesse na robótica não é a solução ótima em si, mas o caminho percorrido até ela.

Outra maneira de formular o problema do path planning é fazendo uma otimização sobre todos os caminhos que conectam as configurações iniciais e finais. A função objetivo deve penalizar caminhos com colisões, e dar pontos para aqueles curtos e contínuos [7]. Devido á complexidade do problema, o principal interesse é achar *algum* caminho válido. Achar uma solução ótima é muito difícil na maioria dos casos. Para lidar com esse problema da complexidade, na prática são usados métodos aproximativos. Técnicas que utilizam da aleatoriedade, como o "Probabilistic Roadmap Method" (PRM) [43, 61] , têm ganho destaque.

Uma tarefa mais difícil é coordenar múltiplos robôs se movendo e interagindo em um mesmo espaço de trabalho. A complexidade do problema cresce rapidamente com o número de robôs. Para reduzir essa complexidade, vários níveis de planejamento descentralizado são estudados [49, 76]. A idéia geral é primeiro planejar para cada robô separadamente e independentemente dos outros robôs. Então os caminhos gerados devem ser modificados e coordenados para evitar colisões e otimizar a performance global.

O ambiente também algumas vezes é desconhecido, e precisa ser explorado para que se detectem os obstáculos, que podem mudar de lugar em um ambiente dinâmico. Incertezas no posicionamento e nos sensores são outros aspectos que precisam ser considerados na prática. Todos esses pontos devem ser levados em consideração para a elaboração de um sistema para aplicação em robôs no mundo real [48].

Por outro lado, quando estudamos o trabalho já desenvolvido na área de tolerância a falhas em robótica, não verificamos uma integração entre as soluções de planejamento ("alto nível") e as de controle ("baixo nível"). Os esforços de uma área em geral ignoram os avanços da outra.

Uma corrente bastante popular na tolerância a falhas robótica busca utilizar de desvios de valores esperados em propriosensores para encontrar indicativos de erros, numa abordagem reativa ao problema do diagnóstico. Essa linha, que começou com comparações simples entre níveis medidos e esperados para valores de propriosensores [89], já está bastante desenvolvida. Tópicos recentes incluem o uso de redes neurais para diferenciar ruídos de leituras de erros [86, 85] e a Redundância Analítica [50, 90], que consiste de técnicas matemáticas para gerar vários testes comparativos a partir de poucos propriosensores, de

maneira a aumentar a capacidade de detecção de erro do sistema.

Os trabalhos dessa linha, porém, trabalham com soluções locais a certos atuadores de um sistema robótico e, em geral, ao desativar um dispositivo que tenha sido diagnosticado como faltoso, não dão uma sugestão melhor do que "O planejador se encarrega a partir daqui" para o que fazer depois disso. Ainda assim, eles apresentam instrumentos muito valiosos para garantir a detecção de erros, em baixo nível, de maneira rápida o suficiente para que o sistema robótico possa tomar uma atitude imediata (o que, aliás é uma das grandes preocupações desses trabalhos).

Outro tipo de abordagem para o problema da tolerância a falha em robôs vem pelos sistemas robóticos auto-configuráveis [44]. Esses sistemas são compostos de vários robôs simples, que tentam se comportar como sistemas biológicos multi-celulares. Assim, há um grande grau de redundância no sistema que pode ser usado para aumentar o nível de tolerância a falhas. Existem também propostas para capacitar tais sistemas a mudar suas configurações para se adaptar a novas situações no ambiente (mas novamente em relação apenas à ação imediata, e não a um possível plano em execução) [60, 88]. Seguindo essa direção, algoritmos evolutivos também já foram utilizados para desenvolver hardware tolerante a falhas.

No ALLIANCE [62] um sistema para tolerância a falhas robótico é apresentado, onde a redundância inerente a sistemas multi-robôs é utilizada para prover substituições para robôs faltosos em uma missão distribuída. Nesta abordagem, a substituição é feita centrada em erros no plano, causados por falhas nos robôs, e esses erros são tratados usando uma combinação de replanejamento e características de baixo nível do sistema. De uma maneira mais detalhada, os robôs que falham irão parar de realizar as atividades atribuídas a eles. Com o tempo o sistema vai notar que uma atividade não está sendo completada da maneira esperada, e outros robôs tentarão abandonar suas atividades atuais para atender a esta falha, criando um equilíbrio no sistema.

Da mesma maneira, o nosso trabalho busca usar a capacidade de tolerância a falhas já disponíveis no sistema robótico (diagnóstico de atuadores, redundância direta - através de atuadores duplicados - e indireta - através de dispositivos parcialmente equivalentes) para prover uma capacidade de reparo de planos reativa.

2.3 Web services e composição de webservices

Um Serviço Web (SW), de acordo com o grupo de trabalho da W3C sobre arquitetura de Serviços Web [6], é uma aplicação de software identificada por uma URI (Uniform Resource Identifier), cuja descrição e transporte utilizam padrões da Internet, isto é, tecnologias baseadas principalmente em XML [2]. Estes serviços podem ser vistos como componentes de software. Portanto, para permitir seu reuso, um componente deve: ter

uma descrição que permita a outros componentes entender a sua funcionalidade e poder usá-la; permitir a outros componentes a sua localização para que seja usado quando necessário; e ser facilmente invocado quando outro componente precise de suas funções. A busca de interoperabilidade faz com que serviços usem tecnologias baseadas em XML. As descrições independentes da linguagem e da plataforma permitem a integração fácil de sistemas heterogêneos.

Os principais elementos usados na caracterização e descrição de serviços para garantir reusabilidade e interoperabilidade são o WSDL, o UDDI e o SOAP. O WSDL ("Web Services Description Language") [20], uma linguagem baseada em XML, é um padrão utilizado pelos web services para descrever suas funcionalidades. O UDDI ("Universal Description, Discovery and Integration") [77], é um padrão desenvolvido pelas indústrias para prover uma maneira de descobrir serviços na Web. O SOAP ("Simple Object Access Protocol") [36], é um protocolo baseado em XML de passagem de mensagens que permite a um SW a chamada de funções de outros SW.

Atualmente o espectro de produtos disponibilizados via SW vai desde sistemas de BDs e servidores de aplicações até aplicações customizadas. Os SW têm se convertido em um aspecto integral da arquitetura moderna de sistemas [25]. Segundo Fensel e Bussler [30], os serviços web fizeram com que a web se transformasse de uma coleção de informação para um dispositivo computacional distribuído.

Um passo importante no desenvolvimento de aplicações baseadas em SW é a habilidade de selecionar e integrar em tempo de execução e de forma eficiente e efetiva serviços na web heterogêneos e de diferentes empresas. Isto tem conduzido à criação de linguagens de composição de SW tais como: BPEL4WS ("Business Process Execution Language for Web Services") [4] ou WSCI ("Web Service Choreography Interface"), entre outras. O problema destas propostas é que a definição de novos processos que interagem com outros já existentes deve ser feita de forma manual, o que pode ser uma tarefa muito demorada e que pode gerar muitos erros.

2.3.1 Serviços Web Semânticos e OWL-S (H)

Uma abordagem utilizada para resolver esse problema é a reorganização das informações disponibilizadas pelos web services de uma maneira designada como *web semântica*. A idéia da Web Semântica é permitir que os dados e serviços contidos nela possam ser interpretados automaticamente pelos agentes de software e, portanto, sem os problemas de ambiguidade [9]. Sendo assim, é necessário estender a semântica das linguagens de descrição existentes (WSDL, UDDI, BPEL4WS, entre outras) ou usar novas linguagens que permitam descrever as capacidades e conteúdo dos SW de forma interpretável pelos computadores. Tem sido feitas algumas tentativas para aplicar semântica à tecnologia de

SW, entre elas podemos citar o uso de DAML-S, hoje chamado OWL-S [23]; e o trabalho sendo desenvolvido pelo grupo da WSMF ("Web Service Modelling Framework") [30], que pretendem desenvolver uma plataforma totalmente flexível e escalável para comércio eletrônico baseada em SW.

OWL-S, que é a proposta mais conhecida na literatura, pretende prover a semântica necessária para facilitar a construção de serviços web semânticos (SWS). OWL-S fornece uma ontologia abstrata (alto nível) para SWS. Conceitos mais específicos não são incluídos nessa ontologia. Ela oferece uma estrutura na qual ontologias mais específicas podem ser construídas.

OWL-S tem por objetivo suportar as seguintes tarefas [55, 23, 25]:

Descoberta automática de SW visa a localização automática (por agentes de software) de SW que oferecem um serviço específico que cumpre certas restrições. Por exemplo, achar um serviço que venda passagens entre Campinas e Rio de janeiro e que aceite cartões de crédito "Diners";

Chamada automática de SW visa a execução automática de um SW, previamente identificado, por um programa ou agente. Através de APIs declarativos, o programa ou agente será capaz de entender quais entradas são requeridas por um serviço, quais são suas saídas e como executá-lo.

Composição e interoperação automática de SW visa a seleção, composição e interoperação automática de SW para desenvolver alguma tarefa, dada uma descrição em alto nível de um objetivo. Para alcançar o objetivo, é necessário dividir automaticamente a tarefa em componentes, selecionar os SW correspondentes, compô-los e a seguir fazer a sua interoperação. Por exemplo, o usuário pode desejar o planejamento para uma viagem de apresentação de um trabalho em uma conferência.

Monitoração automática da execução de SW visa o acompanhamento da execução do SW, para saber seu status e se algum problema não antecipado ocorreu. Por exemplo, qual é o status do meu planejamento de viagem? Já foi reservado o hotel?

A ontologia abstrata de serviços OWL-S é formada por três classes chave. ServiceProfile responde à pergunta "O que o serviço faz?"; sua utilidade é anunciar e descobrir serviços através da descrição de suas funcionalidades. ServiceModel responde à pergunta "Como trabalha o serviço?"e para isso dá uma descrição detalhada da operação do serviço. ServiceGrounding responde à pergunta "Como o serviço é usado?"e para isso especifica os detalhes de como acessar o serviço (enviar e receber mensagens para ou a partir do serviço), especificamente os formatos das mensagens e protocolos, transporte e endereçamento. Esta última classe permite o mapeamento das descrições abstratas dos

elementos que são requeridos para interagir com o serviço para uma especificação mais concreta que possa ser viabilizada através de uma mensagem. Em geral a linguagem de especificação é WSDL.

2.3.2 Composição Automática de web services

Como foi dito inicialmente, o problema da interoperabilidade e sincronização de SW é importante, e várias de propostas de solução têm sido desenvolvidas. A classe Service-Model do OWL-S é uma proposta para a representação de SW compostos. O problema destas propostas é que o fluxo de um processo e as ligações entre os serviços precisam ser conhecidos a priori e especificados manualmente.

Fileto [33] menciona alguns desafios destas tecnologias:

- Reduzir a quantidade de programação necessária para a interconexão de SW, utilizando, por exemplo, linguagens declarativas. Este aspecto é tratado pelo OWL-S que oferece um bom suporte semântico para o raciocínio automático, como, por exemplo, a implementação de geradores de planos [52].
- Prover flexibilidade para estabelecer interações em tempo de execução entre um número crescente de SW.
- Projetar mecanismos para o controle de transações descentralizado e flexível para a execução de processos cooperativos na Web.

A composição automática de web services é uma tendência recente para tratar os desafios e problemas mencionados [56]. Esta inclui a seleção e interoperação automática de web services. A composição automática terá um papel importante na viabilização da web semântica [9, 55]. A idéia é que os usuários especifiquem o "que"eles desejam da composição (e.g., metas ou ações a serem desenvolvidas em um alto nível) e o sistema encarrega-se do *como*, isto é, os SW a serem utilizados, como interagir com esses serviços, etc. O processo de compor os SW deve ser transparente aos usuários e as descrições detalhadas dos serviços compostos devem ser geradas automaticamente pelo sistema a partir das especificações dos usuários.

Vários tipos de solução têm sido propostos para o problema de composição automática de serviços, entre os quais podemos mencionar: as soluções baseadas em composição de workflows e as baseadas em planejamento [71]. Outros métodos são mencionados por Hull e Su [70].

2.3.3 Utilizando planejamento para composição automática de web services

Recentemente, esforços de pesquisa tentam tratar o problema de composição automática de web services através de planejamento [1, 3]. Em geral um problema de planejamento pode ser concebido da seguinte maneira: dados uma maneira de descrever o mundo, um estado inicial do mundo S_0, uma descrição de uma meta G, e um conjunto de ações A que mudam o mundo, devemos achar uma composição de ações que transformam o estado inicial S_0 em um estado S_f que satisfaz a meta. Em termos de web services, S_0 e G podem ser o estado inicial e a meta especificados pelos usuários que solicitam o serviço; A é o conjunto de serviços disponíveis.

OWL-S [23] é a única linguagem para web services que declara especificamente o propósito de ter conexão com a área de planejamento. Da mesma forma que os sistemas planejadores em IA, as mudanças de estado produzidas pela execução dos serviços são especificadas através das pré-condições e efeitos no ServiceProfile em OWL-S. Portanto, muitas das propostas definidas para planejamento usam OWL-S, embora existam algumas que usam WSDL ou suas próprias linguagens. Seguem algumas dessas propostas.

McIlraith e Son adaptam e estendem a linguagem Golog [51] para a construção automática de web services. Golog é baseado no cálculo de situações [72] e suporta a especificação e execução de ações complexas em sistemas dinâmicos. Os autores tratam o problema de composição de SW através de procedimentos genéricos ("Plan templates"predefinidos) e restrições personalizadas do usuário, que podem ser definidas por uma variedade de usuários sob condições variantes. Os web services são representados como ações, que podem ser primitivas ou complexas. As ações primitivas podem ser ações que mudam o estado do mundo ou de coleta de informação que mudam o estado de conhecimento do agente. As construções procedurais (if-then-else, while, etc.) são usadas para formar os serviços complexos.

Peer [63] utiliza PDDL como linguagem de representação de planos em web services. McDermott [54] também usa PDDL para a especificação dos planos, porém a estende para introduzir um novo tipo de conhecimento chamado valor de uma ação, que serve para representar a passagem de informação entre passos dos planos. Esta característica facilita distinguir a transformação de informação e a mudança de estados produzida pela execução do serviço. O forte interesse da comunidade de Planejamento em IA pela composição de web services pode ser explicado pela similaridade entre as notações usadas em OWL-S e PDDL. Portanto, quando se usa planejamento na composição de web services pode-se transformar as descrições de OWL-S para PDDL, permitindo o uso de diferentes tipos de planejadores já existentes para sintetizar os planos.

