PLANEJAMENTO E EXECUÇÃO UTILIZANDO AGENTE POLIMÓRFICO DINÂMICO

MICHAEL VIDIGAL, LEONARDO M. HONÓRIO, LUIZ E. SOUZA

Universidade Federal de Itajubá
Instituto de Engenharia de Sistemas e Tecnologia da Informação
CRTI - Centro de Referência em Tecnologia da Informação
E-mails:{michael, demello, edival}@unifei.edu.br

Abstract—This paper addresses the problem of finding the minimum set of resources and agents. Each one with has its own characteristics to plan and execute specific problems. One of the current methodologies used to deal with this situation is modeling available equipments and problems to a PDDL domain. This domain is presented to an AI planner, which tries to find and execute a feasible operative plan. However, there are problems associated with this methodology; when a huge domain is presented to a planner, the plan quality can be compromised with, for instance, more resources being allocated than the necessary. To deal with scenarios like this, the present paper suggests an innovation at the Intelligent Agent theory, a new concept called Dynamic Polymorphic Agents (DPA). By on-line actualization, the DPA can search for available agents present in the system and being feed by information about their respective domains through a PDDL based protocol. Using this knowledge, the DPA uses a predefined heuristic model to dynamically assist the planner obtaining a plan with a reduced number of resources. It is also identifies and dispatches the necessary agents in order to carry out the plan. Another benefit of this methodology is that, due to its dynamic behavior, it is possible to change the original plan by the advent of an unexpected problem, even if the plan is being already executed.

Keywords— Polymorphic Agent, IA Planning, multi-agent systems.

Resumo— Este artigo é destinado ao problema de se encontrar um conjunto mínimo de agentes e recursos, (cada qual com suas próprias características) para se resolver um determinado problema utilizando técnicas de Planejamento Inteligente. Uma das metodologias utilizadas para lidar com este tipo de situação é modelar os agentes, recursos e problemas existentes em formato PDDL para que posteriormente um planejador de IA seja responsável por apresentar uma possível solução para o problema, e para que, finalmente, após a geração do plano, o mesmo seja executado. No entanto, existem alguns problemas associados à esta metodologia; ao se apresentar um domínio extenso a um planejador, a qualidade do plano pode ser comprometida o que geraria uma alocação desnecessária de alguns recursos, que poderiam estar disponíveis para a realização de outras tarefas. Para resolver cenários como este, o presente artigo sugere uma inovação na teoria de agentes inteligentes, um novo conceito chamado Agente Polimórfico Dinâmico (APD). Através de atualização on-line, APD pode procurar pelos agentes existentes no sistema que estão disponíveis, e pode também obter informações dos seus respectivos domínios através de um protocolo baseado na linguagem PDDL. Com este conhecimento, o APD utiliza uma heurística pré-definida para auxiliar dinamicamente o planejador, obtendo assim, um plano com um número reduzido de recursos. Após a determinação deste conjunto mínimo, os agentes se unem a outra entidade, o APD, que receberá o plano gerado pelo planejador e será capaz de executar cada função específica de cada um dos agentes através de polimorfismo, realizando, assim a tarefa inicialmente atribuída. Outra funcionalidade importante desta metodologia, é quando ocorre uma parada no sistema durante a execução do plano. Para resolver isto, é realizado um replanejamento para se buscar um novo conjunto de agentes que continue a execução da tarefa interrompida.

Palavras-chave— Agente Polimórfico, Planejamento em Inteligência artificial, sistemas multiagentes.

1 Introdução

No ambiente industrial as linhas de montagem devem ser flexíveis para se adequarem a uma série de produtos. Para isso, é necessário levar em consideração inúmeros fatores como; prazo de entrega, recursos disponíveis, estoque final e custos de produção, paradas de manutenção e restabelecimento de paradas não programadas.

