3
Trabalhos Relacionados

Os trabalhos relacionados se enquadram em no grupo de trabalho relacionado à modelagem de SMA.

3.1.
Linguagens de Modelagem de SMA

Atualmente existem inúmeras linguagens de modelagem de SMA. Cada uma destas linguagens traz características de melhor se aplicarem em determinado escopo, isto será mais bem apresentado nas seções seguintes.

Dentro deste conjunto de linguagens de modelagem foram escolhidas para estudo as linguagens ANote, AUML e Tropos. O objetivo deste estudo é mostrar que as linguagens não conseguem representar as variabilidades existentes em alguns agentes de software e com isso desperdiçam uma flexibilidade que poderia ser projetada para o sistema. No anexo B encontra-se a modelagem do sistema CQPM com as três linguagens apresentadas, tal sistema será apresentado nas seções seguintes.

3.1.1.
AUML

A linguagem de modelagem AUML surgiu como um primeiro resultado da cooperação entre FIPA e OMG visando aumentar a aceitação dos agentes de software na indústria (Huget & Odell, 2004; Bauer et al., 2001). AUML traz uma extensão da UML tradicional para representar agentes de software (Poggi et al., 2002; Bauer et al., 2001). O ciclo proposto pela AUML se enquadra na fase de design do projeto.

A linguagem AUML traz uma rica representação de troca de mensagens através do seu diagrama de protocolo de interação, que estende os diagramas de seqüência e de estado (da UML) em vários pontos. O diagrama de protocolo é considerado o principal diagrama da AUML, pois através dele é possível

identificar os tipos de mensagens (padrão FIPA-ACL) que estão sendo trocadas entre os agentes o que facilita bastante o entendimento de como os agentes estão se comunicando no sistema. A figura 3 mostra um exemplo do diagrama de protocolo.

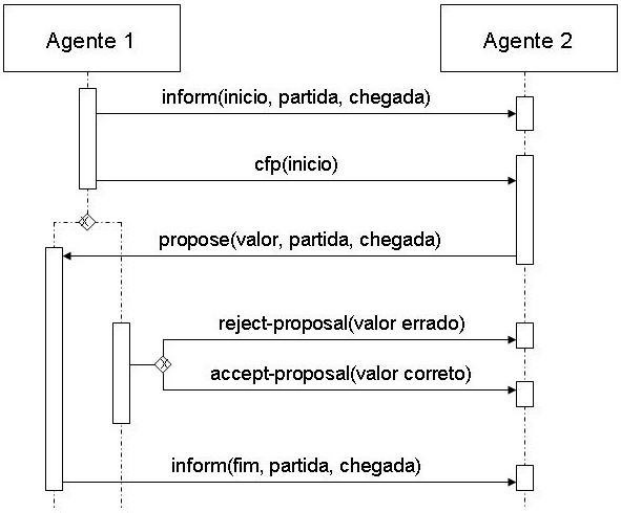

Figura 3. Diagrama de protocolo da AUML.

A AUML é composta de mais dois diagramas: o diagrama de classes e o de deployment. O diagrama de classes de agentes que representa um agente dentro do sistema, os protocolos de comunicação que serão utilizados por ele dentro do sistema, a lista de agentes que podem iniciar uma colaboração. Além disso, o diagrama apresenta uma descrição da ontologia que o agente pode utilizar durante a comunicação com outros agentes.

<<agent>> Nome do agente
Organization Document sub-society: Nome do agente
Protocols Descrever aqui os protocolos utilizados Ex. Initiates a FIPA-Contract-Net protocolo para responder ao usuário e adicionar requisições
Collaborators Aqui contém a lista de agentes colaboradores
Expertise Aqui deve estar descrita a ontologia que o agente em questão pode usar durante a comunicação com outros agentes.

Figura 4. Diagrama de classe da AUML.

O último diagrama da AUML é o diagrama de *deployment* ou de posicionamento, com este diagrama é possível dar ao desenvolvedor uma visão de como o sistema deve ser estruturado, mostrando qual a plataforma que esta sendo trabalhada, onde se localiza o servidor, etc. Este diagrama possibilita representar o conhecimento que um agente tem de um outro agente ou o conhecimento que um agente tem de um outro artefato utilizado no sistema através da *tag* <<*acquaintance*>>.