Medjahed et al. [56] apresentam uma técnica para gerar serviços compostos a partir

de uma descrição declarativa em alto nível. A composição consiste de quatro fases: especificação, casamento, seleção e geração. A principal contribuição deste método é o uso de regras de composição, que podem ser usadas como guias para outros métodos baseados em planejamento. Outro método baseado em regras é o SWORD [66]. Uma de suas características é que não usa nem WSDL nem OWL-S para especificar os web services, mas sim o modelo Entidade-Relacionamento (E-R). Um serviço é modelado pelas suas pré-condições e pós-condições. Elas são especificadas em um modelo do mundo que consiste de entidades e relacionamentos entre as entidades. Um web service é representado na forma de uma regra de Horn que descreve as pós-condições que são alcançadas se as pré-condições são verdadeiras. Para criar um serviço composto, o usuário unicamente precisa especificar os estados inicial e final do serviço. A seguir a geração do plano pode ser feita por um sistema especialista baseado em regras.

Wu et al. [93] usam o planejador SHOP2 ("Simple Hierarchical Ordered Planner 2") [58] para a composição automática de web services. As entradas para o planejador são especificadas em OWL-S. SHOP2 é um planejador baseado numa rede hierárquica de tarefas (HTN) [73]. Os autores acreditam que o conceito de decomposição automática de tarefas usado no planejamento HTN é muito similar ao conceito de decomposição de processo compostos usado na ontologia OWL-S. Um dos problemas desta proposta é o fato de não poder gerar planos com estruturas de controle concorrentes (i.e. Split e Split+Join). Esse tipo de estruturas é útil na composição de SW, sendo parte da estrutura dos processos compostos de OWL-S.

O trabalho de Traverso e Pistore [87] apresenta uma técnica para a composição automática de web services descritos em OWL-S, que permite a geração automática de processos executáveis (em BPEL4WS). As metas que especificam o serviço a ser automaticamente gerado são representadas na linguagem Eagle [46], uma linguagem com uma semântica clara que pode expressar solicitações complexas. Os autores usam uma abordagem de planejamento baseada em técnicas de verificação simbólica de modelos [21], que tem apresentado bons resultados práticos para o problema de planejamento com ações não determinísticas, observações parciais (o ambiente não é totalmente conhecido), metas e domínios complexos (espaço de estados muito grande). Um dos problemas desta proposta é o fato de não explorar os aspectos hierárquicos e taxonômicos existentes em OWS-L, como é feito em SHOP2.

Capítulo 3

Metodologia Proposta

Neste capítulo descrevemos a técnica de substituição; como ela se compõe, e seu modo de funcionamento. Na primeira seção descrevemos a formulação geral do sistema de substituição, e os três componentes de software necessários em sua composição. Na segunda seção utilizamos essa estrutura para reparar falhas de um plano. Finalmente, apresentamos como utilizar a mesma estrutura para realizar diagnóstico planejado durante a execução da tarefa.

Procuramos descrever a técnica de uma maneira mais genérica, deixando problemas específicos de certas implementações para os próximos capítulos. Porém é necessário especificar algumas características do sistema e notações para que a descrição da técnica seja mais compreensível.

Chamamos de *sistema*, então, o dispositivo computacional no qual será aplicada a técnica de substituição. Por exemplo um robô, ou uma empresa que utiliza de web services para disponibilizar os seus serviços. Chamamos de *Ambiente* todo o mundo externo ao sistema, sobre o qual aquele age. O sistema age no ambiente através de *Ações*. Cada ação é definida pelas suas pré-condições e pós condições.

Um tipo especial de pré-condição que iremos definir para o sistema é a *dependência de um dispositivo*. Cada ação do sistema dependerá de um ou mais dispositivos deste. Esses dispositivos são partes do sistema necessárias para execução da ação. Por exemplo, as rodas de um robô móvel são os dispositivos dos quais dependem as ações de movimentação de um robô. Uma ação pode depender de um ou mais dispositivos diferentes, e várias ações podem depender dos mesmos dispositivos.

Definimos também um tipo de ação que chamaremos de *ação de diagnóstico*. Essa ação serve para nos indicar quais dispositivos, se algum, apresentam falhas. Cada sistema pode apresentar ou não ações de diagnóstico, e quando executadas, elas tem uma chance de detectar corretamente o estado do dispositivo, que varia de sistema para sistema. As ações de diagnóstico podem ser enfileiradas junto com as outras ações do plano, ou realizadas em

paralelo a estas. Finalmente, dividimos as falhas que podem ocorrer em nossos planos em *internas* e *externas*. As falhas internas são causadas por falhas nos dispositivos. Quando um dispositivo falha, consquentemente todas as ações que dependem daquele dispositivo apresentarão erro. As falhas externas são as falhas normalmente esperadas em um sistema de planejamento, como falta de pré condições devido ao ambiente, e resultados inesperados das ações.

3.1 Sistema de substituição

Seja A o conjunto de ações de um sistema, de maneira que, para qualquer plano P desse sistema, todas as ações de P pertencem a A. Chamamos de uma substituição do subplano p' de P um segundo subplano p'', diferente de p', tal que se substituirmos toda ocorrência de p' por p'' em P, não haverá mudança nenhuma no resultado final do plano.

Suponha-se que neste sistema existe um conjunto de dispositivos, de maneira que cada ação existente em A dependa de um ou mais desses dispositivos. Da mesma maneira, cada plano (combinação de ações), depende do conjunto de todos os dispositivos dos quais cada ação pertencente ao plano dependa. Caso haja uma falha de um dispositivo, qualquer ação que dependa dele causará um erro ao ser executada. Da mesma maneira, qualquer plano que possua uma dessas ações falhará.

Para utilizarmos os planos de substituição como uma técnica de tolerância a falhas, procedemos da seguinte maneira. Quando uma falha é detectada em um dos componentes do sistema, marcamos esse componente numa lista de componentes falhos. Caso apareça, durante a execução de um plano, uma ação dependente de algum componente dessa lista, substituímos essa ação por um subplano equivalente que não dependa desse componente falho. E o plano continua em execução.

Vale dizer também que as substituições descritas aqui não precisam ocorrer unicamente no caso de uma falha. Da mesma maneira que um dispositivo de um sistema pode estar indisponível devido a uma falha, ele pode também estar ocupado com alguma outra atividade. Por exemplo, um robô com dois braços pode estar com um deles ocupado por uma ação anterior que não estava no plano original. Será necessário então substituir a ação atual, que utiliza o braço ocupado, por uma que utilize o desocupado. Para efeitos de simplicidade, mencionaremos apenas os casos de falha neste texto.

Do ponto de vista do planejador, não existe nenhuma diferença entre erros no plano causados por falhas nos dispositivos do sistema conforme especificamos, e erros causados por mudanças no ambiente, ou por resultados não determinísticos de ações. Poderia-se, então, preparar-se substituições para todo e qualquer tipo de erro, ao invés de apenas para os erros de dispositivos. A nossa divisão, porém, tem dois objetivos muito importantes.

O primeiro deles é isolar um conjunto de falhas possíveis que permanecerá o mesmo

independentemente da tarefa que for executada pelo sistema. Assim, o mesmo sistema, com a mesma tabela de substituição, ainda será valido para vários ambientes diferentes. Por exemplo, um robô explorador utilizará o mesmo sistema de substituição ao navegar por Marte ou pela cozinha de uma casa.

O segundo é que o conjunto de falhas relacionadas a dispositivos do próprio sistema em geral é mais fácil de ser enumerado do que a quantidade possivelmente inumerada de falhas relacionadas ao ambiente, na maioria das aplicações no mundo real. Existe já um esforço de pesquisa na área de detecção de falhas robóticas, como sugerido no capítulo 2 que pode ser utilizada para detectar falhas diversas.

Assim, limitando o escopo do sistema de substituição a falhas originadas dentro do próprio sistema, tornamos a aplicação da técnica mais realista e útil.

A figura 3.1 mostra um exemplo simples que ilustra bem o mecanismo do sistema de substituição. Os dispositivos 1, 2 e 3 indicados poderiam ser rodas independentes em um triciclo. Num sistema real encontraremos um número maior de componentes e ações. E, para cada ação, em geral teremos bem mais substituições possíveis.

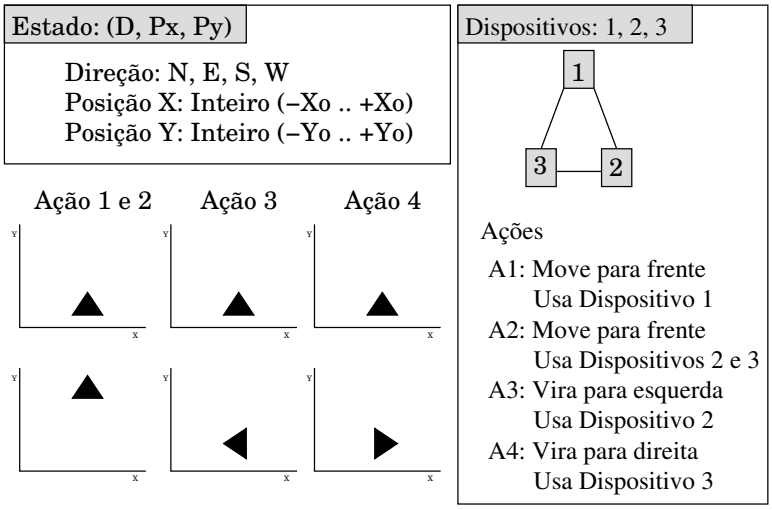

Figura 3.1: Exemplo de um sistema para uso do sistema de substituição. Aqui, $A1$ e $A2$ são substituições uma da outra. $A3$ é substituível pelo subplano $A4$,$A4$,$A4$. Da mesma maneira, $A4$ é substituível pelo subplano $A3$,$A3$,$A3$. Assim, a falha de um componente não impedirá a realização de qualquer plano neste exemplo. A falha de dois componentes torna todos os planos inviáveis.

Em linhas gerais, o mesmo princípio do exemplo simples se aplicará a sistemas mais complexos. Na prática, para implementar técnicas de substituição em sistemas reais, simulados ou não, precisaremos nos preocupar com a existência de três componentes: O sistema de diagnóstico, a tabela de substituição, e o método que utilizaremos para

escolher qual ação de substituição será utilizada. A seguir, vamos detalhar cada um desses componentes.

3.1.1 Sistema de diagnóstico

Para podermos substituir corretamente as ações de um plano, primeiro precisamos saber que existe uma falha que impedirá uma ação de ter sucesso. Assim, para cada ação que for realizada, é preciso primeiro verificar se não existe uma falha nos dispositivos necessários para aquela ação. Se existir, então a ação deve ser substituída por um outro subplano.

A natureza específica do sistema de diagnóstico varia de acordo com a aplicação. Deve haver alguma forma de segurança que avisará ao sistema sobre falhas em dispositivos antes que estas afetem o resultado de uma ação do plano. Na ausência de auto-sensores ou outras formas de auto-diagnóstico, o sistema de diagnóstico planejado, descrito mais à frente, poderia também ser utilizado. Qualquer que seja a forma desse sistema de diagnóstico, ele deve ser capaz de prover o sistema de substituição com a resposta à seguinte pergunta: "dado o estado atual dos dispositivos da aplicação, a ação a_i é válida?"

Além disso, é importante que o sistema de diagnóstico possa detectar falhas nos componentes antes que estas gerem erros no plano. A ocorrência de um erro no plano pode ser um problema para o sistema de substituição, que é preparado apenas para substituir ações que não podem ser realizadas, e não para recuperar o sistema de um estado inesperado. Se a falha em uma ação não alterou de maneira significativa o estado do sistema (como se não houvesse sido realizada ação nenhuma), então podemos simplesmente repetir a ação falha, substituindo-a de maneira adequada. Se, porém, o resultado da falha for um outro estado, então é necessário utilizar alguma técnica de replanejamento para recuperar o plano de maneira a alcançar o objetivo original.

3.1.2 Tabela de Substituição

O segundo componente necessário em um sistema de substituição é a tabela de substituição, que é a estrutura de dados que abrigará as informações a respeito de todos os sub-planos de substituição disponíveis. Ao montar a tabela de substituição, temos que ter três questões em mente: Que tipo de informações armazenar sobre os sub-planos de substituição, como indexá-los e, mais importante, como escolher *quais* subplanos armazenar na tabela.

Uma entrada na tabela de substituição consiste do subplano de substituição (listando todas as suas ações), as pré-condições e pós condições para esse subplano, e de quais dispositivos ele depende. O subplano pode consistir de um número arbitrário de ações. Mas, de forma geral, subplanos longos são menos desejáveis, pois tem requisitos mais severos de pré-condições.

Para recuperar essas informações da tabela, indexamos as entradas pelas suas pós condições. Assim, quando uma ação não pode ser executada, procuramos por um subplano que alcance o mesmo resultado. Uma escolha inicial, mais óbvia, seria indexar a tabela pela ação a ser substituída, por substituição. Porém, em sistemas com alto nível de redundância, podemos ter um grupo de ações com as mesmas pós-condições, de maneira que temos um mesmo conjunto de subplanos de substituição para essas ações (e, em geral, elas também servem de substituições entre si).

Além de indexadas pelas pós condições resultantes, as ações de substituição têm que estar indexadas pelas suas pré-condições necessárias também. é necessária, pelo menos, a informação sobre de que dispositivos depende aquela substituição em particular, embora em geral informações sobre as condições do ambiente também sejam úteis, como explicado mais à frente.

Pode-se argumentar que em sistemas não determinísticos, onde uma ação pode ter vários resultados possíveis, indexar por pós condição pode não armazenar as mesmas informações que o índice por ação. Por exemplo, se uma ação hipotética tem resultados a e b diferentes, como escolher qual substituição utilizar, se indexarmos por pós condições? De acordo com a literatura (listada no capitulo 2), nesses casos em geral temos um resultado desejado e vários indesejados, de maneira que podemos escolher a substituição de acordo com o resultado desejado apenas.

Combinando então as informações sobre as pós-condições esperadas com as sobre os dispositivos atualmente em falha, obtemos uma lista de planos de substituição apropriados para o uso. Em uma tabela de substituições ideal, existirá exatamente uma substituição para cada ação para cada situação possível de erro. Na prática, nem todas as combinações de erro têm substituições possíveis para um dado sistema. Entre aquelas que tem, temos outros fatores que podem complicar o trabalho de escolha de uma tabela de substituição. Por exemplo, as condições do ambiente podem fazer com que certas substituições não estejam disponíveis em certos momentos, devido a restrições de pré-condições (como ilustrado na figura figura 3.2). Para essas situações, é útil ter múltiplas substituições disponíveis para um par de pré/pós condições.

Em qualquer sistema, o número de possíveis substituições para uma dada ação é grande demais para que uma pessoa possa escolher todo o seu conteúdo [5]. Não apenas existe uma grande quantidade de planos de substituição possíveis para cada ação, como também muitos deles são redundantes (subplanos com exatamente as mesmas pré e pós condições), ou inúteis (subplanos que dependem dos mesmos dispositivos que a ação original, por exemplo). Por isso, é importante que a tabela de substituição seja gerada automaticamente.