A literatura apresenta uma série de técnicas para tratar o problema. Referências [Chan e Chung 2006, Zacharia e Apragathos 2005] utilizam algoritmos genéticos, já em [Shiue e Guh 2006] um algoritmo baseado em aprendizado é utilizado para determinar o escalonamento de células de manufatura. Técnicas como algoritmos evolucionários [Liu e Cheng 2003], *rough sets* [Pawlak 1988] entre outras, também são utilizadas. A grande desvantagem destas técnicas é a falta de padrão na modelagem e desenvolvimento dos planos, levando a soluções individuais dedicadas e de difícil integração.

Desenvolvida em paralelo com os métodos apresentados está uma técnica denominada Planejamento Inteligente (PI), baseada em grafos que utiliza máquinas de estado e metaheuristicas de busca em árvore que possibilitam solucionar uma série de problemas. Entretanto, para realizar a busca, é necessário definir quais são as ações reativas, ou seja, as ações que levam o sistema a alterar seu estado corrente. Outras considerações como, a pré-condição necessária para executar uma ação, sua pós-condição também podem ser modelados. A referêcia [Knoblock

1995] mostra um sistema de planejamento, execução e replanejamento interessante, porém utilizando um planejador criado para domínio específico. Todavia, com o objetivo de desacoplar o Planejador (ferramenta que realiza o PI) da modelagem do domínio do problema foi desenvolvido um padrão de linguagem denominado PDDL (*Plan Domain Definition Language*) [Ghallab 1998]. Desta forma é possível modelar qualquer domínio e, uma vez feito isso, pode-se utilizar qualquer planejador existente para resolver o problema. Neste trabalho a linguagem PDDL é utilizada na modelagem alguns agentes. Portanto, cada um destes agentes poderá ser modelado separadamente, e já estará pronto para ser inserido no contexto do sistema, independente de serem agentes robóticos ou de software.

Para ampliar este paradigma o presente trabalho sugere um nova metodologia denominada Arquitetura de Agentes Polimórficos Dinâmicos (AAPD). O AAPD é baseada na união de três arquiteturas já conhecidas; (a) linguagem PDDL como padrão de representatividade das ações de cada agente, (b) *Remote Procedure Call* (RPC) [Heindel e Kasten 1996] como forma de conexão entre os diversos agentes presentes e (c) reflexão orientada a aspectos, que é responsável por identificar as ações presentes em casa agente e enviar mensagens para seu acionamento, ou seja, para o agente remoto executar a ação desejada.

Basicamente, a arquitetura proposta funciona como o explicado a seguir. Cada agente possui um sistema computacional, representado por um objeto, que tem dados referentes às propriedades e métodos responsáveis pela execução de suas atribuições. Estes dados representam as meta-informações de um agente e é possível obtê-las através de uma técnica chamada reflexão [Sullivan 2001]. Com estas metainformações, e ainda utilizando reflexão, é possível executar qualquer método presente em um agente. Entretanto, saber quais as funções que um agente possui não significa ter conhecimento de sua utilidade. Para contornar este problema, para cada função existente em um agente, está associada uma definição de sua utilidade, modelada através da linguagem PDDL. Este conjunto de meta-informações, juntamente com o *host* do agente (endereço na rede),o estado atual e os domínios PDDL é denominado hiperdado. Um elemento importante na arquitetura proposta é o Agente Monitor (AM), que incorpora 4 funcionalidades. A primeira é identificar a presença de um novo agente, importar o seu hiperdado e montar uma tabela indexada (*Hash Table – HT*), fazendo a conexão do *host* deste agente com cada funcionalidade apresentada. Caso o AM identifique a saída de um agente, ele executa a remoção de suas funcionalidades da HT. Sua segunda funcionalidade é identificar um agente problema (AP), que é uma requisição de um usuário (ou uma variável de sistema representando um sensor) para que o sistema de agentes realize uma determinada tarefa. Depois de identificado o problema, o AM executa sua terceira funcionalidade, que é gerar e apresentar a um planejador um domínio de planejamento com todos os agentes presentes. Sua última funcionalidade é receber o plano gerado, utilizar a *HT* para identificar quais os agentes necessários para sua execução e gerar uma instância de um novo agente denominado Agente Polimórfico Dinâmico (APD). O *APD* tem a ordem das ações com os respectivos agentes e *hosts,* e é responsável por, via RPC (*Remote Procedure Call*), executá-las.