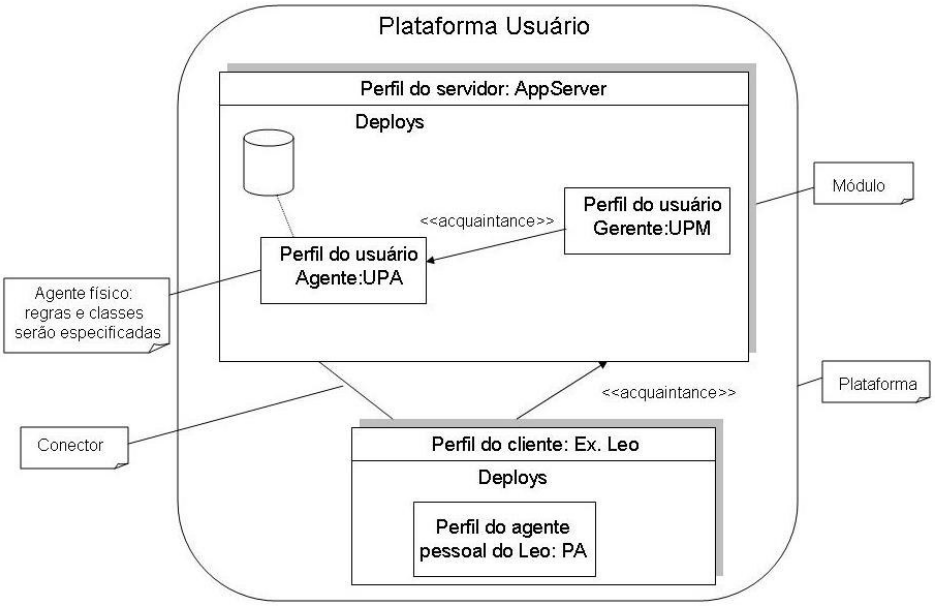

Figura 5. Diagrama de *deployment* da AUML.

3.1.2. Tropos

A metodologia Tropos é uma adaptação do modelo i* de Eric Yu (Giorgini et al., 2000). O modelo i* traz em sua representação cinco elementos principais (*actor*, *goal*, *task*, *resource* e *softgoal*). Tropos insere na sua representação o conceito de atores (agentes, papéis e posições), objetivos e atores de dependência geralmente utilizado na fase *early requirements analysis* da modelagem. Tal metodologia traz consigo quatro fases de desenvolvimento: *early requirement analysis* que se preocupa com o entendimento do problema através do estudo da configuração organizacional, *late requirement analysis* onde o sistema que deve ser provido é descrito no interior de ambientes operacionais, juntamente com suas qualidades e funcionalidades relevantes, *architectural design* onde a arquitetura

global do sistema é definida em termos dos subsistemas, interconectados através de dados, domínio e outras dependências, *detail design* onde o comportamento de cada componente é definido em mais detalhes; estes detalhes são representados pelo *social pattern* que tenta representar um padrão de soluções dentro de uma sociedade, semelhante à idéia de *design pattern* de orientação a objeto. Nesta fase o Tropos inclui a especificação da comunicação dos agentes e seus comportamentos para isto é sugerida uma adaptação de linguagens de comunicação como FIPA-ACL e extensões da UML como a AUML.

Os elementos oferecidos pelo Tropos são os seguintes:

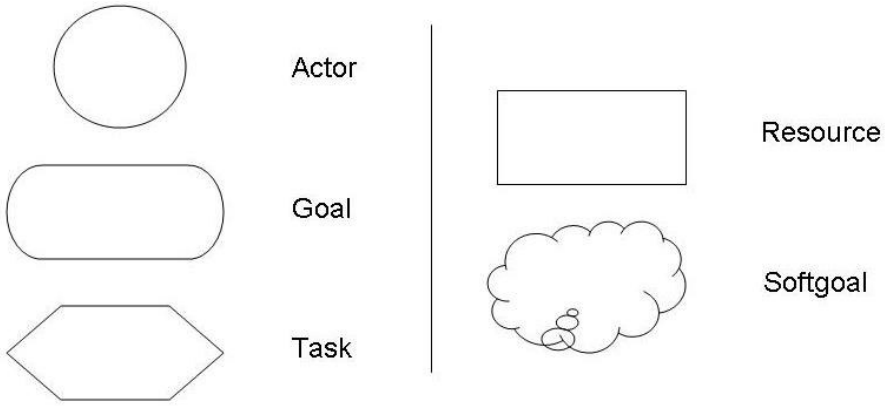

Figura 6. Elementos da notação i* utilizada pela metodologia Tropos.