Podemos começar a gerar a tabela de substituição realizando uma busca pelo espaço de ações. Para cada subplano que encontrarmos, comparamos esse subplano com as

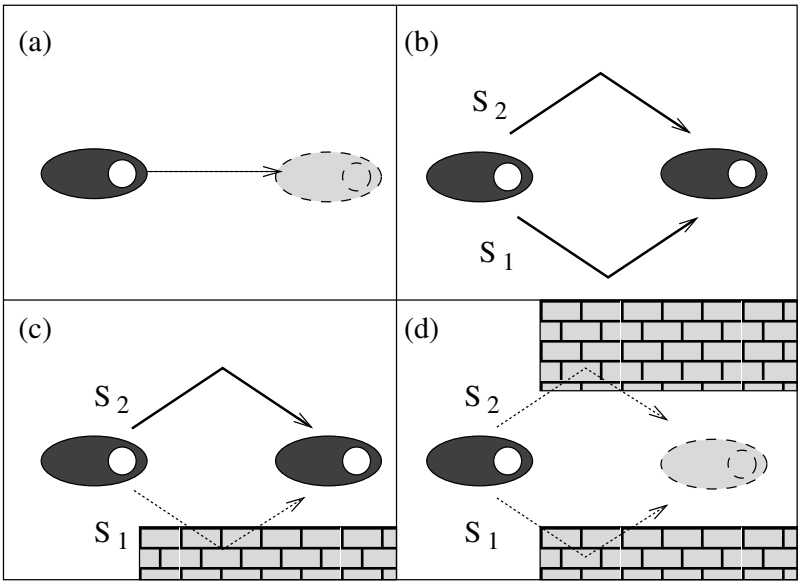

Figura 3.2: Uma falha impede que a ação em (a) seja realizada. S_1 e S_2 são substituições possíveis. Em (c), a situação do ambiente impede S_1 de ser executada.

ações originais, para verificar se ele é equivalente a alguma delas (e, logo, um plano de substituição). Para orientar essa busca, que em essência é uma busca em profundidade com limite, utilizamos uma máquina de estados que descreve a situação do sistema em um ambiente limitado. Assim, temos um estado inicial, do qual saem os estados relacionados a cada ação. Durante a busca, quando uma sequência de ações voltar a um desses estados de "primeira geração", podemos listá-la como uma substituição relacionada à pós condição das ações que geraram aquele estado. Essa técnica é uma aplicação invertida do trabalho descrito em [84], onde é apresentado o uso de uma máquina de estado para evitar laços em buscas em profundidade. Na nossa geração de ações de substituição, utilizamos a mesma técnica para *encontrar* esses laços em nossa busca em profundidade.

Gerando a tabela dessa maneira, porém, não teremos a informação de que substituição é útil para que falha. Para obtermos essa informação ainda na fase da busca por substituições, dividiremos a busca em "estágios". A cada estágio da busca, eliminamos alguns dos dispositivos disponíveis no sistema e tentamos realizar a busca por substituições sem esse dispositivo. As substituições encontradas dessa maneira são então identificadas para a falha dos dispositivos eliminados naquele estágio. Eventualmente, a busca não pode ser realizada para a combinação de certas falhas (um exemplo extremo: todos os dispositivos), essas situações também devem ser identificadas, para que o sistema saiba que tem que buscar alguma outra solução (por exemplo, entrar em contato com um operador humano).

A divisão da busca em estágios de falha, porém, pode levar a um grande número de substituições repetidas para um sistema com um grau de redundância alto. Por exemplo, duas ações equivalentes entre si, a_1 e a_2, sendo que a_1 utiliza dois componentes não utilizados por a_2. A ação a_2 será citada como substituta de a_1 três vezes: quando o primeiro componente falhar, quando o segundo falhar, e quando ambos falharem simultaneamente. Deve-se fazer uma busca entre as substituições resultantes para se eliminar essas repetições.

Outra fonte de repetições na tabela de substituição são as "ações inúteis". Um exemplo simples de ação inútil é quando temos uma ação que afeta uma pós condição não relacionada (direta ou indiretamente) com a substituição em questão (por exemplo, em um robô formiga, soltar feromônios quando a substituição em questão afeta a movimentação do indivíduo). A ação inútil será realizada no início do subplano de substituição, e desfeita antes do final do plano, mas depois de algumas ações, para não ser detectada como um loop pelo sistema de busca. Assim, temos um segundo plano que é idêntico ao primeiro em qualquer situação relevante, mas é listado como um plano diferente.

Uma situação mais sutil de ação inútil acontece quando temos dentro do subplano ações que afetam a substituição em questão, mas não adicionam nada ao sistema em relação a substituições já existentes. Cada substituição extra pode adicionar valor ao sistema ao prover uma alternativa para um conjunto de falhas ainda não abordado, ou ao prover uma alternativa que utilize recursos do ambiente diferentes das substituições já existentes. Para detectar se uma alternativa oferece variações em termos de recursos do ambiente, testamos ela contra outras substituições já existentes em algumas situações críticas. Os casos específicos vão variar com o sistema em que é implementada a substituição.

Esses três fatores, conteúdo, índice, e geração automática, têm que manter em equilíbrio o tamanho da tabela de substituição. Um número muito pequeno de substituições disponíveis para cada ação pode resultar na falta de uma substituição apropriada para uma determinada combinação de falhas e pré-condições. Por outro lado, um número excessivo de entradas na tabela vai levar o sistema a ficar muito lento, tanto na recuperação das informações, quanto na hora da escolha entre as opções disponíveis.

Em sistemas baseados em software, como Web Services e criaturas simuladas, o número de substituições de uma ação normalmente é limitado, e a combinação *ações* $\times$ *falhas possíveis* não é muito grande (no nosso robô simulado, por exemplo, é por volta de 2×10^4). Existem também casos mais complexos. Por exemplo, no robô bípede WL16 (descrito com mais detalhes na seção 4.2), a combinação de falhas diferentes possíveis para um passo é da ordem de 10^{12}. Essa aplicação é bem diferente dos casos tradicionais de planejamento, mas ilustra o fato de que a geração e manutenção da tabela de substituição deve ser calculada com cuidado.

3.1.3 Seleção da Substituição

Por causa da necessidade de adaptar a substituição ao contexto do robô, normalmente temos mais de uma substituição para cada par de pós-condição/pré-condição. Para sistemas mais complexos, podemos ter dezenas de opções. Assim, o terceiro componente necessário para a implementação de um sistema de substituição é o algoritmo que decidirá qual dos subplanos é mais adequado para uma dada substituição em um dado momento da execução da tarefa.

Em um sistema ideal, teríamos a qualquer momento uma única escolha, óbvia e ótima, para realizar a substituição. Porém, em geral temos várias complicações que podem impossibilitar o uso de uma substituição em particular: Um ambiente dinâmico, falhas nos dispositivos que realizam a substituição, laços, e a seleção de múltiplas ações, por exemplo.

Poderíamos definir um conjunto complicado de regras para buscar em uma base de dados grande o suficiente, um subplano que substitua perfeitamente uma dada ação para qualquer situação. Porém, esse conjunto de regras provavelmente seria tão complicado que demorará muito tempo para ser processado, fazendo com que valha mais a pena replanejar todas as ações para encontrar uma sequência ótima para o novo estado do ambiente. Assim, uma preocupação importante para o algoritmo que escolha de ações é que ele seja simples e rápido, mesmo que isso signifique fazer algumas escolhas sub-ótimas.

Em geral, a geração das substituições se encarrega de gerar subplanos adequados para os principais casos de falhas no ambiente, como visto anteriormente. Se o ambiente for dinâmico ou não completamente conhecido, uma solução válida pelas pré-condições na hora da substituição pode falhar durante a execução.

Exceto nos modelos mais restritos e controlados, é impossível fazer uma lista de subplanos capaz de levar em conta todas as possíveis situações de erro devido ao ambiente. Porém, de uma maneira geral, quando houver múltiplas escolhas de substituição disponíveis para uma determinada ação, procuraremos escolher aquela que assume a menor quantidade de coisas a respeito do ambiente. O que isso significa, depende da aplicação na qual está sendo implementado o sistema de substituição.

Por exemplo, para o nosso robô simulado, uma das preocupações é evitar as diferentes configurações que levam o robô a bater nas paredes. Enquanto normalmente a posição das paredes é conhecida, de forma que substituições adequadas podem ser escolhidas pelas pré-condições, quando o conhecimento do mundo é incompleto, pode haver uma parede da qual o robô ainda não sabe. Para evitar colisões, utilizamos uma política de, entre duas substituições equivalentes, escolhemos aquela com o menor desvio do segmento de reta entre o ponto inicial e final do movimento.

Outra complicação pode ocorrer quando mais de um componente falha ao mesmo tempo. Ao analisar os subplanos disponíveis, pode-se chegar à conclusão de que nenhum

deles é adequado, quando na verdade ainda existem alternativas possíveis. Isso pode acontecer em duas situações: A combinação de falhas é tal que qualquer substituição possível seria mais longa (em número de ações) do que o limite estabelecido para a criação da tabela. Ou, as falhas são em dispositivos muito independentes. Como exemplo do segundo, os braços e rodas do nosso robô simulado são independentes; mas as substituições de ação dos braços costumam conter ações simples das rodas. Se todas as alternativas dessas ações simples das rodas forem citadas em todas as substituições dos braços, a multiplicação das entradas aumentaria o tamanho da tabela a níveis proibitivos.

Para esses casos, fazer a recursão do processo de substituição pode resultar em um plano válido. O subplano é empilhado normalmente, e quando necessário, ações inválidas são novamente substituídas. Para situações em que são poucos os dispositivos que falharam, em geral podemos achar substituições das substituições que geram um plano válido. Essa abordagem tem dois problemas. O primeiro deles é que o encadeamento de várias substituições faz com que o plano fique mais e mais vulnerável à ação de elementos dinâmicos do ambiente, de maneira similar aos subplanos "longos" mencionados anteriormente.

O segundo problema, mais sério, é a chance do robô "se trancar em um canto", através da recursão sucessiva de suas ações de substituição. Ele pode, por exemplo, substituir várias ações dentro de subplanos em sequência, se afastando mais e mais de seu objetivo original. Um sintoma comum dessa situação é quando o plano entra em laço, como na figura 3.3. A substituição para uma dada ação contém uma ação com erro, e a ação com erro contém a ação original que falhou. Laços de três ou quatro níveis também são possíveis, e são progressivamente mais difíceis de se perceber.

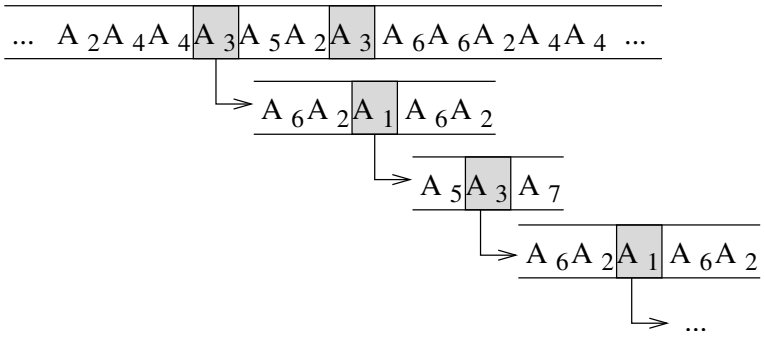

Figura 3.3: Exemplo de Loop. O sistema apresenta falhas diferentes que tornam a ação A_3 e A_1 inválidas. O subplano de substituição de uma ação contém a outra.

Para evitar esse problema, o sistema precisa armazenar as informações sobre quais substituições já realizou, quando começar a substituir suas ações, para que possa evitar de usar múltiplas vezes uma mesma substituição.

Uma complicação um pouco diferente na seleção das ações é a substituição de múltiplas ações. Durante a explicação inicial do sistema de substituição, falamos que a substituição se define por dois subplanos que são equivalentes entre si. Nos exemplos dados até agora, sempre falamos apenas da substituição de uma ação por um subplano. Mas em teoria, poderíamos substituir um trecho inteiro do plano por um sub-plano de uma vez.

A substituição de um subplano por outro subplano (a qual chamamos de substituição de múltiplas ações) consiste de um problema razoavelmente complicado. A primeira questão que se deve fazer é quando devemos verificar se uma substituição é necessária? Se estamos sempre substituindo uma ação por um subplano, essa pergunta tem uma resposta simples: antes de executar a ação. Porém, quando podemos ter um número arbitrário de ações válidas antes de uma ação não válida, no subplano que desejamos substituir, a resposta já não é tão simples.

Existem algumas vantagens em se tentar substituir um conjunto de ações de uma vez, ao invés de uma única ação. Para alguns sistemas, certas ações podem vir precedidas, ou sucedidas, por ações auxiliares que servem apenas de preparação para a ação principal (por exemplo, abrir e fechar a garra de um manipulador com apenas um grau de liberdade). Faz sentido procurar substituir essas ações como um grupo.

Porém, o número de opções disponíveis para substituição multiplica-se enormemente. Ao invés de apenas se escolher qual subplano utilizará para substituir, o algoritmo de substituição precisará decidir qual par conjunto de ações/substituição é o mais apropriado para uma dada situação. Não abordamos esse assunto em nossa pesquisa, mas tecemos alguns comentários no capítulo 6 para que sejam seguidos como direções futuras deste trabalho.

3.2 Sistema de Reparo de planos

Combinando o sistema de diagnóstico da aplicação, a tabela de substituição e a seleção de ação como descritos acima, podemos implementar um sistema para a recuperação de falhas no plano. Esse sistema co-existirá com algum tipo de replanejador, que atuará em circunstâncias extremas que não possam ser resolvidas pelo sistema de substituição.Procuramos minimizar tais ocorrências, dado que o custo do replanejamento pode ser alto demais para aplicações de tempo real onde o tempo de resposta seja importante.

O fluxo de ações do sistema de substituição ocorre de acordo com o descrito na figura 3.4. Inicialmente, o planejador cria um conjunto de ações (plano) para alcançar um objetivo. Essas ações são colocadas em uma pilha e executadas em série.