Para demonstrar esta proposta, o artigo está estruturado da seguinte forma; na seção 2 é feito um detalhamento de planejamento inteligente, na seção 3 é apresentado o paradigma de programação orientada a aspecto utilizado tanto na fase de reflexão quanto na de geração do Aspecto Funcional. Na seção 4 é apresentada a metodologia de AAPD, na seção 5 é apresentado um exemplo de utilização e, por fim, na seção 6 é apresentada a conclusão.

2 Planejamento Inteligente

Os Sistemas Inteligentes decidem por si próprios como executar um plano de ações dados os objetivos e as características do ambiente ou domínio em questão. As técnicas de Planejamento inteligente (PI) consideram linguagens, modelos e algoritmos para descrever e solucionar problemas que envolvam a seleção de ações para atingir um determinado objetivo. São várias as metodologias apresentadas na literatura para o desenvolvimento de um planejador, e já existem vários planejadores baseados em diferentes técnicas disponíveis para utilização e estudo.

A Linguagem padrão de planejamento é o PDDL [Ghallab 1998]. O objetivo da criação da linguagem foi além da padronização, uma maior facilidade de se comparar o desempenho dos planejadores além de aumentar a interoperabilidade entre eles

Os planejadores trabalham com adição e exclusão de átomos, que são os predicados instanciados. À medida que as ações vão sendo analisadas, é montada uma árvore de busca onde são adicionados e removidos os devidos átomos no espaço de estados. O espaço de estado em qualquer problema de planejamento não trivial é muito grande e computacionalmente difícil. Por esta razão, planejadores que utilizam esta técnica são dependentes de boas funções heurísticas para guiar a busca.

Uma dificuldade encontrada durante a fase de testes deste trabalho foi com relação ao número de ações geradas pelos planejadores. Durante a fase de testes após a geração de um plano de ações, alguns agentes eram retirados do sistema, para verificar o comportamento dos planejadores. Foram constatados casos onde o planejador gerava uma resposta não esperada, onde o número de ações após a retirada de agentes era menor do que no caso anterior. Isto se deve aos planejadores, que tendo que percorrer uma extensa árvore de busca, nem sempre conseguem atingir uma solução ótima. Além disso, existe o problema de alocação desnecessária de agentes, onde um agente é capaz de realizar determinada tarefa sozinho, porém o plano é gerado utilizando um outro

agente em paralelo com a mesma função: Ex.: Supondo que dois Agentes, 1 e 2, possuam uma única ação que é a de TransportarBloco (PosiçãoInicial, PosiçãoFinal). O Objetivo é transportar dois Blocos de uma posição a outra. A solução é encontrada em dois passos, porém dependendo do caminho que o planejador escolher, cada passo pode ser executado por um agente diferente. Sendo o processo seqüencial, dois agentes diferentes foram alocados para realizar uma tarefa que apenas um agente conseguiria realizar.

Para evitar estes problemas, foi desenvolvida neste trabalho, uma característica de busca direcionada onde o sistema irá buscar o menor número de agentes que tenham a capacidade de resolver o problema. Tal heurística segue o procedimento a seguir: Para a primeira tentativa de solução é apresentado para o planejador o menor conjunto de agentes capaz de resolver o mais simples dos problemas. Caso o problema não seja solucionado com estes agentes, uma varredura é feita onde a cada iteração um novo agente é adicionado. Esta varredura continua até que uma solução seja encontrada ou até não ter mais agentes disponíveis. Desta forma, apesar de não poder afirmar que a solução encontrada é ótima, pode-se garantir um numero baixo de agentes.