A seguir pode ser vista a sintaxe possível durante a modelagem de uma estratégia para um agente (ator) dentro de um sistema.

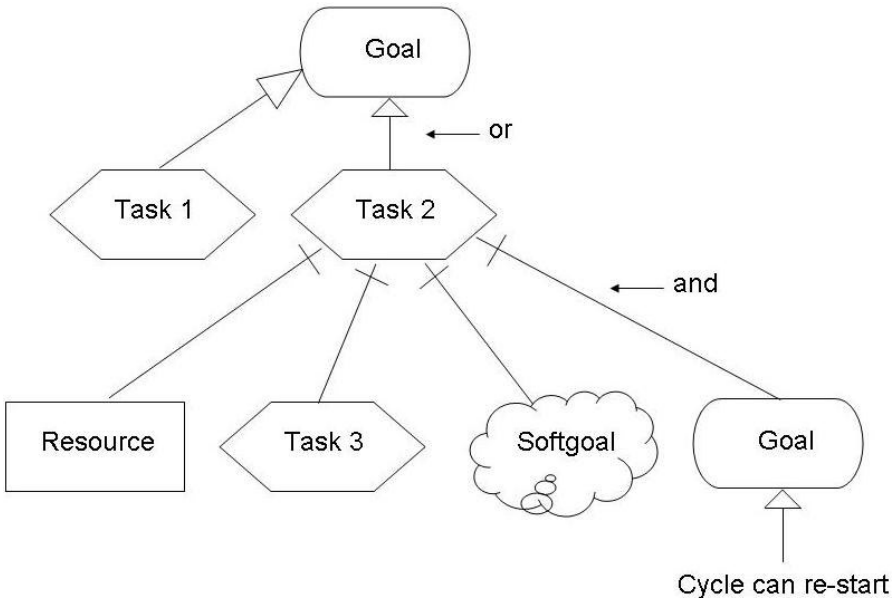

Figura 7. Modelagem parcial com o Tropos.

A metodologia Tropos é a que apresenta o maior ciclo com várias fases que se estendem desde as fases iniciais de elicitação de requisitos até as fases finais de projeto e implementação. A metodologia se baseia em duas idéias principais, a primeira é introduzir a noção de agente relacionando-os com as noções de planos e ações, as quais serão utilizadas em todas as fases de desenvolvimento de software, desde as analises iniciais até a implementação; a segunda é que o Tropos cobre as fases iniciais de analise de requisitos, o que permite um profundo conhecimento do ambiente onde o software que se quer fazer irá ser operado (Giorgini et al., 2000).

O meta-modelo apresentado no i* foi desenvolvido para ser utilizado com engenharia de requisitos, reengenharia de processos de negócios e modelagem de processos de software, já que com o uso do i* é possível determinar não apenas questões como "o que" ou "como", mas também o "porque" pedaços de software são desenvolvidos (Henderson-sellers et al., 2003).

Deste modo o modelo suporta uma refinada analise de dependência de sistemas e encoraga o tratamento uniforme de requisitos funcionais e não funcionais do sistema.

3.1.3. ANote

O ANote, assim como as demais linguagens citadas, visa dar um significado para especificação de soluções baseadas em agentes através de seus modelos. A linguagem traz uma abordagem baseada em orientação a objetivo, onde seus modelos são instâncias do meta-modelo conceitual (Choren & Lucena, 2003a) e a partir destes modelos são capturados os principais conceitos da linguagem ANote como objetivos, agentes, cenários, ações, mensagens, fonte (ou recursos) e organização (grupo de um ou mais agentes).

Figura 8. Meta-modelo do ANote.