Para cada ação a ser executada, o sistema deve verificar se essa ação é válida ou não, de acordo com os dispositivos requeridos pela ação, e a situação desses dispositivos. Para isso, é utilizado o sistema de diagnóstico. Se a ação requerer um dispositivo que se

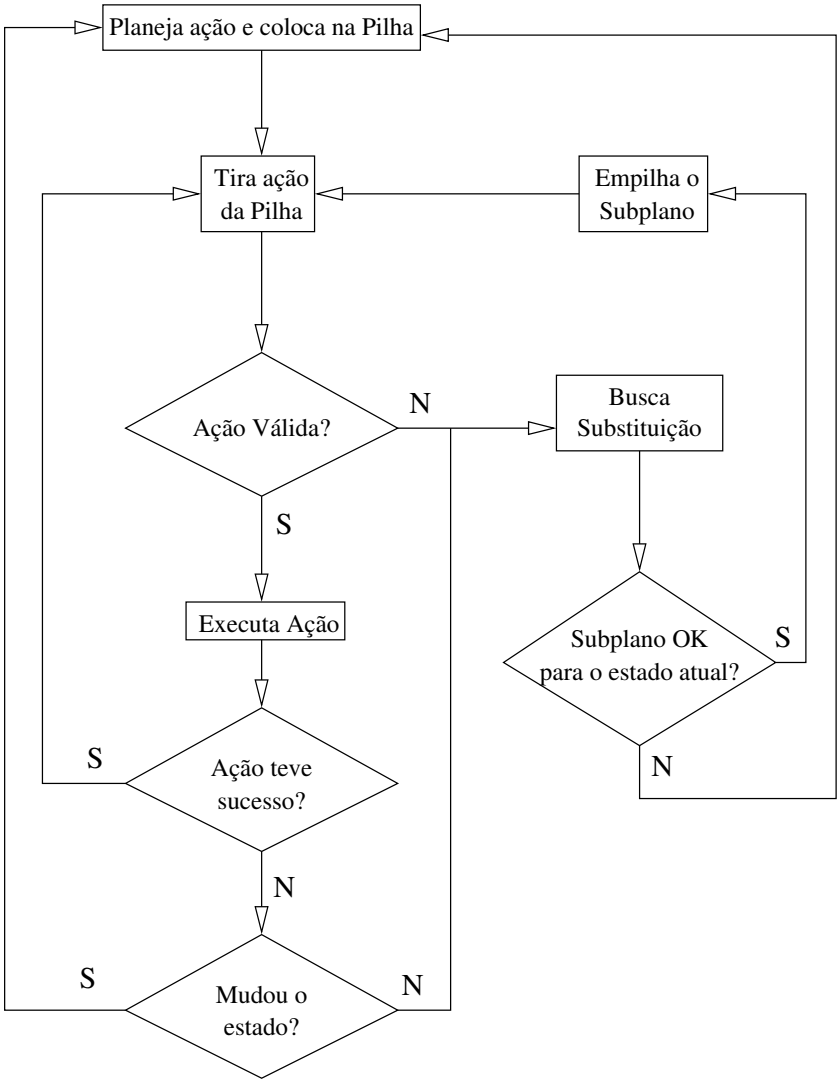

Figura 3.4: Fluxograma de um sistema de substituição

encontra falho no momento, o sistema de diagnóstico indica a necessidade de substituição.

Caso seja necessária a substituição da ação, o algoritmo de escolha verifica a lista de substituições disponíveis para aquela ação em particular, e escolhe o subplano mais apropriado. Faz a comparação das pré-condições do subplano em comparação com o estado do mundo, de maneira a saber se o subplano em questão é executável, e analisa o subplano para verificar que não existe um risco imediato de loop.

Havendo um subplano de substituição adequado, suas ações são colocadas na pilha de substituição, e o sistema volta para a fase de tirar uma ação da pilha. Se não for possível encontrar algum subplano que seja capaz de substituir a ação com problema, será necessário então utilizar algum algoritmo de replanejamento para tentar manter o sistema em execução.

Caso não seja necessária a substituição, é executada a ação atual. Se a ação teve sucesso, então o sistema volta para a fase de pegar uma nova ação da pilha de execução. Caso a ação falhe, é verificado o estado do mundo resultante. Se o estado do mundo não foi modificado pela execução da ação (a falha resultou na ação não sendo executada), tenta-se encontrar uma substituição para essa ação (marcando os possíveis sistemas que falharam). Se o estado do mundo mudou, de maneira que não é possível substituir a ação original, torna-se necessário realizar o replanejamento.

É importante notar que, no ato da substituição, apenas a instância atual da ação é substituída. Não ocorre, nesse momento, substituição de futuras ocorrências da mesma ação já previstas pelo plano. Como a escolha do subplano apropriado de substituição é dependente do estado atual do sistema (aplicação e ambiente), o subplano de substituição escolhido poder variar em diferentes momentos. Por exemplo, o estado do mundo pode ser outro, levando a um conjunto de pré-condições diferentes, ou a falha de certos dispositivos pode ter sido temporária, tornando esses dispositivos disponíveis para uso novamente. Assim, o reparo de um plano é uma atividade essencialmente reativa.

3.3 Diagnóstico planejado online

Diagnóstico planejado é uma técnica de diagnóstico onde o sistema alvo realizará uma série de ações para juntar a informação e isolar uma falha. Se temos um conjunto de dispositivos "candidatos" a uma falha que causou um erro no plano, e a informação disponível pelo sistema de diagnóstico da aplicação não é suficiente para determinar qual dos dispositivos apresentou falhas, podemos tentar isolar a falha testando os dispositivos, através de ações que os utilizem, até descobrir qual deles não está funcionando corretamente.

Uma maneira simples de realizar isso é testando os dispositivos um a um, executando em ordem ações que seriam afetadas por um dos dispositivos "suspeitos", mas não pelos outros. Podemos tornar esse procedimento um pouco mais eficiente se, ao invés de tes-

tarmos os componentes um a um, utilizarmos uma busca. Dividimos os "suspeitos" em dois grupos, e testamos uma das metades, como em uma busca binária. Para fazer essa busca, o sistema precisa conhecer um conjunto de ações (ou uma ação) que utilize todos os componentes de um dos grupos e nenhum do outro. Então executa-se esse conjunto de ações. A ocorrência ou não de um erro determina em qual dos grupos de componentes se localiza a falha, e repete-se o processo com esse grupo. A árvore binária sugerida pode ser generalizada para uma árvore de decisão, na qual o objetivo é descobrir qual dispositivo falhou, através da obtenção de informação relevante por ações.

Falta então determinar como escolheremos as ações para testar um determinado grupo. Podemos utilizar o planejador do sistema para escolher um conjunto de ações que satisfaça a idéia acima. Cria-se um plano no qual o objetivo é ter executado cada um dos dispositivos que desejamos investigar, e não ter executado os dispositivos que queremos excluir. O planejador tenta encontrar o plano que cumpra esses objetivos com o menor número de ações possíveis (figura 3.5).

O Diagnóstico Planejado é uma técnica barata de se implementar, na medida que não exige nenhum sensor extra ou componente de software complexo no sistema. Por outro lado, executar as várias ações necessárias para isolar o componente em falha significa interromper o plano em execução, e perder uma quantidade considerável de tempo. Além disso, a quantidade de ações com erros que tem que se executar deliberadamente para detectar a falha significa que o estado final do sistema pode ser impredizível. A idéia de um sistema de diagnóstico é que ele execute o seu trabalho com um mínimo de interferência no sistema sendo diagnosticado [75].

Podemos utilizar os componentes de um sistema de substituição para implementar o diagnóstico planejado de maneira que ele ainda seja relativamente barato, mas reduzindo o problema da interferência com a tarefa. A idéia é que, ao invés de executar o plano de diagnóstico todo de uma vez só, esse plano seja quebrado e "diluído" entre as ações normais do sistema. Assim, evitamos parar o sistema completamente para a execução do diagnóstico, e distribuímos os erros resultantes de ações falhas, evitando que seus efeitos se acumulem.

Quando ocorre a falha, primeiramente o sistema de diagnóstico marca todos os dispositivos relacionados como suspeitos de falha, a partir da comparação entre a pós condição gerada e a pós condição esperada. Se há algum fator ambiental que possa ter influência sobre o resultado da ação que errou, considera-se que todos os dispositivos daquela ação possam ter apresentado falha (pois se torna impossível medir a influência do ambiente no resultado da falha). Através da diferença de pós condições resultante do erro, escolhe-se uma ação ou plano que permita ao robô voltar ao caminho do plano original ou, se necessário, faz-se um replanejamento parcial local.

Durante a execução do plano, tentamos lentamente juntar informações sobre os dispos-

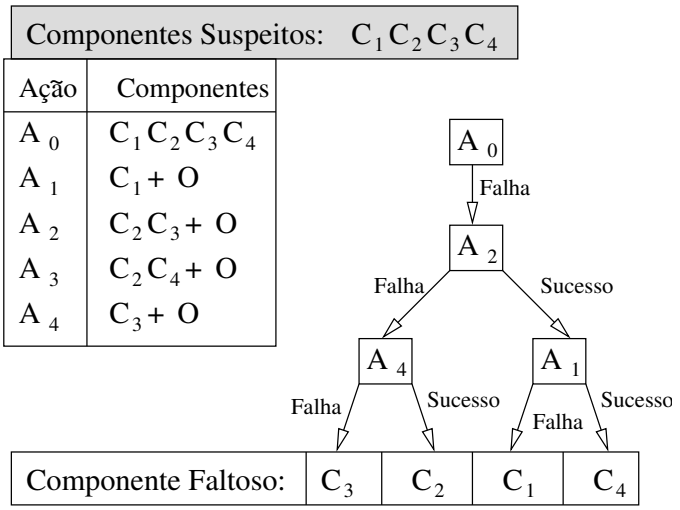

Figura 3.5: Exemplo simples de um diagnóstico planejado. A ação A_0 falha, fazendo com que todos os seus componentes sejam considerados suspeitos. O sistema então executa ações com alguns desses componentes de cada vez para descobrir qual deles é o faltoso.

itivos suspeitos, para poder isolar a falha. Essas informações vêm de *ações de diagnóstico*. Inicialmente, essas ações de diagnóstico são ações, no plano, que contém alguns dos dispositivos suspeitos, mas não todos. O sistema *não* substituirá essa ação, para ver se isso gerará um erro ou não. Dependendo do resultado, observamos se a falha procurada se encontra ou não entre os dispositivos relevantes à ação executada, e adicionamos essa informação ao nosso sistema de diagnóstico.

Para decidir quando realizar a ação de teste do dispositivo (deixar de executar a substituição), utilizamos as seguintes regras para o sistema:

1. A ação de teste deve possuir o menor número possível de dispositivos "suspeitos". Assim, podemos obter informações mais precisas de cada ação. Como as substituições do plano garantem que a tarefa pode ser concluída apesar das falhas, não há a necessidade de testar vários dispositivos de uma vez. Idealmente, testaremos um dispositivo por ação.

2. A ação de teste deve ser executada quando as influências do ambiente forem mínimas no resultado da ação, para não confundirmos um erro devido ao ambiente com um erro devido ao dispositivo que estamos analisando. Isso requer conhecimento da dinâmica do mundo da aplicação. Por exemplo, se uma falha fizer com que um robô vire um pouco à direita, no momento dessa falha deve-se evitar que haja algo que bloqueie esse movimento. Este ponto é mais importante no caso de ações com mais

de um componente suspeito, já que o comportamento do erro pode indicar qual dos componentes falhou.

3. Após determinarmos que um dispositivo está falhando, devemos continuar a testar os outros, devido à possibilidade de múltiplas falhas.

Podemos melhorar a eficiência da substituição aumentando um pouco a fase de planejamento após uma falha. Ao invés de simplesmente recuperar o sistema da falha e continuar a execução da tarefa normalmente, podemos preparar um roteiro de como o diagnóstico deve ocorrer. O primeiro requisito desse roteiro é ordenar os dispositivos suspeitos na ordem e agrupamento pelos quais quais eles devem ser testados. Se temos dois dispositivos, A e B, tais que A sempre ocorre junto a B, mas B pode ocorrer isoladamente de A, testar B primeiro pode nos deixar numa situação na qual não existe uma ação que possibilite que A seja testado. Normalmente podemos inferir a situação de A por eliminação, baseado na situação dos outros dispositivos. Mas se houver a possibilidade, no sistema, de dois ou mais pares desses ocorrerem numa mesma ação, podemos nos encontrar em uma situação onde é impossível isolar completamente o problema sem repetir testes desnecessários.

Uma adição mais avançada que pode ser feita no sistema de diagnóstico planejado é o teste ativo das falhas. Ao invés de esperar a ocorrência de uma ação com falha para executar um subplano de diagnóstico, o sistema de substituição analisará cada ação que é realizada pela aplicação. Se o estado do ambiente e as substituições disponíveis forem propícios, faz-se a substituição daquela ação com o objetivo de adicionar informação ao diagnóstico. Fazer uma análise do ambiente em combinação com as substituições possíveis a cada ação executada causa um overhead considerável, proporcional ao número de dispositivos "suspeitos". Porém, isso pode ser necessário no caso em que as ações que falham ocorram sempre em condições ambientais não favoráveis ao diagnóstico planejado.

Caso estejamos lidando com uma aplicação não determinística, uma precaução extra deve ser observada ao montar o sistema de diagnóstico planejado. O não determinismo pode atuar de duas maneiras em nosso sistema. Primeiro, e mais comum, o resultado de ações falhas pode não ser sempre o mesmo. Além disso, o resultado de ações normais pode ser diferente do esperado. A partir do momento em que só podemos confiar que o resultado de uma ação é o esperado com uma probabilidade P, precisaremos executar os testes um número de vezes tal que nos permita um valor razoável de confiança no resultado.

Cada regra que se adiciona ao sistema de escolha de subplano aumenta a qualidade da substituição escolhida (no sentido que ela tem maiores chances de trazer mais informação para o diagnóstico, e menos chance de resultar numa falha que perturbe o plano). Mas aumenta também a complexidade do sistema que vai escolher a ação. Diferentemente do sistema de reparo de planos, aqui podemos admitir uma demora maior na escolha da

substituição adequada. O tempo limite, como sempre, é a comparação da substituição com um replanejamento completo das ações. Neste caso, porém, as regras descritas acima podem ser usadas também para guiar o replanejamento, caso esse seja usado para criar planos de diagnóstico.

Capítulo 4

Aplicação em robótica

O trabalho com substituição de ações, reparo de planos e diagnóstico planejado foi primeiramente desenvolvido a partir de um modelo de robô de exploração. Não por coincidência, sistemas robóticos autônomos contém várias características que são interessantes para a aplicação desta técnica.

Uma destas características é o grau de redundância indireta normalmente presente em robôs. Por redundância indireta queremos dizer um dispositivo que, embora não seja exatamente idêntico ao outro, pode ser usado para o mesmo objetivo, talvez em conjunto com outros dispositivos. Por exemplo, diferentes tipos de sensores externos em um robô (toque, radar, visão), apesar de diferentes entre si, tem áreas de intersecção em seus campos de atividade. Em um robô humanóide o conceito de redundância indireta fica ainda mais claro quando, por exemplo, poderíamos utilizar os braços para fazer um robô se arrastar, caso suas pernas apresentassem defeito. Essa característica da redundância indireta indica a existência de substituições contendo dispositivos bem diferentes dos necessários para a ação a ser substituída, permitindo o reparo de planos para conjuntos de falhas mais complexos.

Outra característica que facilita a implementação do sistema de substituição em robôs é o fato que, como dito no capítulo 2, o planejamento de um robô em geral pode ser reduzido ao problema do path planning para todos os seus atuadores. No path planning, podemos efetuar operações de soma vetorial entre as posições iniciais e finais descritas pelas ações. Através dessas operações, podemos calcular mais facilmente quais conjuntos de ações são equivalentes a quais outros.

Na primeira seção será abordado o trabalho de pesquisa feito junto à nossa plataforma simulada, o DODÓI. O dodói é um robô simulado baseado em robôs de exploração autônomos. Descreveremos brevemente as características da aplicação, e então discorreremos sobre os resultados dos experimentos que nos levaram às conclusões descritas no capítulo 3.