3 Usando Modelagem Orientada a Aspecto em Agentes Polimórficos

A Programação Orientada a Aspecto (AOP - Aspect-Oriented Programming) [Kiczales 1997] é um paradigma de programação que foi proposto com a intenção de lidar com problemas com alto índice de entrelaçamento entre componentes. AOP funciona decompondo o problema em partes para uma futura recomposição. O grande resultado deste paradigma está relacionado aos novos mecanismos de composição que diminui drasticamente o número de dependência entre os componentes. Em AOP, problemas são decompostos e modelados de acordo com o conhecimento do domínio. Algumas partes do modelo gerado se fundem com outras partes através de mecanismos de orientação a objetos – estes, são componentes normais-, mas algumas partes requerem mecanismos mais avançados – estes são chamados componentes aspectuais do problema (ou simplesmente Aspectos). Neste trabalho, é utilizado um aspecto, representado por um agente polimórfico, que é desenvolvido separadamente e não possui qualquer ligação com qualquer outro agente regular. Na fase de composição os agentes são reunidos por um processo chamado *weaving*, resultando no sistema final. O Processo de *weaving* pode ser implementado de duas diferentes maneiras: a) estaticamente e b) dinamicamente, quando as funcionalidades dos componentes são juntadas, durante o tempo de execução, às funcionalidades de um dado aspecto.

O que diferencia um aspecto de um componente normal é a sua composição com o resto do sistema. Um componente regular representa um agente do sistema e tem acesso a suas funcionalidades internas. Um aspecto seleciona um conjunto de agentes presentes no modelo e cria conexões externas entre eles. Estas conexões entre o aspecto e o resto do sistema são definidas em um terceiro elemento, um *binder*. O *binder* usa meta-informações sobre os componentes, tais como suas variáveis e métodos, para compô-los. Desta forma as conexões por si próprias se tornam extremamente programáveis ao invés de serem definidas estaticamente dentro dos componentes.

A composição do sistema é feita através de um *weaver*, um elemento que através das instruções de ligação entre os componentes, reúne estes diferentes componentes. Neste trabalho, foi implementado um sistema com um *weaver* dinâmico com capacidade de *reflection*. *Reflection* é uma técnica suportada pelas linguagens de programação mais modernas, e que permite que um agente inspecione e manipule a si mesmo e/ou outros agentes. Usando a biblioteca *Reflection* o programa tem acesso aos metadados e podem fazer uso desta informação.

Uma propriedade que merece atenção nesta metodologia é que os componentes presentes no sistema são variáveis por si mesmos. Outra vantagem desta metodologia é que se algum componente novo for adicionado ao sistema, o *weaver* tem automaticamente acesso a seus metadados e ele é automaticamente adicionado ao sistema sem qualquer modificação.

Utilizando-se dos benefícios da metodologia AOP é possível desenvolver um sistema multiagente altamente flexível para ser utilizado em resolução distribuída de problemas. A idéia é desenvolver componentes representando cada agente (dispositivos de manufatura) independente do cenário no qual irão trabalhar.

4 Arquitetura de Agentes Polimórficos Dinâmicos

A arquitetura desenvolvida por este artigo apresenta 7 elementos; hiperdado, agentes regulares/reativos (componentes do sistema/dispositivos de manufatura), agente monitor, agente planejador, aspecto funcional, agente problema e a entidade chamada agente polimórfico dinâmico. A importância e funcionalidade de cada um são explicadas a seguir.

O primeiro conceito a ser definido é o de Agente Monitor (AM). Em uma estrutura de vários agentes não homogêneos, o AM é responsável por monitorar o sistema, centralizando o conhecimento das funcionalidades de cada componente e, a cada nova entrada ou saída de, reestruturar o sistema com o conhecimento atualizado na forma de dados.

Para transformar os dados em informações, um artifício semelhante ao metadado é utilizado. Da mesma forma que métodos têm metadados para a sua descrição, é necessário desenvolver outro conjunto de dados, denominados por este trabalho de *Hiperdados*. Logo, para cada método desenvolvido em um agente, deve-se desenvolver um hiperdado contendo todas as informações necessárias para auxiliar o AM

na escolha dos Agentes necessários. Informações sobre parâmetros, precondições e efeitos estão contidas nos hiperdados (modeladas no formato PDDL) além das informações sobre o host do agente (endereço na rede) e de seu estado atual (disponibilidade).

