MAS-ML 2.0: Suporte à modelagem de sistemas multiagentes com diferentes arquiteturas de agentes

Abstract Os sistemas multiagentes (MAS) envolvem uma ampla variedade de agentes que interagem entre si para atingir seus objetivos. Normalmente, cada agente possui uma arquitetura interna específica, definindo sua estrutura principal que dá suporte à interação entre as entidades do MAS. Muitas linguagens de modelagem foram propostas nos últimos anos para representar as arquiteturas internas desses agentes, por exemplo, os perfis UML. Em particular, destacamos o MAS-ML, uma linguagem de modelagem MAS que executa uma extensão conservadora da UML enquanto incorpora conceitos relacionados ao agente para representar agentes proativos. No entanto, essas linguagens falham no suporte à modelagem de arquiteturas heterogêneas que podem ser usadas para desenvolver os agentes de um MAS. Pior ainda, pouco foi feito para fornecer ferramentas para ajudar no projeto sistemático de agentes. Este artigo, portanto, tem como objetivo estender o metamodelo MAS-ML e desenvolver sua ferramenta para apoiar a modelagem não apenas de agentes proativos, mas também de várias outras arquiteturas descritas na literatura.

1. Introdução

Atualmente, a tecnologia de agentes tem sido amplamente aplicada para resolver um vasto conjunto de problemas. Russell e Norvig (2003) definem um agente como uma entidade que pode perceber seu ambiente através de sensores e atuar no ambiente através de atuadores. Ao contrário dos objetos, os agentes são entidades mais complexas com propriedades comportamentais, como (i) autonomia (ou seja, são capazes de executar sem interagir com seres humanos) e (ii) interação (ou seja, são capazes de interagir enviando e recebendo mensagens e não por invocação explícita de tarefas (Wagner, 2003) O sistema multiagente (MAS) é a subárea da inteligência artificial que investiga o comportamento de um conjunto de agentes autônomos, com o objetivo de resolver um problema que está além da capacidade de um único agente (Jennings 1996).

Um agente simples pode agir com base em comportamento reativo ou proativo e pode ser classificado de acordo com sua arquitetura interna que determina propriedades, atributos e componentes mentais distintos da agência (Russell e Norvig, 2003). Russell e Norvig (2003) definem quatro tipos de agentes de acordo com suas arquiteturas internas: Agente Reflex Simples, Agente Reflex Baseado em Modelo, Agente Objetivo e Agente Utilitário. A arquitetura interna de

um agente é selecionada de acordo com as características do ambiente e com o subproblema que o agente resolverá. Em Weiss (1999), os ambientes são classificados como (i) totalmente observáveis ou parcialmente observáveis, dependendo do grau de consciência; (ii) determinístico (previsível) ou estocástico (imprevisível), dependendo da previsibilidade da evolução do ambiente; (iii) episódico (uma ação não depende de ações anteriores) ou sequencial (a execução de uma ação depende de ações anteriores); (iv) estático (o ambiente não muda enquanto o agente está raciocinando) ou dinâmico (ele muda enquanto o agente está raciocinando); e (v) discreto (possui uma quantidade finita de estados) ou contínuo (possui uma quantidade contínua de estados). O MAS pode abranger vários tipos de agentes com diferentes arquiteturas internas (Weiss, 1999).

O paradigma de desenvolvimento orientado a agentes requer técnicas adequadas para explorar seus benefícios e recursos, a fim de apoiar a construção e manutenção desse tipo de software (Zambonelli et al., 2001). Como é o caso de qualquer novo paradigma de engenharia de software, a implantação bem-sucedida e generalizada de MASs requer linguagens de modelagem que exploram o uso de abstrações relacionadas a agentes e promovem a rastreabilidade dos modelos de design ao código. Para reduzir o risco ao adotar uma nova tecnologia, é conveniente apresentá-la como uma extensão incremental de métodos conhecidos e confiáveis e fornecer ferramentas explícitas de engenharia que suportam métodos aceitos pelo setor de implantação de tecnologia (Castro et al., 2006). Uma linguagem de modelagem para um sistema multi-agente, de preferência, deve ser uma extensão incremental de uma linguagem de modelagem conhecida e confiável. Como agentes e objetos coexistem nos MASs, a linguagem de modelagem UML (Castro et al., 2006) pode ser usada como base para o desenvolvimento de linguagens de modelagem MAS. A linguagem de modelagem UML é um padrão de fato para modelagem orientada a objetos. A UML é usada na indústria e na academia para modelar sistemas orientados a objetos. No entanto, em sua forma original, a UML fornece suporte insuficiente para a modelagem de MASs.