Na segunda seção, descreveremos o início do trabalho desenvolvido no robô bípede WL-16. Apesar do WL-16 não usar um sistema de controle por planejamento "típico", como o dodói, as considerações de implementação traçadas para ele são importantes para o uso do sistema de substituição em robôs reais. O trabalho no WL-16 não chegou a ser completado, devido a problemas externos ao projeto, mas suas conclusões parciais são interessantes de serem estudadas de qualquer maneira.

4.1 Dodói: Robô explorador simulado

Para todos os estudos e provas de conceito relacionados com este projeto, desenvolvemos um simulador simples, descrito a seguir. A premissa do simulador é de um veículo autônomo de exploração e coleta de amostras.

O ambiente representado pelo simulador consiste de uma sala plana ocupada por diversas paredes intransponíveis. Dentro do ambiente também encontram-se pedras de diferentes pesos, que podem ser levantadas e carregadas para outros lugares. O espaço no ambiente é descrito por um quadriculado. A simulação ocorre em turnos, com uma ação do robô por turno (o ambiente é estático, com excessão das falhas do robô).

O robô é composto por um corpo retangular com um container para um número arbitrário de pedras, e 6 atuadores, sendo 4 deles motores (um para cada uma das quatro rodas), e dois deles as garras manipuladoras do robô.

Cada motor pode ser encontrado em um de três estados: Acionado para a frente, acionado reverso, e parado (indicados por 1, -1, e 0). Eles se localizam próximos aos extremos do corpo retangular do robô. Cada combinação desses quatro motores determina a movimentação do corpo do robô. Assim, temos 243 possíveis combinações de ações de movimento (descritas parcialmente na figura 4.1).

As duas garras possuem, cada uma, dois graus de liberdade. Cada garra pode se mover sobre 7 regiões diferentes: 6 ao redor do robô, e uma sobre o corpo do robô. Cada garra pode também abrir-se ou fechar-se. Uma garra fechando ou abrindo sobre uma das regiões ao redor do robô agarra ou solta, respectivamente, uma pedra que esteja naquela região. Se uma garra abre-se sobre a região sobre o robô enquanto carrega uma pedra, ela "guarda" essa pedra no robô. Assim, cada garra tem um total de 13 ações. Cada garra pode segurar uma pedra de peso 1 individualmente, ou, em conjunto, segurar uma pedra de peso 2. As ações das garras são ilustradas na figura 4.2.

Essa configuração de ações provê um alto grau de redundância. Isso significa que o mesmo efeito pode ser alcançado por várias ações diferentes. Essa redundância nos permite gerar situações interessantes para um sistema de substituição, com várias opções para geração de alternativas.

Há também um conjunto de falhas possíveis para o sistema, que podem ocorrer de

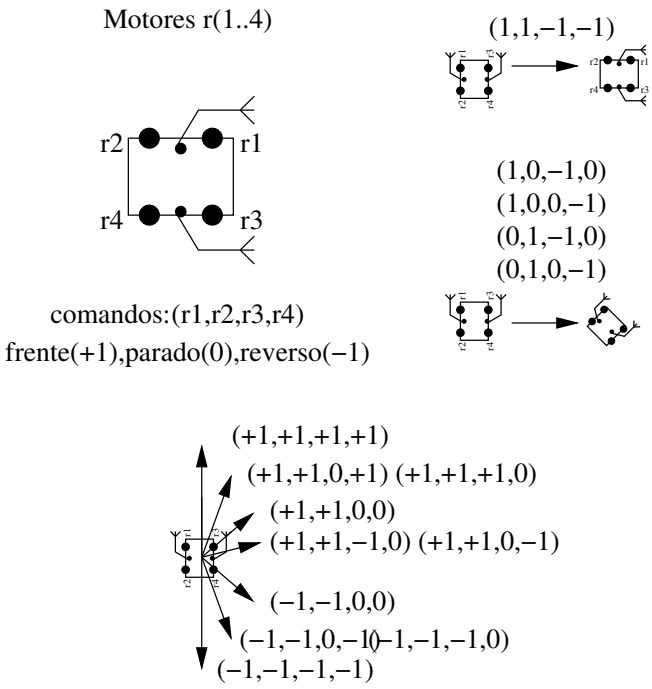

Figura 4.1: O robô se move no plano através da combinação dos estados dos 4 motores. O corpo em si pode estar em uma de 8 orientações.

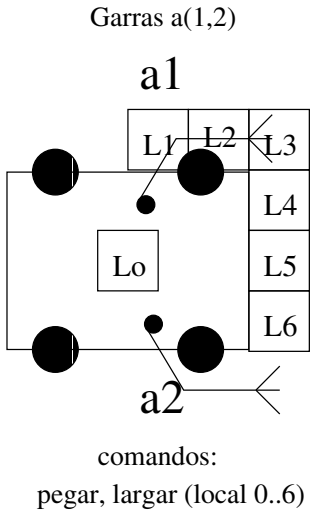

Figura 4.2: Cada garra pode agir em 6 regiões ao redor do robô, e sobre o robô em si. Há uma intersecção entre as áreas de ação das garras, para aumentar o nível de redundância do sistema.

maneira aleatória durante a execução de um plano, ou serem inseridas. Para cada um dos quatro motores temos 4 configurações de falha: nenhuma falha, incapaz de mover-se para frente (movendo-se apenas para trás, ou ficando parado), incapaz de mover-se para trás, ou totalmente parado. Com 4 estados de falha diferentes para 4 motores, temos um total de 256 combinações de falha.

Para os dois braços, cada um pode se tornar incapaz de agir em cada uma das 7 áreas, isoladamente, ou em conjunto, para um total de 128 configurações de falhas. Dessas, 7 onde o braço só pode se encontrar em uma posição serão equivalentes àquela em que o braço não pode se mover, reduzindo o número de configurações para 121, ou 14.641 combinações de falhas para ambos os braços.

4.1.1 Implementação do sistema de substituição no robô simulado

O trabalho de implementação do sistema de substituição no *Dodói* determinou a direção do desenvolvimento do modelo de sistema de substituição apresentado nesta tese.

Nós buscamos, a cada etapa do projeto, desenvolver um sistema que atendesse a duas características: 1 - Permitisse ao robô alcançar seus objetivos, e 2 - Fosse mais interessante de ser utilizado do que o replanejamento, para o mesmo problema.

Para testar essas duas características, utilizamos os seguintes testes a cada passo do projeto. O robô é colocado dentro de um labirinto, e um plano é formado para ele (normalmente levando o robô de uma ponta à outra do labirinto). Os labirintos variam em tamanho e em complexidade (alguns exemplos de labirintos utilizados estão na figura 4.3).

Figura 4.3: Alguns exemplos de labirintos utilizados no projeto

Em cada teste, falhas eram progressivamente inseridas no robô durante a execução da tarefa. A inserção de falhas era feita de acordo com o seguinte critério:

- Inserção incremental de defeitos: Durante a execução, novas falhas eram inseridas a cada 50 ações, até que o robô chegasse a uma situação em que ele não pudesse mais

continuar sua tarefa. A ordem da inserção das ações varia de acordo com a tarefa sendo executada naquele teste - por ordem dos dispositivos mais usados naquela tarefa.

- Inserção aleatória de defeitos: Durante a execução, um teste era feito a cada 50 ações para decidir se uma nova falha seria adicionada ou não. O defeito a ser adicionado é escolhido aleatoriamente.

- Inserção temporária: Durante a execução, a cada 50 ações uma falha era inserida no sistema de maneira aleatória. Essa falha era retirada do sistema depois de mais 50 ações, sendo então trocada por outra.

Desenvolvimento do Sistema de Substituição

No início do projeto tentamos manualmente gerar uma tabela de substituição adequada para utilização no robô simulado. O resultado que encontramos foi um número enorme de casos onde a substituição não ocorria, devido às alternativas de substituição existentes serem inadequadas para as situações e falhas que se apresentavam. Com três falhas simultâneas, a maior parte das substituições manuais se tornava inválida, e muitas das restantes estavam simplesmente erradas, tendo que ser corrigidas durante os testes. O total de ações na tabela de substituição era aproximadamente 200.

Com esse problema, realizamos a geração automática para movimentos das rodas, como descrito no capítulo 3. O número de substituições existentes era enorme, e para podermos utilizá-las na tabela de substituição, tivemos que desenvolver e aplicar uma série de filtros nas ações encontradas (como exemplificado na tabela 4.1.

Filtro Aplicado	Número restante de substituições
Total	48.429
Cortando loops diretos	3051
Cortando ações que utilizam mesmas pré-condições	657

Tabela 4.1: Número de substituições disponíveis para a ação 1 1 1 1, com até 5 passos

Esse número é para todas as substituições da ação (1,1,1,1), a que possui o maior número de substituições (devido ao design do robô). Para cada falha em particular utiliza-se um subconjunto dessas ações. Por exemplo, para uma falha total do motor esquerdo da frente, temos das seiscentas, por volta de 50 ações substitutivas disponíveis.

Um outro fator importante para a geração de ações neste ambiente é o nível de profundidade. Se aumentarmos o limite de passos para os subplanos de substituição de 5 para 6

ações, o maior número de substituições citado acima sobe para 2×10^{11}. Para esta ação em particular, podemos achar a maioria das substituições válidas entre 2 e 4 passos. Para outras ações, a complexidade das substituições aumenta, tornando a busca mais custosa.

Comparações com Replanejamento

Comparamos o desempenho do robô quando este utilizava a substituição para reparar as falhas do plano, e quando utilizava o replanejamento. Efetuamos dois tipos de comparação. O primeiro é a comparação em número de ações necessárias para a conclusão do plano. O segundo é a comparação da quantidade de tempo gasta durante a execução da tarefa.

A quantidade de tempo gasta é a principal medida que queremos utilizar para comparar a substituição com o replanejamento. Como veremos abaixo, a cada falha o replanejamento ocupará uma boa parte do tempo da tarefa quando comparado à substituição. Utilizamos inicialmente um simples replanejador A*, e depois um replanejador local, que demonstrou uma eficiência um pouco melhor em certas situações.

O tamanho do plano é a "compensação" que temos que pagar pelo uso da substituição. De uma maneira geral, planos substituídos ficaram muito maiores do que os replanejados. Para entender o porquê, segue a descrição de uma situação encontrada no teste:

Para o movimento "andar à frente" (1,1,1,1), a substituição escolhida pelo sistema consistia em girar o robô de costas, andar para trás o mesmo espaço, e virar o robô de frente (5 ações - 2 para cada virada). Em um longo trecho de caminho que o robô tinha que andar em linha reta, o planejador simplesmente virou o robô, percorreu todo o caminho, e virou-o de frente no final deste. Já o sistema de substituição fazia o robô dançar, rodando-o para trás no início de cada substituição, e para frente ao final de cada uma.

Assim, devido às ações preparatórias antes e depois da substituição, temos essa degradação da eficiência do plano. Testes foram feitos em diferentes percursos nos quatro mapas mostrados anteriormente, utilizando o A* como planejador, e diversos conjuntos de falhas.

Esses valores são para os testes com falhas aleatórias. Aqueles com falhas incrementais, quando levados até o ponto onde o robô não pode mais operar, acabaram se mostrando não muito úteis para este tipo de comparação (pois o tempo de execução da tarefa se tornava infinito em qualquer caso, pelo fato do robô não poder continuá-la. Quando limitados a um certo número de falhas, os testes incrementais mostraram resultados semelhantes aos acima.

Já os testes com falhas temporárias mostram uma significativa diminuição no número de ações para o sistema de substituição. O motivo disto é que quando uma falha "reverte" para o seu estado normal, o sistema de substituição para de trocar essa falha. O plane-

jador, porém, continua com o mesmo novo plano anterior, que em geral é bem diferente do plano original, e não toma vantagem do fato da falha ter se resolvido.

Para compensar esta mudança, passamos a utilizar um replanejador local, que ao invés de replanejar todos os movimentos após a falha até o final da tarefa, replanejava apenas do ponto de falha até o próximo ponto do plano no qual o dispositivo falho não é necessário. Nos testes com falhas aleatórias em que ocorria apenas uma falha, houve uma melhora sensível na eficiência do replanejador em comparação com o sistema de substituição, mas essa melhora se perdia rapidamente com o aumento do número de falhas.

Observamos que quanto mais complexo o sistema (refletido pelo ambiente e pelo número de falhas), maior a diferença entre a eficiência da substituição e do replanejamento. Havendo uma solução para alcançar o objetivo da tarefa através da substituição, esta se mostra vantajosa quando é necessária uma reação imediata a uma falha.

Experimentos com braços

A implementação do sistema de substituição, nos braços, trouxe um conjunto de desafios interessantes e diferentes, comparados ao das rodas. Diferentemente destas, o sucesso das ações das garras do robô é dependente de certas condições do ambiente externo. Além disso, como uma mesma ação da garra é capaz de pegar uma pedra em diferentes posições (a garra tem seu campo de ação em 9 pixels, como explicado na seção anterior), deixa a situação um pouco mais complexa.

Podemos caracterizar as garras como um sistema no qual existem poucas ações provenientes dos braços em si, mas o grande número de variações do ambiente essenciais para a ação faz com que haja várias possibilidades de pré-condições para a tabela de substituição.

Devido a essas característica, criar uma tabela de substituição única para o braço e as rodas mostrou-se complicado. O número de possíveis configurações dos dois sistemas, quando combinado, torna-se grande demais para ser tratável. Assim, aplicamos dois sistemas de substituição paralelos, um para o braço, e outro para os motores das rodas.

O sistema de substituição das rodas, como descrito anteriormente, ignora completa-

Média de ações Substituição	Média de tempo Substituição	Média de ações Replanejamento	Média de tempo Replanejamento
117	27s	1	275s
60	47s	1	720s
24	491s	1	789
120	265s	1	1656

Tabela 4.2: Comparação de tempo de execução e número de ações para testes com falhas aleatórias

mente as ações do braço. Já o sistema de substituição dos braços consideram que as rodas nunca falham, e criam suas substituições de acordo com essa crença. Caso haja uma falha de um dispositivo das rodas durante a execução de uma ação de substituição do braço, o reparo dessa falha em particular é delegado de volta para o outro sistema (como ilustrado na figura 4.4). Assim, a substituição passa a ser feita progressivamente, primeiro substituindo a ação falha dos braços, e em seguida substituindo ações das rodas conforme o necessário. Este esquema pode ser repetido para sistemas mais complexos, com vários dispositivos independentes.