Com este conjunto de hiperdados o AM fica conhecendo todas as funcionalidades associadas aos agentes presentes no sistema. Porém, ainda dispondo de todo este conhecimento, o AM não tem capacidade para solucionar um problema por mais simples que seja. Tomando como objetivo ter uma entidade que seja capaz de, dado um problema, determinar como solucioná-lo e como implementar esta solução, é necessário definir duas novas entidades; Agente Problema e Agente Planejador.

O agente problema é responsável por apresentar ao AM qual o estado final desejado do sistema. Isto é feito obtendo informações do ambiente e convertendo-as no formato de problema PDDL. O agente problema tem ainda uma função de guardar o estado corrente do sistema. À medida que o plano de ações é executado, vão sendo criados estados correntes correspondentes a cada passo da ação. A razão disto está no fato de que se a execução do plano, por algum motivo, tiver que ser interrompida, um novo planejamento poderá ser executado tendo como estado inicial o estado corrente no momento da paralisação. Se durante a paralisação, ocorrer inclusão ou exclusão de agentes ou recursos, o Agente Problema têm a função de adaptar o estado inicial incluindo e/ou excluindo características pertencentes a estes Agentes.

O Agente Planejador pode ser uma entidade interna ao sistema onde, neste caso, é tratado como uma funcionalidade do próprio AM, mas também pode ser uma entidade externa, como foi implementado neste trabalho. Para o planejamento, propriamente dito, dos problemas no formato PDDL foi utilizado o planejador FF [Hoffmann 2001].

Porém, a solução adotada por este trabalho possibilita utilizar qualquer planejador externo. Isto possibilita aproveitar toda a funcionalidade da linguagem PDDL, e a eficiência e "know-how" dos planejadores existentes. Com isso, qualquer domínio modelados em PDDL, já estará pronto para ser inserido no contexto dos agentes pertencentes ao sistema.

O AM com todas as informações necessárias para determinar a forma de se solucionar um problema (domínio PDDL e informações sobre funcionalidades de cada agente), se comunica com o agente Planejador para que este gere um plano de ações para cumprir um objetivo. A geração deste plano é um processo iterativo entre o AM e o Planejador.

Como discorrido anteriormente, apresentar muitas funcionalidades para um planejador pode gerar resultados incoerentes, desta forma, foi adotada a forma iterativa já apresentada aonde, através de uma busca direcionada, o AM vai apresentando combinações crescentes de agentes para o Agente Planejador, e este tenta encontrar uma solução. Agentes vão sendo acrescentados até que uma solução seja encontrada ou até que se utilizem todos os agentes possíveis, e o planejador indique que não existe solução. Caso encontrada, a solução apresenta a descrição de todas as funcionalidades, e por sua vez os agentes que as contém, necessários para sua execução. É importante ressaltar que o custo das execuções repetidas de planejamento não é tão elevado. Isto acontece pois durante a extração da heurística de busca do planejador FF (que extrai esta heurística do próprio domínio e antes da execução da busca em árvore [Hoffman 2000]), a determinação da não-possibilidade de se encontrar uma solução é satisfatoriamente rápida. Porém, este processo de busca iterativa pelo mínimo conjunto de agentes poderia ser evitado caso o planejador utilizado conseguisse garantir um resultado ótimo. Além disso, com os novos recursos da linguagem PDDL como otimização [Younes 2005], custo de ações [Nyblom 2005], preferências [Gerevini e Long 2005], entre outras, poderia se definir custos e prioridades para cada agente, o que levaria a uma melhora qualitativa na escolha dos agentes.

Após a geração do plano e com as funcionalidades e seus agentes identificados, o AM através de um processo de weaving, combina estes agentes com o aspecto funcional, gerando uma nova entidade denominada Agente Polimórfico Dinâmico (APD). Um APD pode ser considerado uma simbiose entre agentes, onde, elementos com características diferentes podem se unir e, como se fossem uma única entidade, realizar funções que nenhuma das entidades básicas seria capaz. Este novo agente, com inteligência própria e dedicada à resolução de uma tarefa, se desagrega do sistema de agentes até que a tarefa seja realizada.