O ANote traz consigo sete diagramas que possibilitam a modelagem dos SMA. Os diagramas do ANote são classificados em diagramas estáticos e os dinâmicos, onde os diagramas estáticos são os diagramas de objetivo (*goal*), agente (*agent*) e ambiente (*ontology*); os diagramas dinâmicos são compostos pelo de cenário (*scenario*), planejamento (*planning*), Interação (*interaction*). O diagrama de organização (*organizational*) não se em caixa nesta classificação já que é o responsável pela definição da estrutura do sistema que não é definida por agente de software.

Diagrama de objetivo é o diagrama onde são especificados os objetivos do sistema que são definidos como serviços e funcionalidades, um objetivo é representado por um retângulo com as pontas arredondadas, os objetivos são interligados através de setas que vão do objetivo mais especifico para o mais geral.

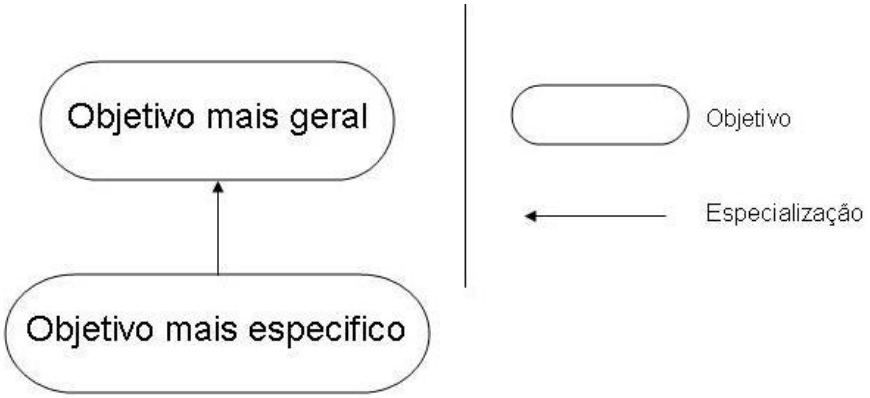

Figura 9. Elementos do diagrama de objetivo do ANote.

O diagrama de agentes (classe de agentes) especifica os agentes que existem no sistema e as relações entre tais agentes. O principal elemento neste diagrama é a classe agente que é representada por um retângulo e uma *tag* A. Os agentes são interligados através de uma linha.

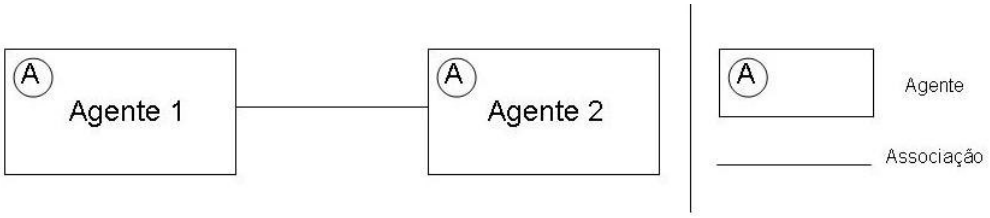

Figura 10. Elementos do diagrama de agente do ANote.

O diagrama de ambiente é o diagrama que especifica componentes do sistema que não são agentes e para tanto utiliza a UML para modelar tal visão. Serve para especificar o ambiente de componentes onde agentes interagem. Se o sistema interage com outros programas ou banco de dados, este diagrama pode representá-los através do diagrama de componentes da UML.