Existem várias linguagens de modelagem MAS (Argente et al., 2009) que estenderam a UML para modelar características orientadas a agentes. No entanto, ainda é necessário uma linguagem de modelagem capaz de modelar diferentes arquiteturas de agentes internos. O principal objetivo deste artigo é apresentar uma linguagem de modelagem baseada em UML capaz de modelar as características internas do agente apresentadas em Russell e Norvig (2003). Para atingir nosso objetivo, estendemos o MAS-ML (linguagem de modelagem de sistemas multiagentes) (Silva et al., 2008a), uma linguagem de modelagem que executa uma extensão conservadora da UML com base nos conceitos orientados a agentes definidos no quadro conceitual TAO (Taming Agents and Objects) (Silva e Lucena, 2004). O MAS-ML e o TAO foram projetados originalmente para suportar a modelagem de apenas agentes proativos que são entidades baseadas em objetivos, guiadas por planos predefinidos. No entanto, nem todos os MASs requerem agentes

proativos, como no caso de simulações para uma colônia de formigas (Dorigo e Stützle, 2004). Portanto, é importante estender o MAS-ML e o TAO para modelar não apenas agentes proativos, mas também reativos. Juntamente com a extensão do MAS-ML e TAO, também sentimos a necessidade de estender o ambiente de modelagem do MAS-ML.

Optamos pelo MAS-ML, pois ele dá suporte à modelagem (i) das principais entidades do MAS: agentes, organização e ambientes; (ii) as propriedades estáticas e dinâmicas de um MAS; (iii) papéis de agentes, o que é importante na definição de sociedades de agentes; e (iv) agentes proativos. Linguagens como AUML (Agent Unified Modeling Language) (Odell et al., 2000; Guedes e Vicari, 2009) e AORML (Agent-Oriented Relationship Modeling Language) (Wagner, 2003; Wagner e Taveter, 2005) não definem (i) o ambiente como uma abstração; portanto, não é possível que os agentes do modelo possam passar de um ambiente para outro; e (ii) as propriedades das organizações não são descritas; portanto, não é possível modelar agentes de uma organização para outra. Um dos fatores que influencia a escolha da arquitetura interna é o tipo de ambiente em que o agente está inserido; portanto, é desejável a representação ambiental em diagramas de idiomas, pois, uma vez modelado, é possível explicar seu tipo por meio de notas textuais. , por exemplo. Para a organização, embora não esteja relacionada à escolha da arquitetura interna, sua representação é importante do ponto de vista da coordenação entre agentes. A linguagem ANote Modeling (Choren e Lucena, 2005), por exemplo, não suporta objetos convencionais usados para modelar entidades não autônomas. Além disso, na AML (Agent Modeling Language) (Cervenka, 2012), os relacionamentos de comunicação são modelados como uma especialização de elementos UML, o que não é adequado para a comunicação de agentes de modelagem.

Para validar as extensões propostas para o MAS-ML 2.0, TAO e a ferramenta de modelagem, foi realizado um estudo de caso para usar nossas extensões para modelar o sistema TAC-SCM (Trading Agent Competition - Supply Chain Management) (Sadeh et al., 2003). Esse sistema simula leilões simultâneos com consumidores e fornecedores de computadores. Cinco agentes implementados com diferentes arquiteturas internas foram modelados usando o MAS-ML 2.0 e outras três linguagens de modelagem: MAS-ML original, AUML e AML. Esses diferentes modelos nos permitiram demonstrar a expressividade das extensões propostas em relação a outras abordagens. O MAS-ML 2.0 pode modelar todas as quatro arquiteturas de agentes internos, enquanto o MAS-ML, AUML e ML originais podem modelar apenas as duas arquiteturas mais simples.