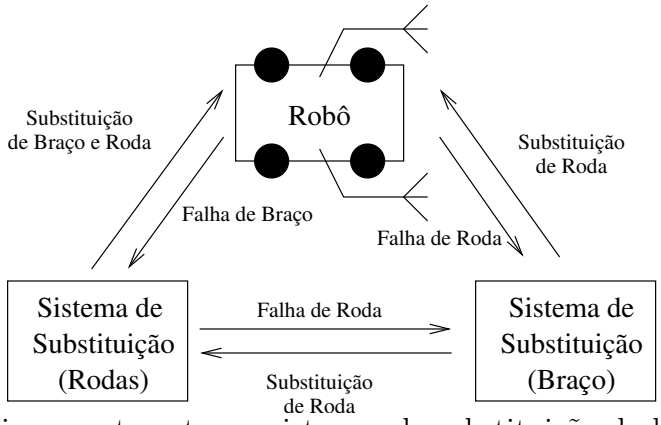

Figura 4.4: Relacionamento entre os sistemas de substituição do braço e das rodas.

Diagnóstico Planejado

O principal problema para a implementação de técnicas de diagnóstico planejado é como lidar com os erros que serão causados pelas ações de diagnóstico. Em nossos trabalhos, planos para detecção de múltiplas falhas nas rodas requeriam em média 3 erros para poder isolar o componente em falha. Usar replanejamento para corrijir esses erros custa muito caro.

Para contornar esse problema, adotamos uma solução na qual calculamos a diferença entre o estado desejado e o resultado final. A partir dessa diferença, podemos saber qual foi a ação que foi executada. Isso nos ajuda em dois pontos: primeiramente, a partir da diferença entre a ação desejada e a realmente executada, podemos ter uma idéia de quais dispositivos provavelmente falharam (verificando quais combinações de dispositivo participam da ação realizada). Em segundo lugar, usando essa informação como pré-condição, verificaremos se existe uma substituição indexada que nos leve até a pós-condição (ponto no plano) originalmente desejada.

4.2 WL16: Robô bípede

Na área de robótica, além do trabalho com o *Dódoi*, descrito acima, também foi realizado brevemente um trabalho de substituição de ações com o robô bípede humanóide *WL-16*, da universidade de Waseda [81]. O WL-16 consiste de uma base sustentada por duas pernas, cada uma acionada por 6 atuadores paralelos (como na figura 4.5).

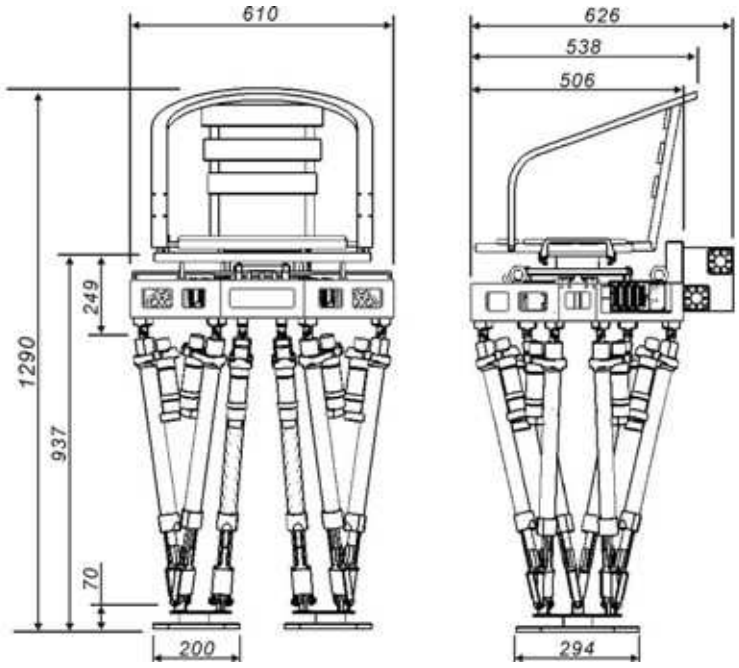

Figura 4.5: Estrutura do robô WL-16

A idéia do WL-16 é servir de base de locomoção para outros equipamentos robóticos que seriam colocados sobre sua parte superior. Outro objetivo no seu desenvolvimento é servir como uma "cadeira de pernas" para pessoas com problemas físicos. Assim, a pesquisa atual tem se concentrado em aumentar a capacidade de peso carregável pelo robô, e sua habilidade de andar por terrenos irregulares.

O WL-16 difere bastante da idéia de robô descrita nesta dissertação, pois a geração de caminho para ele não é feita por um planejador. No seu estado de desenvolvimento atual, o caminho em alto nível (por onde ele vai andar), é determinado por um operador humano. O controlador do robô então calcula a sequência de posições de cada um dos 12 atuadores com uma resolução de 0.001 segundo para determinar a trajetória dos passos. A geração de cada passo aproxima-se das técnicas de "potential field path planning", descritas no capítulo 2.

Assim, as "ações" que podem ser definidas para o robô são contínuas, ao invés de

discretas, como no planejamento tradicional. Essa diferença poderia ser modificada em outro sistema abstraindo-se essa camada de baixo nível, e lidando com o plano em um nível mais alto, como o caminho que o robô percorre. Mas nesta aplicação em especial isso ainda não estava disponível.

O WL-16 oferece um desafio muito interessante para o sistema de substituição, na medida que uma falha no meio do passo tem um resultado desastroso imediato (queda do robô). Assim, uma solução precisa ser encontrada instantâneamente. Essa é a situação ideal para se aplicar um plano de substituição.

4.2.1 Substituição no WL-16

A substituição no WL-16 acontece no "plano" criado pelo computador para dar um passo. O controlador do robô é capaz de gerar a sequência de posições para um passo rapidamente quando está em uma posição estável (parada). Porém, se um erro ocorre no meio do passo, não há tempo para calcular a movimentação necessária para evitar que o robô caia. Assim, é necessário que haja um conjunto de ações (constituídas de sequências de configurações do atuador) prontas que possam ser substituídas rapidamente para estabilizar o robô. Uma vez estabilizado, ele pode calcular com calma os movimentos seguintes para continuar a tarefa.

Dois tipos de falhas podem causar o erro no passo. O primeiro tipo acontece devido ao travamento de um dos atuadores. O diagnóstico desta falha é realizado comparando a posição real dos motores com a posição estimada dos mesmos, e verificando que a diferença está acima da variação esperada. O outro tipo ocorre quando um valor incorreto de "controle de variância" [82] é usado. Esse valor regula a resposta do robô a irregularidades do terreno, e um valor incorreto pode causar um aumento na instabilidade, e eventual queda, do robô. O diagnóstico dessa falha se dá por uma variância maior do que a normal nos sensores de posição da cintura do robô.

O número de substituições necessárias, então, é razoavelmente grande. Para cada configuração possível do robô, temos que ter uma sequência de configurações dos atuadores que levem daquela posição até uma posição estável. A geração dessa sequência em si não é problemática, uma vez que as fórmulas utilizadas para a geração do passo e para a geração da substituição são as mesmas (um problema em aberto é, como fazer com que o sistema de geração de passo encontre uma posição final válida automaticamente, caso um dos atuadores "trave").

Assumindo uma granularidade do controlador de 0.001 segundo, e um passo de 0.92 segundo de comprimento, temos 920 configurações diferentes para um único passo. Apesar do número mínimo de atuadores em funcionamento necessários para ser possível se alcançar uma posição estável não ser conhecido, se pensarmos que no máximo 2 dos 12

atuadores possam quebrar antes que seja impossível encontrar uma solução para a falha, teremos aproximadamente 60.000 planos de substituição para esse passo.

Para cada passo teremos um conjunto diferente de planos de substituição. Porém, como cada conjunto é individual para aquele passo, podemos criar uma tabela para cada tipo de passo (de acordo com a velocidade, pé direito ou esquerdo, peso carregado), e trocar as tabelas durante a execução da tarefa. Assim, quando ocorre a falha, robô só terá que procurar junto às opções de substituição daquele período de tempo.

Capítulo 5

Aplicação em web services

Um web service é um conjunto de protocolos e padrões usados para a troca de dados entre aplicações. Aplicativos de Software escritos em diferentes linguagens de programação, e rodando em diferentes plataformas podem usar web services para trocar dados através de redes de computadores, como a internet. Essa interoperabilidade (por exemplo, entre Java e Python, ou aplicativos de Windows e GNU/Linux) se dá devido ao uso de padrões abertos.

Assim, os web services provêem uma conexão fraca entre componentes heterogêneos de software e provedores de serviços, para satisfazer requisições de usuários. Conforme eles se popularizaram, requisições que combinam múltiplos web services de maneiras variadas começaram a surgir, gerando o problema da *composição de web services*: Dado um objetivo que pode ser alcançado por um número de web services disponíveis de diferentes maneiras, como escolher quais web services combinar, e em que ordem, para alcançar aquele objetivo?

Compor manualmente um conjunto de web services torna-se uma tarefa difícil e disposta a falhas, conforme os sistemas ficam maiores e mais complexos. Em resposta a isso, a comunidade do web service procurou soluções no campo do planejamento em inteligência artificial, com bons resultados (como discutido no capítulo 2). Quando transportado o problema para a área de planejamento, cada serviço é uma ação, que comporá o plano que leva o sistema do estado inicial até o objetivo definido pelo usuário/cliente. Para isso, as interfaces de web services foram extendidas de maneira a tornar sua operação compreensível para outros serviços: a "web semântica".

Conforme os provedores de web services começarem a tomar decisões de maneira autônoma, eles também precisarão se tornar capazes de lidar com falhas no planos que utilizarem. Serviços compostos que recebam muitas requisições simultaneamente se tornam complexos demais para que operadores humanos sejam capazes de monitorar e resolver cada falha.

Assim, estudamos como aplicar a técnica de substituição de ações a web services de maneira a torná-los capazes de reparar automaticamente falhas nos planos. Nossa idéia é utilizar a redundância inata do sistema para provê-lo com tolerância a falhas de planos, em troca de certa perda de eficiência (medida pelo custo do plano). Em web services, essa redundância normalmente é encontrada na existência de serviços semelhantes (ou algumas vezes idênticos) providos por companhias distintas.

Por exemplo, uma composição para um serviço web de transporte de bens. Digamos que o método escolhido para entregar um certo produto a um cliente é o transporte por caminhão, que também, incidentalmente, é o método mais barato. Assim, o web service faz uma requisição para a companhia de transporte para levar a carga em um determinado horário e custo, e insere essa ação no plano. Após algum tempo, antes da ação se realizar, um prego faz com que o transporte por caminhão se torne indisponível. O web service deve mudar esse plano, substituindo a ação *Transporte por Caminhão*. O sistema de substituição terá já pré-calculadas algumas alternativas para a ação falha, a um custo mais alto. Por exemplo, transportar a mesma carga por avião.

Neste capítulo teceremos considerações sobre a implementação de um sistema de substituição de ações para um web service que utiliza planejamento. Primeiro descrevemos o serviço, um web service parar atender pedidos de logística (transporte de cargas), citando algumas características que achamos importantes na modelagem de web services com sistemas de substituição. Na seção seguinte discutimos a implementação de um sistema de substituição nesse modelo proposto.

5.1 Descrição da Aplicação

Descrevemos abaixo um modelo de web service para um sistema de transporte de bens, que será usado como pano de fundo para as considerações de implementação do diagnóstico planejado em web services.

O sistema descrito trabalha como um centralizador de pedidos para a companhia, que possui diversos serviços de transporte. Assim, o cliente informa que deseja transportar uma dada carga de um local inicial até um local final, e o sistema então escolhe conjuntos de serviços que atendam a essa requisição. Ele apresenta essas opções ao cliente, que escolhe uma delas, e o planejador compõe então uma sequência de ações que atenderão à requisição do cliente.

Modelamos esta situação utilizando o planejamento clássico, representando o estado mundo como um conjunto de predicados, e os serviços como ações que atual sobre esses. Os predicados no nosso modelo são a definição de tempo, a carga e suas características (que determinam que tipo de serviços podem trabalhar com aquela carga em particular), os recursos utilizados pelos diversos serviços, como descrito abaixo, e os locais. O mundo

é dinâmico. Além do tempo passar conforme as ações são executadas, a disponibilidade de recursos também muda com o tempo, de maneira independente das ações do sistema.

Para compor o plano que irá atender à requisição, o web service tem disponível dois tipos de ações, baseadas nos serviços disponíveis pela rede: serviços de *transporte* e de *armazenamento*.

Os serviços de transporte levam a carga de um local para outro. Em nosso exemplo eles podem representar transportadores locais, que podem receber e entregar pedidos em qualquer local de uma dada "área de efeito" (ex. peruas, motoboys), ou transportadores intermunicipais/estaduais/nacionais, com local de origem, destino e cronogramas fixos (ex. navios, trens, aviões). Além disso, os serviços dividem-se em *internos* e *terceirizados*. De uma maneira geral, os serviços terceirizados custam mais caro que o seus equivalentes internos, mas eles servem a função, no nosso sistema, de prover soluções redundantes para falhas nos serviços internos.

Cada serviço de transporte é definido por: um conjunto de recursos necessários para a realização do serviço (para um caminhão, por exemplo, um motorista, combustível, o caminhão em si, etc - que são o equivalente, no ambiente dos robôs, aos dispositivos), um conjunto de pré-condições (hora e local de partida, restrições de capacidade de carga, custo), e um conjunto de pós condições (hora e local de chegada). O web service pode então requisitar dois tipos de ação para o serviço de transporte: *Confirmar* e *Usar*. A ação confirmar é o nosso sistema de diagnóstico. Ela serve para verificar se o serviço está disponível para uso, e se todas as pré condições são atendidas no estado atual do mundo. A ação usar faz o serviço realizar o transporte dos bens do local de origem ao local de destino, após a passagem do tempo estipulado na descrição do serviço, causando o gasto em dinheiro do custo relacionado ao serviço.

Os serviços de armazenamento mantém a carga em um determinado local por um período de tempo determinado. Em nosso exemplo, eles servem para conectar serviços de transporte consecutivos, que tenham uma "janela" de tempo entre si. Da mesma maneira que os serviços de transporte, temos serviços de armazenamento com diferentes características (em geral relacionadas com os tipos de cargas que podem armazenar. Também temos serviços de armazenamento "internos" e "terceirizados".

Da mesma maneira que os serviços de transporte, os serviços de armazenamento são definidos por: um conjunto de recursos necessários (eletricidade, empregados, segurança, etc), um conjunto de pré-condições (horário de entrada, capacidade de armazenamento, custo), e uma série de pós-condições (horário de saída, valor pago). O web service também pode realizar duas ações: *Confirmar*, que verifica se o serviço está disponível, como descrito antes, e *armazenar*, que requisita e armazena a carga por um certo período de tempo no local, e aplica um certo custo.

Na tabela 5.1 mostramos um exemplo de requisição para esse sistema, e um plano de

composição de web services que satisfaria o pedido. Em geral, a composição de serviços web é mais complexa do que o plano ilustra, envolvendo a troca de mensagens entre o cliente e provedor de serviço para combinar mutuamente várias variáveis. Mas essas trocas já são bem trabalhadas na literatura de web service (ref. capítulo 2), de maneira que podemos abstrair essas negociações e nos concentrar em um nível um pouco mais alto. Na figura 5.1 podemos ver um conjunto de serviços relevantes ao plano em questão, que ilustra como seria a distribuição de serviços no sistema como um todo.