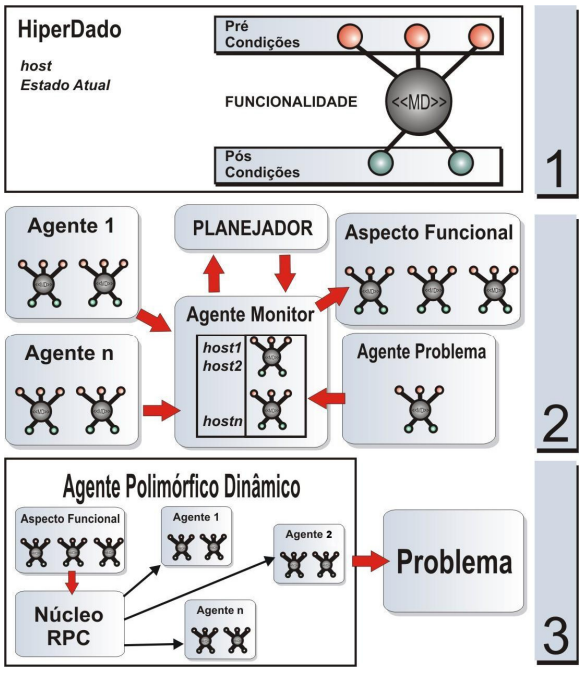

Figura 1 - Arquitetura do Sistema

O gráfico da Figura 1 ilustra os conceitos acima descritos onde: Em 1 se tem a representação do hiperdado, representado por um conjunto de pré-condições, uma funcionalidade, um conjunto de pós-

condições ou efeitos, o endereço de rede e o estado atual do agente; Em 2, o agente planejador recebe os hiperdados dos agentes de 1 a n e, dado um problema, realiza um planejamento identificando quais as funcionalidades necessárias para solucioná-lo e gera um aspecto funcional; Em 3 o aspecto funcional se agrega aos agentes de 1 a n através de RPC gerando um novo agente, o APD, especializado em resolver os desafios apresentados pelo agente problema.

5 Aplicação e Resultados

Para validar a metodologia, um cenário didático de manufatura foi utilizado [Honório 1994]. O Ambiente proposto é mostrado na figura 2.

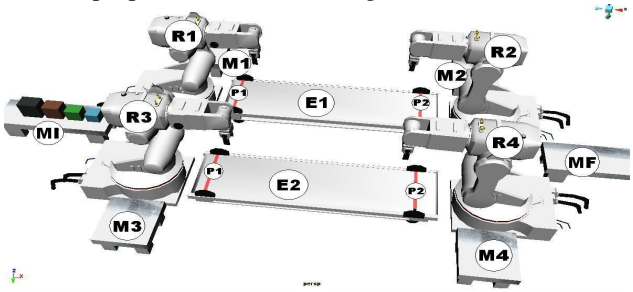

Figura 2 – Ambiente de Aplicação.

Este cenário é composto de: 2 esteiras transportadoras (E1, E2) com posições internas (P1, P2) que possuem a ação de transportar um único bloco da posição P1 até P2; 1 mesa de entrada de blocos (MI) e outra de saída de blocos (MF), ambas com posições internas e controladas por um único agente que possui a ação de mover a fila de blocos presentes nas mesas; 4 robôs manipuladores (R1, R2, R3, R4), que possuem ações de Pegar ou Soltar um bloco em determinada posição; 4 magazines armazenadores (M1, M2, M3, M4), cada um dedicado à um dos robôs e com capacidade de armazenar 1 único bloco. O problema é, dada uma ordenação específica dos blocos de entrada, encontrar uma forma de colocá-los na seqüência correta na mesa de saída.

O problema apresentado ao sistema envolve a existência de 4 blocos na ordem (1, 2, 3, 4) que devem ser transportados para a mesa final, e formarem uma nova ordem (4, 3, 2, 1).