O artigo está estruturado da seguinte forma. As principais arquiteturas internas para agentes, TAO e MAS-ML, são descritas na Seção 2. A Seção 3 apresenta trabalhos relacionados que envolvem estruturas conceituais e linguagens de modelagem. A extensão do TAO é descrita na Seção 4, a extensão MAS-ML é detalhada na Seção 5 e a ferramenta MAS-ML é detalhada na Seção 6. Na Seção 7,

a modelagem do TAC-SCM (Competição de agentes comerciais - Gerenciamento da cadeia de suprimentos) (Sadeh et al., 2003) é apresentada para ilustrar a contribuição deste trabalho. Finalmente, conclusões e trabalhos futuros são discutidos na Seção 8.

2. Prática

Esta seção descreve os principais conceitos necessários para entender este trabalho, incluindo os conceitos relacionados a arquiteturas de agentes, estrutura conceitual TAO e linguagem de modelagem MAS-ML.

2.1. Arquitetura dos Agentes

As arquiteturas internas do agente podem ser categorizadas com base em fundamentos proativos e reativos. Nesse contexto, quatro tipos de arquiteturas de agentes internos foram definidos por Russell e Norvig (2003). Essas arquiteturas são detalhadas nas próximas seções.

2.1.1. Agente Reflexivo Simples

Um agente simples de reflexo (ou reativo) (Russell e Norvig, 2003) é considerado a arquitetura interna mais simples. Regras de ação-condição são usadas para selecionar as ações com base na percepção atual. Essas regras seguem a forma "se condição, então ação" e determinam a ação a ser executada se a percepção ocorrer. Essa arquitetura pressupõe que a qualquer momento o agente receba informações do ambiente por meio de sensores. Essas percepções consistem na representação de aspectos do estado que são usados pelo agente para tomar decisões. Um subsistema é responsável pela tomada de decisão, ou seja, responsável pelo processamento da sequência de percepção e pela seleção de uma sequência de ações no conjunto de ações possíveis para o agente. O agente executa a ação selecionada em um ambiente por meio de atuadores.

2.1.2. Agente Reflexivo Baseado em Modelo

A estrutura desse tipo de agente é semelhante ao Simple Reflex Agent apresentado anteriormente, pois lida com as informações usando regras de ação e condição. Para lidar com ambientes parcialmente observáveis e alcançar um desempenho mais racional, o agente também pode armazenar seu estado atual em um modelo interno.

Segundo Weiss (1999), os agentes reflexos baseados em modelo selecionam ações usando as informações em seus estados internos. Uma função chamada próxima função é introduzida para mapear as percepções e o estado interno atual em um novo estado interno usado para selecionar a próxima ação. Tal estado descreve aspectos do mundo (chamado modelo) que não podem ser vistos no

momento atual, mas foram percebidos mais cedo ou surgiram por inferências (Russell e Norvig, 2003).

2.1.3. Agentes Baseados em Objetivos

Às vezes, o conhecimento sobre o estado atual do ambiente não é suficiente para determinar a próxima ação e são necessárias informações adicionais sobre situações desejáveis. Agentes baseados em metas são agentes baseados em modelo que definem uma meta específica e selecionam as ações que levam a essa meta. Isso permite que o agente escolha um estado de meta entre várias possibilidades.

A atividade de planejamento é dedicada a encontrar a sequência de ações que são capazes de atingir os objetivos do agente (Russell e Norvig, 2003).

A sequência de ações previamente estabelecidas que leva o agente a atingir uma meta é denominada plano (Silva e Lucena, 2004; Silva et al., 2008a). Assim, o agente baseado em metas com planejamento envolve o próximo componente de função e também inclui os seguintes elementos:

- Função Formular Objetivo, que recebe o estado e retorna o objetivo formulado;
- Formule a função do problema, que recebe o estado e a meta e retorna o problema;
- Planejamento, que recebe o problema e usa abordagens de pesquisa e / ou lógica para encontrar uma sequência de ações para atingir uma meta; e
- Ação, representada com suas pré-condições e pós-condições.

2.1.4. Agentes Baseados em Utilidade

Considerando a existência de múltiplos estados de objetivo, é possível definir uma medida de quão desejável é um estado específico. Nesse caso, com o objetivo de otimizar o desempenho do agente, a função utilitário é responsável por mapear um possível estado (ou grupo de estados) para essa medida, de acordo com os objetivos atuais (Russell e Norvig, 2003).