Requisição	Plano.
Transportar 200 relógios Rolax de Asakusa para a Praça da Sé	*Confirmar* serviço de entrega de Asakusa para Yokohama - SE recebeu a confirmação, *Transportar* de Asakusa para Yokohama *Confirmar* serviço de entrega de Yokohama para Santos - SE recebeu a confirmação,*Transportar* de Yokohama para Santos *Confirmar* serviço de entrega de Santos para Jabaquara - SE recebeu a confirmação,*Transportar* de Santos para Jabaquara *Confirmar* serviço de entrega de Jabaquara para Sé - SE recebeu a confirmação,*Transportar* de Jabaquara para Sé

Tabela 5.1: Um exemplo de plano - A lista de ações disponíveis está na figura 5.1

Definiremos dois casos de falha para o nosso sistema. Primeiramente, um serviço (ação) não possuir parcialmente ou totalmente os recursos necessários na hora da execução, de forma que não pode realizar ação planejada. Este tipo de falha pode ser detectado pelo uso da ação *Confirmar* durante a execução da tarefa, e faz com que torne-se necessário trocar a ação falha no plano por um subplano de substituição. Por exemplo, um caminhão pode se encontrar inesperadamente cheio, devido a um pedido emergencial de outro cliente, tornando-se incapaz de realizar a ação planejada, devido a falta de capacidade de transporte.

Existe também o caso onde a ação *confirmar* retornou uma indicação de que o pedido do serviço ocorreria normalmente, mas problemas externos (no ambiente) fazem que a próxima ação retorne um erro. Por exemplo, um serviço de transporte aéreo pode sofrer de atrasos devido a uma súbita tempestade, ou um terremoto pode desabilitar o uso de uma linha de trem depois que a carga já havia sido colocada nesse tem.

Essas faltas resultam em dois tipos de erros: no primeiro tipo, a ação acaba não acontecendo, e o estado do mundo continua o mesmo de antes, com a excessão da passagem do tempo. Isso resulta na necessidade de substituir a ação atual por algum subplano de substituição (como no exemplo do terremoto).

No segundo tipo de erro, a ação é executada, e as pós condições são efetuadas no estado do mundo, exceto pela passagem do tempo, que é maior do que a prevista, devido

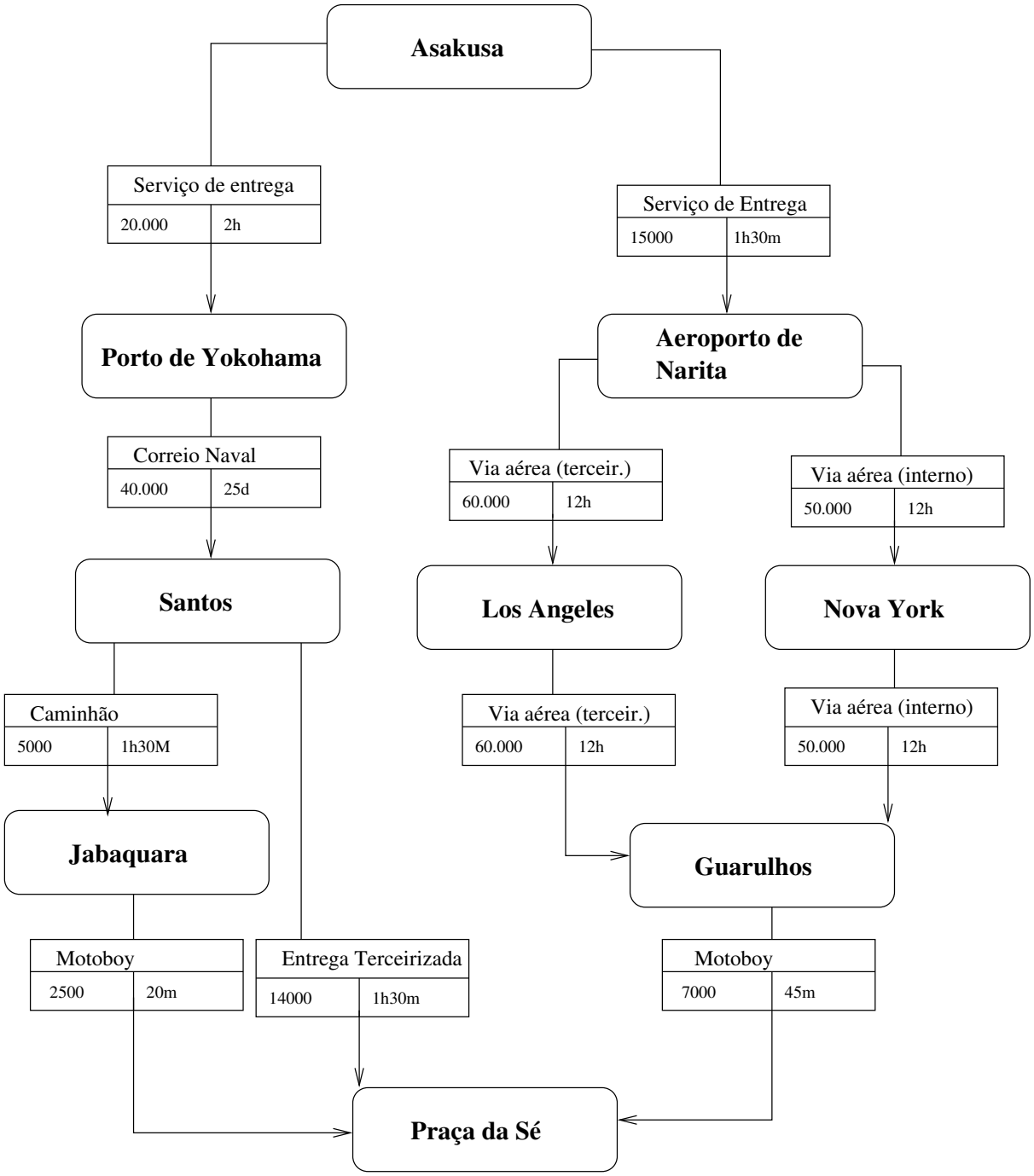

Figura 5.1: Exemplos de serviços (ações) para composição de web service

a um atraso (como no exemplo do avião). Devido a esse atraso, a pré-condição de limite de tempo da ação seguinte no plano não pode ser alcançada, resultando em uma falha no plano (p.ex.: devido ao atraso do avião, a carga perde o trem que a levaria para o seu próximo destino. Isso resulta na necessidade de substituir a *próxima* ação por um subplano.

No mundo real, erros em ações podem também fazer com que o sistema vá para um estado desconhecido. Por exemplo, uma ação de transporte pode entregar a carga para o endereço errado, ou extraviá-la. Na maioria das vezes, esses casos requerem um replanejamento completo, ou mesmo intervenção humana, por isso decidimos não abordá-los em nosso modelo.

5.2 Considerações para implementação

O sistema de substituição foi originalmente desenvolvido para atender a aplicações de robótica. Apesar de utilizar o planejamento da mesma maneira, deve-se tomar algumas considerações especiais ao se desenvolver o sistema de substituição para web services.

O algoritmo de reparo de planos, de uma maneira geral, não precisa ser modificado. Mas a modelagem de cada um de seus três componentes deve ser feita levando em conta algumas considerações. Estas considerações foram feitas baseada no modelo de web service que estudamos (o problema de transporte de cargas), que acreditamos ser representativo das classes mais comuns de web services, de acordo com a literatura (na seção 2.3 deste documento).

As principais diferenças que notaremos entre o web-service e um sistema robótico são: a independência dos diversos dispositivos que geram ações, tanto entre si, como para com o componente controlador; o maior nível de paralelismo no sistema; um ambiente dinâmico influenciando os serviços prestados e uma restrição mais severa ao tempo utilizado por cada ação. Cada uma dessas características e suas consequências serão mais estudadas abaixo.

Independência dos Serviços

Uma das características mais importantes dos web services é a sua independência e heterogeneidade. Um compositor de serviços, idealmente, poderá escolher em um mesmo plano serviço oferecidos em lugares geograficamente distantes, e, mais importantemente, pertencentes a empresas diferentes. Diferentemente de um planejador para um robô, onde geralmente todos os dispositivos são fixos e sob o comando de um mesmo controlador.

Dessa maneira, ao escolher um serviço para, por exemplo, transportar um produto através de uma determinada rodovia, o compositor pode ter diante de si a opção de

diversas empresas diferentes, oferecendo serviços similares. Isso significa que teremos uma maior redundância direta, o que gera planos de substituição mais simples (uma ação por uma ação). Por exemplo, a substituição de um dado serviço passa a ser o mesmo serviço sendo oferecido por uma empresa concorrente.

Essa característica, porém, também torna o diagnóstico dos "dispositivos" (serviços) mais difícil. No caso de serviços oferecidos por terceiros, o compositor do web service não vai ter um acesso direto ao estado de disponibilidade daquele serviço, como um controlador robótico tem em relação aos seus dispositivos. A informação de disponibilidade é feita fazendo-se uma consulta ao próprio serviço. Além disso, o compositor do web service não será necessariamente informado quando houver uma falha no serviço, de maneira que o componente de seleção de substituição deve verificar, a cada ação, o estado daquele serviço em particular.

Podemos imaginar casos em que essa informação não é totalmente confiável por má fé. Para evitar esses casos, uma maneira de observar indiretamente a situação real do serviço se faz necessária para aplicação da substituição em aplicativos reais.

Paralelismo no sistema

Como cada ação é realizada por um "dispositivo" (serviço) independente, web-services costumam apresentar um grau muito grande de paralelismo.

Podemos utilizar esse paralelismo para abrir uma nova série de possibilidades de substituição. No nosso exemplo de sistema de transporte de carga, caso haja uma falha em alguma ação de transporte, ao invés de realizar a substituição por um único subplano, como normalmente acontece, podemos escolher dois subplanos, dividir a carga a ser realizada entre eles, e realizar esses subplanos em paralelo.

Isso normalmente será útil quando as substituições disponíveis não atenderem, individualmente, os requisitos de disponibilidade de carga. Se o conjunto de substituições disponíveis atender a esse requisito, separamos a carga em múltiplas partes, e executamos as substituições simultaneamente.

Ambiente dinâmico

Num robô, podemos esperar que a maior parte do dinamismo do sistema esteja concentrada no ambiente. Ou seja, esperamos que os dispositivos presentes no robô ajam, dentro de um certo erro, de uma maneira previsível. Porém, no caso do web-service o conjunto de "serviços" é disperso, independente e descentralizado. Os serviços não respondem diretamente à entidade realizando a composição.

Devido ao dinamismo nos serviços, um serviço falho pode voltar a funcionar, também sem avisar o compositor. Por exemplo, suponhamos que um caminhão quebre na estrada,

mantendo a linha que ele supria indisponível. Os próprios funcionários da companhia que provê aquele serviço procurarão restabelecer a sua funcionalidade, e ao retornar do estado de falha, o serviço pode não conhecer todos os web services que o marcaram como falho. Outra situação semelhante é aquela na qual um novo serviço é estabelecido na rede, e se torna disponível, enquanto vários compositores de serviços estão no meio de suas atividades.

A consequência desse dinamismo nos serviços que são utilizados como ações pelo planejador é que, os componentes de diagnóstico e geração de substituições têm que ter um papel mais ativo do que o descrito no capítulo 3.

Em particular, a componente de diagnóstico não é mais informada diretamente quando há mudanças na situação dos dispositivos (podendo mesmo receber informações falsas, como sugerido), devendo portanto ativamente procurar a detecção de mudanças nestes. Além disso, a situação de "falho" se torna temporária, já que os serviços com falha vão procurar ativamente corrigir a falha por si mesmos.

Já o gerador de substituições, que normalmente gerava as substituições offline, e não agia durante as execuções da tarefa, agora será requisitado para reconstruir a tabela de substituição a cada vez que for detectado o oferecimento de um serviço que não existia antes. Usando o exemplo da seção anterior, se uma nova empresa for criada, provendo transporte aéreo diretamente entre Narita e Guarulhos, a tabela de substituição deverá ser atualizada para refletir a nova opção de substituição para as outras ações, e para criar as substituições para a nova ação. Ou seja, durante a execução de uma (ou várias) requisições.

Tempo como fator restritivo

Nas substituições para o robô simulado, a *degradação graciosa* que se obtinha ao se substituir uma ação em geral era um subconjunto de ações mais longo (mais demorado).

No caso do serviço de transporte de carga, a substituição por um subplano mais demorado na maioria das vezes não é possível. Uma vez que uma pré condição importante dos serviços são o horário de partida, substituir uma ação por outra mais demorada pode significar que a carga chegará após o horário de partida para a próxima ação, podendo causar um efeito em cascata que tornará o plano inteiro inválido.

Ao invés disso, escolhemos como critério de *degradação graciosa* o custo (em dinheiro) da ação. Ou seja, o plano procura minimizar o custo escolhendo os serviços mais baratos para cada ação. Se um desses serviços falha, as substituições disponíveis consistem de um serviço (ou combinação de serviços) equivalente com um custo maior. Assim, se um serviço de entrega por caminhão a longa distância falhar, no último caso podemos enviar a carga pelo serviço aéreo equivalente.

Obviamente, não podemos "de repente" requisitar ao cliente um valor total maior do que o inicialmente pedido pelo serviço composto. Assim, na composição do plano, o planejador deve incluir uma margem de segurança que leve em conta possíveis substituições de ações (essa margem poderia ser distribuída entre diversas requisições, de acordo com dados estatísticos sobre ocorrência de falhas). Se o total gasto com substituições for maior que um certo valor, um replanejamento total das ações restantes, ou invocação de um operador humano se faz necessário.

Capítulo 6

Conclusão e Direções Futuras

Neste trabalho estudamos a técnica de substituições de ações em um plano. A idéia para essa técnica se originou de um trabalho semelhante na área de desenvolvimento de processadores, e visava prover novos métodos de aumentar a autonomia de aplicações baseadas em planejamento.

Propomos as características básicas de um sistema de substituição, e como esse poderia ser utilizado para reparo de planos, e diagnóstico planejado. Estudamos a aplicação dessas técnicas nas áreas de robótica e de web services, escrevendo propostas de implementação para ambas.

Procuramos, neste trabalho fazer um estudo amplo das possibilidades da substituição de ações, uma vez que esse tipo de proposta ainda não havia sido estudado na área de reparo de planos. Algumas dessas idéias, especialmente na área de geração da tabela de substituição, foram observadas e, apesar de demonstrarem desafios interessantes de serem estudados, não puderam ser abordadas de maneira adequada durante o período de pesquisa.

Aqui introduziremos duas dessas idéias, que podem ser usadas para continuar o desenvolvimento do trabalho em sistemas de substituição. Descreveremos brevemente cada um dos problemas, e algumas idéias que podem ser usadas como direções iniciais de estudo.

6.1 Substituição de múltiplas ações

No nosso trabalho, sempre admitimos a idéia de que uma ação é substituída por um conjunto de ações. Na teoria, isso não é a única possibilidade, pois uma substituição pode se dar entre dois subplanos equivalentes, tendo cada subplano um número qualquer de ações. Porém, existe uma série de problemas para a implementação da substituição múltipla.