O plano de solução do problema apresentado utiliza os seguintes Agentes e Recursos: Mesas Inicial e Final - Robôs: 1, 2 e 3 - Esteira 1 - Magazines 1 e 2, e será mostrado na forma de Diagrama de Seqüência na Figura 3. O nome da ação Transportar foi abreviado para "Tr" por questões de espaço no gráfico.

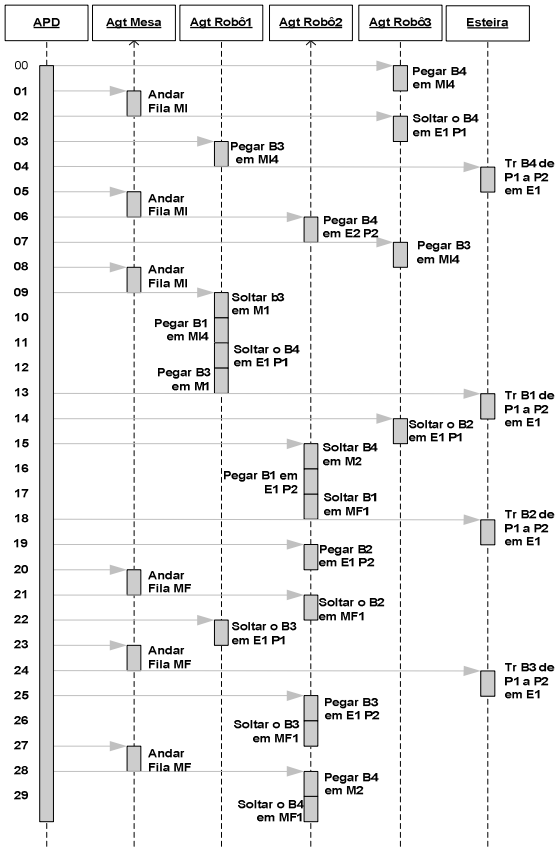

Figura 3 – Diagrama de Seqüência do plano

Para o teste da dinâmica do sistema, foi realizada uma simulação de parada no passo 6 do plano mostrado. O Agente Robô 2 e todos os magazines foram retirados do sistema e uma nova solução foi encontrada com um diferente conjunto de agentes: *Mesas Inicial e Final - Robôs: 1, 3 e 4 - Esteiras 1e 2;* Nota-se que o agente Robô 4 entrou no sistema juntamente com a Esteira 2, gerando um plano diferente e com um diferente contexto de agentes . O procedimento de solução do sistema é mostrado abaixo:

1. O Agente Monitor identifica os hiperdados dos agentes presentes no sistema e verifica quais dos agentes estão disponíveis para a realização de suas tarefas.
2. O AM monta um mínimo de agentes necessários para realizar um tarefa simples (Ex.: 1 Bloco).

2.2. À medida que a solução não é encontrada (Caso este passo seja executado imediatamente após o passo 5), é acrescentado mais um agente;

3. Se não houve acréscimo no número de agentes mínimos após o último loop, significa que o sistema não pôde encontrar uma solução. Caso contrário, o Agente Problema instancia o estado inicial e objetivo desejados, com base nos Agentes Mínimos atual (com o acréscimo de agentes);
4. O Agente Planejador monta o domínio completo do sistema com base também nos Agentes Mínimos corrente e, juntamente com o problema montado pelo Agente Problema, tenta encontrar um plano de ações e identifica os agentes necessários para sua execução.

5. Se a solução não for encontrada, repete-se o passo 2.2 . Caso contrário, o Agente Monitor gera o Aspecto Funcional

6. O aspecto funcional e os agentes envolvidos se separam do sistema de agentes para criar um novo APD para resolver o problema proposto. Este APD começa então a execução das tarefas.

7. Se, por algum motivo, durante a execução das tarefas, existir uma interrupção da mesma, o Agente Monitor guarda o estado corrente do ambiente, que se tornará o estado inicial do problema.

8. Se a execução for reiniciada, volta-se ao passo 1, porém com uma informação ao Agente Problema de que o estado inicial do sistema é aquele guardado durante a interrupção da execução.