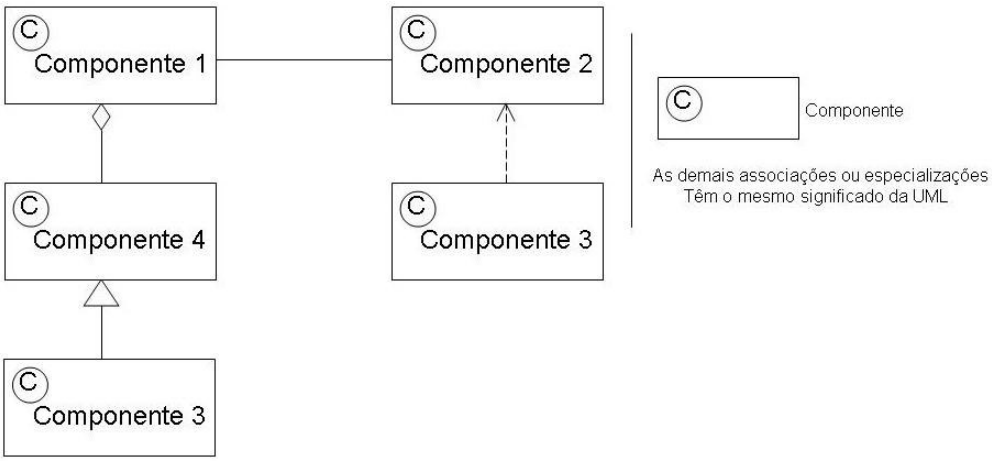

Figura 11. Elementos do diagrama de ambiente do ANote.

O diagrama de organização permite que o desenvolvedor pense no sistema multi-agente com um conjunto de componentes ou como a implementação física da unidade. Este diagrama é representado como cubos, onde os cubos representam organizações de agentes (ou seja, mais de um agente).

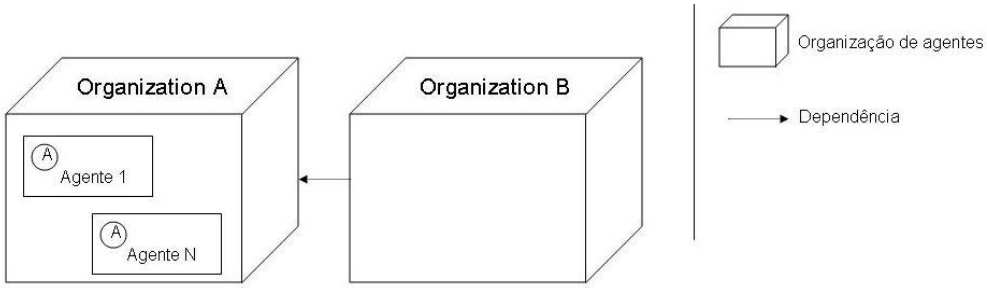

Figura 12. Elementos do diagrama de organização do ANote.

Diagrama de cenário é o diagrama que captura o comportamento do agente em um contexto específico. Diferente das demais, esta visão é constituída apenas de uma descrição. Este diagrama ajuda o desenvolvedor a elaborar os caminhos através dos objetivos dos agentes.

Tabela 1. Elementos do diagrama de cenário do ANote.

Objetivo do agente	
Lead Agent	Nome do agente principal
Precondition	Precondições para que o objetivo possa ser cumprido
Main Action Plan	Seqüência de ações (principais) que podem ser executadas
Interactions	Nome do agente que interage com o agente principal
Variant Plan	Seqüência de ações (secundárias) que podem ser executadas

O diagrama de planejamento especifica o fluxo em que as ações de um agente devem ser executadas. Este diagrama é similar ao diagrama de atividades da UML.

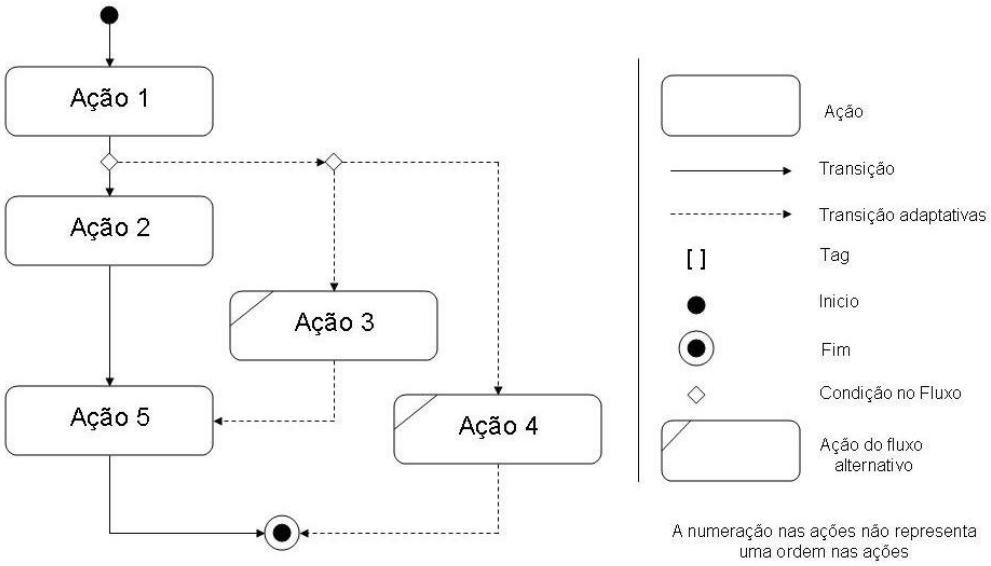

Figura 13. Elementos do diagrama planejamento do ANote.