Primeiramente, há o problema da seleção da ação a ser substituída. Quando temos

apenas uma ação a ser substituída, essa ação *tem* que ser a ação passível de erro. Quando passamos a admitir a substituição de n ações de uma vez, essas n ações podem estar distribuídas de qualquer maneira antes e depois da ação problemática (ou mais de uma ação problemática, intercaladas por um número arbitrário de ações normais).

Em segundo lugar, a ocorrência de uma sequência específica de ações dentro de um plano é muito menos provável que a ocorrência de uma única ação qualquer, e admite uma variedade maior de resultados (pós condições), aumentando consideravelmente o número de entradas na tabela de substituição.

Porém, abordando-se esses problemas, vemos também algumas características que tornam a substituição de múltiplas ações interessante. Quando retiramos a restrição de substituir uma única ação de cada vez, aumentamos a flexibilidade do sistema. Isso ocorre quando duas (ou mais) ações ocorrem frequentemente juntas, mas não separadamente. Por exemplo, a garra do robô simulado descrito no projeto. As ações "abrir" e "fechar" da garra não costumam ocorrer sem a companhia de um movimento do braço. Quando se substitui uma dessas ações, o movimento acompanhante do braço tornou-se inútil, e poderia ser descartado juntamente com a ação da garra durante a substituição.

Uma outra utilidade para substituições múltiplas é quando temos a falha de uma ação que ocorre repetidas vezes em um plano. Em geral os subplanos de substituição possuem ações no seu início ou no seu final para corrigir mudanças em pós condições secundárias, mas que poderiam ser suprimidas se uma série de substituições semelhantes fossem executadas. Utilizando esses subplanos um a um, as ações de correção se acumulam, diminuindo a eficiência do sistema. Um exemplo disso seria uma ação em zigue-zague para substituir um movimento em linha reta.

Duas abordagens imaginadas para este problema são a utilização de linguagens formais, e um período de treino para a criação da tabela de substituição. A primeira abordagem sugerida é modelar o sistema como uma gramática formal, sendo cada ação um símbolo da gramática, e utilizar técnicas de busca de padrões em linguagens formais para ajudar na decisão de quando substituir. A segunda abordagem é realizar uma execução de treinamento com o sistema, e fazer uma análise da frequência de ocorrência de grupos de ações, para determinar quais sequências são mais vantajosas de serem substituídas. Tais sequências são então tratadas, para todos os efeitos, como se fossem ações únicas.

6.2 Representação resumida da tabela de substituição

Outro assunto que surge a partir do estudo dos sistemas de substituição é a representação resumida da tabela de substituição. Atualmente a tabela é uma estrutura que, para cada par de pós condições e pré condições equivalentes a uma ação da aplicação, lista os subplanos de substituição possíveis, um por um, cada subplano descrito com todas as

suas ações. Para sistemas complexos, o número de ações e de substituições pode ser tal que a manipulação das informações da tabela de substituição se torna muito custosa. é interessante buscar novas maneiras de representação para lidar com esses casos.

De uma maneira geral, o problema da representação resumida pode ser abordado por dois ângulos diferentes: Por um lado estuda-se a redundância das informações contidas na tabela, com o intuito de reduzir dados repetidos, sem perder informação de substituição, ou velocidade de acesso. Por outro busca-se técnicas mais "diretas" de aceleração, como a implementação da tabela de substituição em hardware e técnicas de compressão

Ensaiamos um estudo da primeira abordagem a esse problema durante o projeto, como uma aplicação da teoria de grupos no problema da substituição. Definimos, para um problema de path planning, o sistema em questão como um grupo de translações em um plano, com o conjunto de ações como o conjunto de geradores nesse plano. Se pudermos definir esse grupo de maneira que ele tenha solução para o problema da palavra, podemos então buscar um problema da palavra inverso onde, a partir de dois membros do grupo e um conjunto de geradores encontramos várias palavras para a mesma transição. Assim, deixamos de armazenar a substituição inteira, e trabalhamos com geradores que sejam comuns a diferentes substituições, gerando o subplano final de acordo com a demanda.

Referências Bibliográficas

[1] Workshop planning for web services. http://www.isi.edu/info-agents/workshops/icaps2003-p4ws/program.html, 2003.

[2] W3c´s extensible markup language (xml) 1.0 (third edition). http://www.w3.org/TR/REC-xml/., 2004.

[3] Workshop on planning and scheduling for web and grid services. http://www.isi.edu/ikcap/icaps04-workshop/contents.html, 2004.

[4] T. Andrews. *Business Process Execution Language for Web Services Version 1.1.* http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.

[5] C. Aranha, A. Covic, and J. Wainer. High level techniques for self-repairing robotic systems. In *IV Encontro Nacional de Inteligencia Artificial*, 2003.

[6] D. Austin, W. Grainger, A. Barbir, C. Ferris, and S. Garg. Web services architecture requirements. http://www.w3.org/TR/2002/WD-wsa-reqs-20020819, August 2002.

[7] J. Barraquand and P. Ferbach. Path planning through variational dynamic programming. In *Proceedings of IEEE international conference on robotics and automation*, pages 1839–1846, San Diego, CA, 1994.

[8] A. Benso, S. Chiusano, and P. Prinetto. A self-repairing execution unit for microprogrammed processors. *IEEE Micro*, pages 16–21, september-october 2001.

[9] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[10] A. L. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[11] J. Blythe. Decision-theoretic planning. *AI Magazine*, february 1999.

[12] J. Blythe and M. Veloso. Analogical replay for efficient conditional planning. In *Proc. 15th national conference on Artificial Intelligence*, pages 668–673, 1997.

[13] R. Bohlin. *Robot Path Planning.* PhD thesis, Department of Mathematics, Goteborg University, Goteborg, Sweden, May 2002.

[14] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with and architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 997.

[15] F. Z. Brill, G. S. Watson, G. J. Ferrer, and W. N. Martin. The effective field of view paradigm: adding representation to a reactive system. *Engineering Applications of Artificial Intelligence*, 11:189–201, 1998.

[16] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[17] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–204, 1994.

[18] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 35:333–377, 1987.

[19] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, Colorado, April 2000.

[20] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl, 2001.

[21] A. Cimati, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, (147):35–84, 2003.

[22] CNN.com. Mars rover spirit develops wheel problem. http://www.cnn.com/2004/TECH/space/06/15/marsrovers.ap/, Junho 2004.

[23] T. O. S. Coalition. Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1B/owl-s.pdf, 2004.

[24] K. Currie and A. Tate. O-plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[25] N. Davies, D. Fensel, and M. Richardson. The future of web services. *BT Technology Journal*, 22(1), January 2004.

[26] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.

[27] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contigent execution. In *proc 2nd International Conf. on AI Planning Systems*, pages 31–36, 1994.

[28] C. Elkan. Incremental, approximate planning. In *Proceedings of the national conference on artificial intelligence*, pages 145–150, 1990.

[29] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[30] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2), 2002.

[31] G. J. Ferrer. *Anytime Replanning Using Local Subplan Replacement*. PhD thesis, University of Virginia, 2002.

[32] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proc. of the 2o International Joint Conference on Artificial Intelligence*, pages 189–208, London, England, September 1971.

[33] R. Fileto. *A Abordagem POESIA para a Integração de Dados a Serviços na Web Semântica*. PhD thesis, University of Campinas, 2003.

[34] R. P. Goldman and M. S. Boddy. Conditional linear planning. In *2nd International Conference on AI Planning Systems*, 1994.

[35] R. P. Goldman and M. S. Boddy. Expressive planning and explicit knowledge. In *proc. of 3rd international conference on AI planning systems*, pages 110–117, 1996.

[36] M. Gudgin, M. Hadley, J.-J. Moreau, and H. Frystyk. Soap version 1.2. http://www.w3.org/TR/soap12, July 2001.

[37] K. Gupta and A. P. del Pobil. *Practical Motion Planning in Robotics*. John Wiley, West Sussex, England, 1998.

[38] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In *Proceedings of the first international conference on autonomous agents*, pages 363–370, February 1997.

[39] M. P. Herlihy and J. M. Wing. Specifying graceful degradation. *IEEE Trans. Parallel Distrib. Syst.*, 2(1):93–104, 1991.

[40] Y. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM computer surveys*, 24(3):219–291, 1992.

[41] L. Joskowicz and R. H. Taylor. Computers in imaging and guided surgery. *Computing in Science and Engineering*, 3, 2001.

[42] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proc; of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 318–325, Stockholm, Sweden, July/August 1999.

[43] L. Kavraki and J. Latombe. Randomized preprocessing of configuration space for fast path planning. In *proceedings of IEEE international Conference on Robotics and Automation*, 1994.

[44] K. Kotay, D. Rus, M. Vona, and C. McGray. The self reconfiguring robotic molecule. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1998.

[45] N. Kushmerick, S. Hanks, and D. S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[46] U. D. Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *Proc. AAAI´02*, 2002.

[47] J. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[48] S. M. Lavalle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26, 2000.

[49] S. M. Lavalle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE transactions on robots and automation*, 14(6):912–925, 1998.

[50] M. L. Leuschen, I. D. Walker, and J. R. C. ro. Robotic fault detection using nonlinear analytical redundancy. In *IEEE International Conference on Robotics and Automation*, 2002.

[51] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, april-june 1997.

[52] D. Long and M. Fox. Progress in a.i. planning research and applications. *UPGRADE*, 3(5), October 2002.

[53] M. Mason. Kicking the sensing habit. *AI Magazine*, 14(1):58–59, 1993.

[54] D. McDermott. Estimated-regression planning for interactions with web services. In *Proc. of the 6th Int. Conf. on AI Planning and Scheduling*, Toulouse, France, 2002. AAAI Press.

[55] S. A. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, March/April 2001.

[56] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing web services on the semantic web. *VLDB Journal*, 12:333–351, 2003.

[57] H. Muñoz-Avila, D. Aha, D. Nau, R. Weber, L. Breslow, and F. Yaman. Sin: Integrating case-based reasoning with task decomposition. In *Proc. of International Joint Conference on AI*, 2001.

[58] D. Nau, T. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20, 2003.

[59] D. Nau, S. Gupta, and W. Regli. Artificial intelligence planning versus manufacturing-operation planning: a case study. In *Proceedings of 14 International Joint Conference on AI*, pages 1670–1676. Morgan Kaufmann, 1995.

[60] C. Ortega and A. Tyrrel. Reability analysis in self-repairing embryonic systems. July 2000.

[61] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. *Algorithmic Foundations of Robotics*, pages 19–37, 1995.

[62] L. E. Parker. Allance: Am architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on robotics and automation*, 14(2):220–240, April 1998.

[63] J. Peer. A pddl based tool for automatic web service composition. In *Proc. Of the 2nd Workshop on Principles and Practice of Semantic Web Reasoning*. Springer Verlag, 2004.

[64] M. A. Peot and D. E. Smith. Conditional nonlinear planning. In *Proc. 1st Int. Conference on AI planning Systems,*, pages 189–197, 1992.

[65] L. Perdersen, D. Kortenkamp, and D. W. I. Nourbakhsh. A survey of space robotics. In *7th International Symposium of Artificial Intelligence, Robotics and Automation in Space*, 2003.

[66] S. R. Ponnekanti. Sword: A developer toolkit for web service composition. In *the Proc. Of the 9th International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002.

[67] E. Prassler, A. Ritter, C. Shaeffer, and P. Fiorini. A short history of cleaning robots. *Autonomous Robots*, 9(3):211–226, 2000.

[68] L. Pryor and G. Collins. Planning for contigencies: A decision based approach. *Journal of AI Research*, 4:287–339, 1996.

[69] N. R. Fikes. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[70] J. S. R. Hull. Tools for design of composite web services. In *Proceedings of ACM SIG MOD*, June 2004.

[71] J. Rao and X. Su. Survey of automated web service composition methods. In *Proc. Of the 1st International Workshop on Semantic Web Services and Web Process Composition*, San Diego, USA, 2004.

[72] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT press, 2001.

[73] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2003.

[74] E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.

[75] M. W. Shapiro. Self-healing in modern operating systems. *ACM Queue*, 2(8), November 2004.

[76] T. Simeon, S. Leroy, and J. P. Laumond. Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE transactions on robotics and automation*, 18(1):42–49, 2002.

[77] B. Sleeper. The evolution of uddi. Technical report, The Stencil Group, Inc., July 2002.

[78] B. Smith, W. Millar, J. Dunphy, Y. Wen, P. Nayak, and M. Clark. Validation and verification of the remote agent for space-craft autonomy. In *Proc. of the IEEE Aerospace Conf.*, Snowmass, CO., 1999.

[79] S. Smith, D. Nau, and T. Throop. Computer bridge: A big win for ai planning. *AI Magazine*, 19(2):93–105, 1998.

[80] B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *ICAPS 2003 - Workshop on Planning for Web Services*, 2003.

[81] Y. Sugahara, T. Endo, H. ok Lim, and A. Takanishi. Design of a battery-powered multipurpose bipedal locomotor with parallel mechanism. In *Proc. of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2658–2663, Lausanne, Switzerland, October 2002.

[82] Y. Sugahara, T. Hosobata, Y. Mikuriya, H. ok Lim, and A. Takanishi. Realization of stable dynamic walking by a parallel bipedal locomotor on uneven terrain using a virtual compliance control. In *Intelligent Robots and Systems*, volume 1, pages 595–600, october 2003.

[83] A. Tate. Generating projects networks. In *Proceedings of the fifth International Joint Conference on Artificial Intelligence*, pages 888–893, 1977.

[84] L. A. Taylor. Pruning duplicate nodes in depth-first search. Technical report, University of California, October 1992.

[85] M. H. Terra, M. Bergerman, and R. T. a nd Adriano A. G. Siqueira. Controle tolerante a falhas de robôs manipuladores. *SBA controle e automação*, 12(2):73–92, 2001.

[86] R. Tinós and M. H. Terra. Fault detection and isolation in robotic manipulators using a multilayer perceptron and a rbf network trained by the kohonen's self-organizing map. *Revista Controle & Automação*, 12(1):11–18, 2001.

[87] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *the IEEE 3rd International Conference on Web Services*. IEEE, July 2004.

[88] A. Tyrrell. Computer know thy self!: A biological way to look at fault tolerance, 1999.

[89] M. L. Visinsky. Fault detection and fault tolerance methods for robotics. Master's thesis, Rice University, Dezembro 1991.

[90] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker. Robotic fault detection and fault tolerance: a survey. *Reliability Eng. and System Safety*, 46:139–158, 1994.

[91] D. H. Warren. Generating conditional plans and programs. In *Proc. AISB summer conference*, pages 344–354, 1976.

[92] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of experimental and theoretical artificial intelligence*, 7(1):197–227, 1995.

[93] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating daml-s web services composition using shop2. In *the proc. of the Second International Semantic Web Conference*, 2003.