6 Conclusão

Este artigo propôs uma arquitetura inovadora de agente inteligente. Utilizando um framewok que trabalha através de reflexão, foi possível gerar uma arquitetura apresentando uma estrutura de hiperdados capaz de descrever as funcionalidades dos agentes, um monitor responsável por identificar e informar ao planejador quais as funcionalidades disponíveis e um aspecto funcional gerando uma entidade polimórfica durante o tempo de execução com inteligência dedicada à resolução de um determinado problema, e que executa as ações de cada agente através de polimorfismo via RPC. O cenário utilizado para validação do método foi um ambiente de manufatura virtual didático, porém outros domínios de aplicações reais e mais complexas estão sendo estudados. O resultado apresentado corrobora a metodologia ao conseguir solucionar o problema através de uma entidade nova, não existente antes da geração do aspecto funcional, e se mostrou capaz de, no advento de uma parada, o conseguir retomar a execução da tarefa mesmo que seja preciso um novo conjunto de agentes. O sistema se mostrou extremamente flexível e com grande poder de aplicabilidade.

Agradecimentos

Parte deste trabalho foi apiado pela CT-Info/CNPq: 400842/2003-3.

Referencias Bibliográficas

F.T.S. Chan, S.H. Chung, P.L.Y. Chan, G. Finke and M.K. Tiwari, Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach, Robotics and Computer Integrated Manufacturing, 2006, 22, pp. 493-504

Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia (2005)

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL---The Planning Domain Definition Language. AIPS-98 Planning Committee.

H. L. S. Younes, M. Littman, D. Weissmann, and J. Asmuth. The first probabilistic track of the International Planning Competition. In Journal of Artificial Intelligence Research, volume 24, pages 851-887, 2005.

Heindel, L. E., Kasten, V. A.: "Highly Reliable Synchronous and Asynchronous Remote Procedure Calls", IEEE, pp. 103-107, Maio 1996.

Hoffmann, J.: FF: The Fast-Forward Planning System. AI Magazine 22[3], AAAI Press, (2001) 57-62.

Hoffmann, J. A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm. In Proceedings of the Twelfth International Symposium on Methodologies for Intelligent Systems (2000)

Honório, L. M., Souza, L. E. e Rocha, L. C. (1994) Desenvolvimento e Simulação de Dispositivo Robótico Virtual, SUCESU2004, Florianópolis, Brasil.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C. e Lopes, C. V.: Aspect-Oriented Programming. In European Conference on Object-Oriented Programming, ECOOP'97, Springer-Verlag LNCS 1241, Finland (1997)

Knoblock, C.A.:Planning, executing, sensing, and replanning for information gathering', Proceedings of IJCAI, 1686--1693, (1995).

Min. Liu and Cheng. Wu, Scheduling algorithm based on evolutionary computing in identical parallel machine production line, Robot Comput Integrated Manufacturing 19 (2003) (5), pp. 401–407.

P. Th. Zacharia, N. A. Aspragathos, "Optimal Robot Task Scheduling based on Genetic Algorithms", Robotics and Computer - Integrated Manufacturing , 21, pp 67-79, 2005 .

Pawlak, Zdzislaw, Wong, S. K. M., Ziarko, Wojciech (1988): Rough Sets: Probabilistic versus Deterministic Approach. In International Journal of Man-Machine Studies, 29 (1) p. 81-95.

Per Nyblom. Handling uncertainty by interleaving cost-aware classical planning with execution. In Swedish AI Society Workshop, 2005.

Russel, K. e Norvig, P.: Artificial Intelligence, A Modern Aproach, Ed. Prentice Hall.

Sullivan, G. T.: Aspect-oriented programming using reflection and metaobject protocols, Communications of the ACM, v.44 n.10, p.95-97, Outubro 2001.

Y. R. Shiue and R. S. Guh, "Learning-Based Multi-Pass Adaptive Scheduling for a Dynamic Manufacturing Cell Environment," Robotics and Computer-Integrated Manufacturing, Vol. 22, No. 3, pp. 203-216, Jul. 2006.