O diagrama de comunicação representa os agentes que enviam e recebem mensagens enquanto estão executando o plano das ações (Choren & Lucena, 2003b).

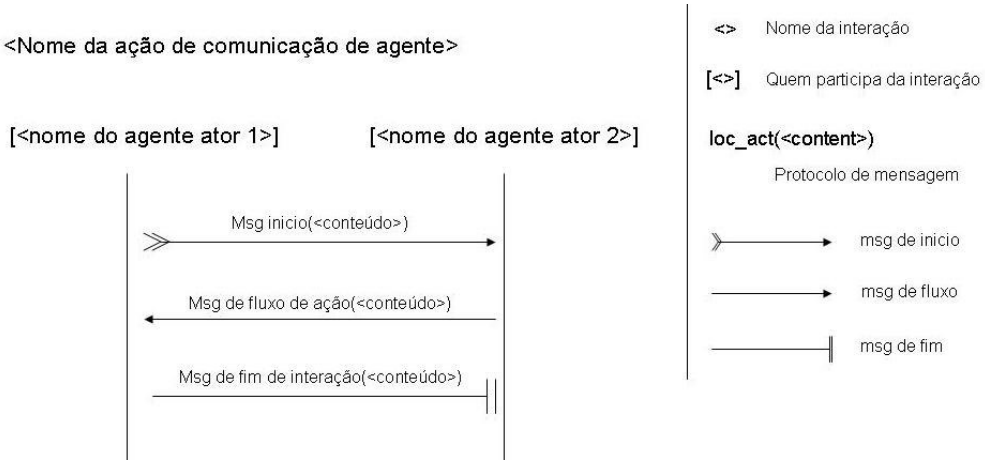

Figura 14. Elementos do diagrama de comunicação do ANote.

3.2. Linguagens de Modelagem de SMA e Representação de Variabilidade

Diversas abordagens foram desenvolvidas para facilitar o desenvolvimento de aplicações distribuídas e complexas. Em particular, evidencias sugerem que o paradigma de agentes oferece a abstração adequada para se desenvolver este tipo

de sistema, o que resultou no desenvolvimento de diversas linguagens de modelagens de SMA.

Dentre as linguagens de modelagens três foram abordadas neste capítulo, ANote, AUML e Tropos. Estas linguagens apresentam diferentes formas de representar SMA e ambas trazem seus pontos possitivos e negativos.

Diferente das linguagens de modelagem OO as linguagens de modelagem de SMA trazem sempre um ponto forte dentro de sua modelagem.

O Tropos, por exemplo, é uma linguagem bastante utilizada em engenharia de requisitos, já que disponibiliza uma maneira de representar análises de dependência de sistemas e de requisitos funcionais e não funcionais.

O Tropos faz uso de padrões sociais para determinar a modelagem do sistema multi-agente.

A AUML é uma linguagem que representa muito bem a troca de mensagem entre os agentes de um sistema, inclusive respeitando o protocolo de troca de mensagens FIPA-ACL. O diagrama de interação da AUML é o responsável por modelar a troca de mensagens entre os agentes em um SMA. Os demais diagramas da AUML mostram uma modelagem mais alto nível determinando características do agente e do sistema.

O ANote é uma linguagem que visa representar um sistema multi-agente de uma forma geral e abrangente de modo que todas as características de um agente possam ser representadas. Esta linguagem de modelagem de agentes apresenta sete diagramas e através destes a modelagem é capaz de expressar tudo o que se passa num sistema multi-agente. O ANote não tem uma modelagem atrelada a linguagem de programação ou um *framework* especifico, o que possibilita ter uma representação bastante genérica.

O ponto negativo das linguagens estudadas é que nenhuma destas disponibiliza uma maneira de representar as características de variabilidade existente em agentes de software, isso porque seus meta-modelos apresentam características inerentes que não disponibilizam uma forma de representar instâncias de novas aplicações.

Desta forma, não é possível representar instância de uma nova aplicação a partir de um agente e assim os SMA acabam se tornam sistemas mais rígidos do que deviam ser.

Na AUML a representação da variabilidade na troca de mensagem entre os agentes fica a cargo do diagrama de protocolo de interação que representam a interação entre os agentes.

Um ponto negativo da AUML é o fato de que não é possível através dela representar as variabilidades que podem ocorrer no desenvolvimento de SMA; as aplicações que devem ser instanciadas não são representadas em seus diagramas.

Na metodologia Tropos observa-se uma representação sem se preocupar com a possibilidade de uma extensão do sistema através de artefatos como um *framework*, onde as variabilidades são bem definidas em função da aplicação que se quer instanciar.

A dificuldade em utilizar Tropos para representar instâncias distintas (como em um framework) se dá em primeiro lugar pelo fato da própria linguagem não dar um suporte adequado para tal abordagem. Como visto anteriormente o Tropos se utiliza do meta-modelo i*. Tal meta-modelo oferece os conceitos primitivos de atores, objetivo e dependência de atores e com isso o i* disponibiliza um framework para modelar processos que envolvem múltiplos participantes (humanos ou agentes de software).

Pode-se observar que o modelo i* dá suporte ao desenvolvimento, mas não prevê o uso de um agente como framework e por isto não disponibiliza uma maneira de modelar o agente como tal. Os tipos primitivos do meta-modelo i* possibilitam a modelagem de atores, dependência e objetivos, mas nenhuma delas permite a modelagem de novas instâncias de aplicações a partir de um agente.

A metodologia Tropos não aborda pontos como a representação da variabilidade segundo a aplicação que se quer instanciar em um projeto de agentes de software. O Tropos utiliza o diagrama de atividade da UML e o diagrama de interação da AUML para representar de forma mais apropriada a interação entre os agentes.

Assim como Tropos e AUML o ANote não representa pontos como a variabilidade de elementos no design dos agentes de software.

O fato de não ser possível projetar a variabilidade existente em agentes de software ocorre, pois o meta-modelo não disponibiliza uma tupla de conceitos, relações e condições de maneira que seja possível representar SMA que contenham agentes com características de variabilidade. Isto faz com que o

sistema se prive de uma flexibilização natural que existe em determinados agentes.

O meta-modelo do ANote determina que agentes têm relacionamentos apenas com objetivo, mensagem, ações, organização, cenário e fonte de dados (banco de dados), e com estes relacionamentos não é possível projetar instâncias de novas aplicações a partir de um agente de software.

3.3. Pontos de Flexibilização de SMA

A primeira etapa no desenvolvimento do método proposto consistiu em estudar cada um dos meta-modelos das linguagens de modelagem de agentes, seus elementos e relacionamentos; dentre as linguagens aqui apresentadas apenas o ANote e o Tropos têm meta-modelo. Com base neste estudo foram determinados os pontos que podem ser variáveis num agente. Após o estudo do meta-modelo e a definição dos pontos que podem ser variáveis, diagramas e técnicas do método proposto foram desenvolvidas e serão utilizados para definir uma linha de produto em um sistema multi-agente.

As representações das variabilidades aqui propostas se assemelham com as variabilidades representadas por *hot-spots* em um *framework*, ou seja, tanto o método proposto como *frameworks* determinam instâncias distintas para aplicações, as quais variam em função do escopo do sistema. A figura a seguir (Fayad & Johnson, 2000) representa a variabilidade em um *framework*.

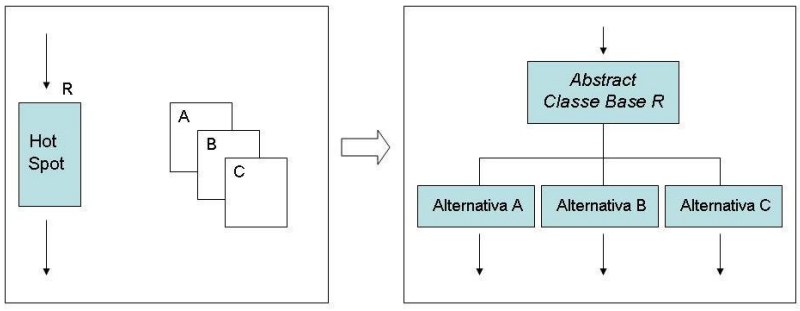

Figura 15. Representação de um *framework*.

Um agente de software tradicionalmente tem uma arquitetura dividida em: objetivos, planos e ações.

Dentro dessa divisão, os planos e as ações surgem como os pontos que apresentam a maior variabilidade no desenvolvimento de SMA, já que estes

pontos muitas das vezes apresentam características que possibilitam novas instâncias de aplicações para um sistema.

Em muitos casos os SMA apresentam uma necessidade de aplicações distintas, tais aplicações podem surgir em função da variabilidade do plano do agente ou da variabilidade da ação de um agente. Porém as linguagens e técnicas atuais não apresentam uma forma de representar os pontos de flexibilização dos agentes e assim os sistemas acabam sendo modelados de forma inflexível, o que torna o código e modelagem muito distantes um do outro.

Aplicações distintas em função da variabilidade dos planos dos agentes ocorrem quando parte das ações de um dado plano variam em função do escopo do sistema e isto geralmente acarreta dois ou mais planos distintos que vão determinar instâncias de aplicações distintas.

O plano de um agente é composto de ações (ações concretas) e podem ser vistos como um ambiente de execução das ações de um agente. A flexibilização de um plano talvez seja mais bem entendida se um plano for tido como o local de execução das ações, assim fica claro que a necessidade de mudança de local se dá em função das ações que serão executadas. A flexibilização de um plano traz a idéia de plano abstrato que tem características distintas do plano concreto, uma definição mais formal de um plano abstrato é a seguinte: Um plano que é composto de ações, onde algumas destas ações podem ser definidas antes mesmo de ter sido definido o foco principal da aplicação (estas podem ser chamadas de ações concretas); as demais ações só poderão ser definidas após a determinação do foco principal da aplicação (ou seja, a aplicação final) tais ações só são implementadas no plano que será instanciado (estas podem ser chamadas de ações abstratas), o plano abstrato é composto de ações concretas e abstratas; o plano que será instanciado contém ações concretas que complementam e implementam o plano abstrato, finalizando o plano do agente e em função desta variabilidade é que surge o plano abstrato.

O mesmo ocorre com as ações dos agentes, ou seja, um sistema multi-agente pode comportar aplicações distintas, em função de suas ações, as quais podem ser implementandas de diferentes maneiras de executar a referida ação. Isto é o que traz também a variabilidade para as ações. Por isso as ações e os planos dos agentes são os pontos de flexibilização.

Na solução o método proposto traz uma forma de projetar variabilidade a qual será arquiteturada em linhas de produto. As linhas de produto representarão todas as aplicações que podem ser instanciadas dentro do escopo do sistema através de uma família de produtos. Como visto anteriormente tais aplicações podem estar presentes em planos e ações e isto representa a variabilidade do agente, porém numa abordagem de modelagem. A definição de linhas de produto traz muitos benefícios como agilidade e redução de custo o que torna as aplicações multi-agentes mais atraentes para a indústria (Peña, J. et al., 2006).

Com isto pode-se ver que a planificação dinâmica e o método proposto têm características em comum, no que diz respeito à variabilidade de um agente de software. Porém uma abordagem trata de questões de implementação e outra questões de modelagem.

A flexibilização dos planos possibilita a introdução de variabilidades distintas no desenvolvimento de agentes de software o que traz mais clareza e coerência para os desenvolvedores, já que dessa forma será possível determinar qual aplicação deverá ser implementada. Toda a abstração deve ser feita em função das aplicações que se pretende desenvolver e assim será possível agregar uma grande flexibilização para a abordagem de agentes.

A seguir será apresentada toda a estrutura para a flexibilização de planos.

Apesar de os requisitos citados terem como foco o desenvolvimento de *framework* orientado a objeto, estes podem ser facilmente transportados para a abordagem de agente proposta aqui nesta dissertação.