Assim, a função utilidade é incorporada à arquitetura.

Além disso, o Agente Baseado em Utilidade preserva os mesmos elementos que os de um Agente Baseado em Objetivo: próxima função, formule a função objetivo, formule a função problema, o planejamento e a ação.

2.1.5. Escolha de Arquiteturas Internas

Escolhemos as arquiteturas internas dos agentes relacionados por Russell e Norvig (2003) porque (i) eles são amplamente utilizados no desenvolvimento do MAS e (ii) não é possível modelá-los com a sintaxe disponível nas linguagens de modelagem do MAS. Russell e Norvig (2003) e Weiss (1999) relacionaram a escolha da arquitetura interna do agente às características do ambiente.

O comportamento proativo é especialmente adequado para ambientes com tarefas bem definidas, ou seja, ambientes determinísticos, observáveis e estáticos. Projetar agentes proativos em ambientes estocásticos e parcialmente observáveis é uma tarefa muito complexa. Além disso, em ambientes dinâmicos, onde as pré-condições podem mudar enquanto o agente está em execução, o comportamento puramente proativo pode produzir efeitos indesejados. Nesses casos, o agente deve poder parar o processo de deliberação ou a execução de um plano que foi deliberado e responder adequadamente às novas condições do ambiente. Assim, nesses ambientes em que o tempo de resposta é crucial, é importante implementar um agente reativo combinado com características proativas.

Um agente simples de reflexo pode ser mais adequado a ambientes completamente observáveis, pois esse tipo de agente deve tomar decisões baseadas apenas nas percepções atuais; portanto, não mantém as percepções do passado histórico. Nesse caso, costumamos dizer que o agente esquece seu próprio passado.

Os agentes reflexivos baseados em modelo podem se sair melhor em uma ampla gama de ambientes do que os agentes reativos. Eles são capazes de manter representações internas do ambiente que não estão sendo observadas atualmente. Portanto, em ambientes parcialmente observáveis, os agentes reflexos geralmente apresentam desempenho superior aos agentes reativos simples.

Os agentes reflexos são mais adequados para subproblemas que requerem respostas rápidas. Ao agir em conjunto, os agentes reflexos podem obter bons resultados, como no caso das colônias de formigas (Dorigo e Stützle, 2004). Por outro lado, os agentes com base em metas que selecionam um plano com base na meta que o agente deseja executar são mais adequados para ambientes que não são modificados durante a execução do plano. Se o ambiente mudar, a sequência de ações executadas poderá falhar devido ao ambiente inesperado. Portanto, um agente baseado em objetivos é adequado para ambientes parcialmente observáveis, estáticos e determinísticos.

Um agente baseado em utilidade é mais eficaz em duas situações: (i) quando o resultado da execução das ações é incerto e (ii) quando há mais de um objetivo a ser alcançado. No primeiro caso, um agente orientado a utilidade pode selecionar a ação com maior probabilidade de produzir o estado desejado. No segundo caso, o agente pode selecionar uma meta com o utilitário mais alto. Assim, um agente baseado em utilidade é mais adequado para ambientes não determinísticos e parcialmente observáveis.

2.2. TAO: Domando agentes e objetos

A estrutura conceitual 1 TAO (Taming Agents and Objects) fornece uma ontologia que abrange os fundamentos da Engenharia de Software com base em agentes e objetos e possibilita o desenvolvimento do MAS em larga escala (Silva et

al., 2003). Essa estrutura provoca uma ontologia que conecta abstrações consolidadas, como objetos e classes, e abstrações "emergentes", como agentes, funções e organizações, que são os fundamentos da engenharia de software baseada em objetos e agentes. O TAO apresenta a definição de cada abstração como um conceito de sua ontologia e estabelece as relações entre elas. A figura 1 mostra as abstrações e relações propostas no TAO. As abstrações do TAO são definidas da seguinte maneira:

- Objeto: É um elemento passivo ou reativo que possui estado e comportamento e pode estar relacionado a outros elementos.
- Agente: É um elemento autônomo, adaptável e interativo que possui um estado mental. Seu estado mental possui os seguintes componentes: (i) crenças (tudo o que o agente sabe), (ii) objetivos (estados futuros que o agente deseja alcançar), (iii) planos (sequências de ações que atingem um objetivo) e (iv) as próprias ações.
- Organização: é um elemento que agrupa agentes e sub organizações, que desempenham papéis e têm objetivos comuns. Uma organização oculta características intra características, propriedades e comportamentos representados por agentes dentro dela. Pode restringir o comportamento de seus agentes e suborganizações através do conceito de axioma, que caracteriza as restrições globais da organização a que agentes e sub organizações devem obedecer.
- Função do objeto: é um elemento que guia e restringe o comportamento de um objeto na organização. Uma função de objeto pode adicionar informações, comportamento e relacionamentos que a instância do objeto executa.
- Função do agente: é um elemento que guia e restringe o comportamento do agente que desempenha o papel na organização. Uma função de agente define (i) tarefas como ações que devem ser executadas pelo agente que desempenha a função, (ii) direitos como ações que podem ser executadas pelo agente que desempenha a função e (iii) protocolo que define uma interação entre as funções de agente .
- Ambiente: É um elemento que é o habitat de agentes, objetos e organizações. O ambiente tem estado e comportamento.

Além disso, Silva et al. (2003) definem os seguintes relacionamentos no TAO: Habitação, Propriedade, Brincadeira, Especialização / Herança, Controle, Dependência, Associação e Agregação / Composição. Esses relacionamentos não serão descritos aqui, pois a extensão proposta neste trabalho não envolve relacionamentos, mas é descrita em Silva et al. (2003).

Os conceitos são apresentados por Silva et al. (2003) em uma abordagem semi-formal que utiliza modelos para formalizar os Objetos, Agentes, Organização,

Função do Objeto, Função do Agente, Ambiente e seus relacionamentos. Um modelo de agente e um modelo de função de agente são apresentados a seguir:

O modelo de agente apresentado acima é composto por seu nome, objetivos, crenças, ações, planos, eventos gerados e percebidos, além de papéis associados e relacionamentos. O papel de agente apresentado abaixo é composto por objetivos, crenças, deveres, direitos, protocolos, compromissos e relacionamentos.

2.3. MAS-ML

O MAS-ML é uma linguagem de modelagem que implementa os conceitos de TAO e foi originalmente projetada para suportar a modelagem de agentes proativos baseados em objetivos, guiados por planos pré-estabelecidos.

O MAS-ML modela todos os aspectos estruturais e dinâmicos definidos no metamodelo TAO estendendo o metamodelo UML. Os diagramas estruturais definidos pelo MAS-ML são Diagrama de Funções, Diagrama de Classes e Diagrama de Organização (Silva et al., 2004). Os diagramas dinâmicos são Diagrama de Sequência e Diagrama de Atividades (Silva et al., 2008b). Usando todos esses diagramas, é possível modelar todos os aspectos estruturais e dinâmicos das entidades definidas no TAO.

2.3.1. Diagramas Estáticos

Os diagramas estáticos propostos pelo MAS-ML são três extensões do diagrama de classes UML. O diagrama de classes MAS-ML pode representar os relacionamentos de classe (entidade do diagrama de classes UML original) e de diagrama de classes UML (associação, agregação e especialização). Além disso, foram incluídas as entidades e os relacionamentos específicos: Agente, Ambiente e Entidade da Organização e habitar o relacionamento; O diagrama de organização do MAS-ML pode representar organizações, suborganizações, classes, agentes, funções de agente, funções de objeto e ambiente e propriedade e propriedade, reproduzir e habitar relacionamentos. Um diagrama de função MAS-ML pode representar a função de agente e a função de objeto e seus relacionamentos Controle, Dependência, Associação, Agregação e Especialização.

As entidades adicionadas aos diagramas estáticos são mostradas na Fig. 2. A Fig. 2 (A) mostra o elemento agente usado nos diagramas estáticos MAS-ML. É um retângulo com bordas arredondadas com três compartimentos: o superior tem o nome do agente; o meio tem objetivos e crenças; e o inferior tem planos e ações. A figura 2 (B) mostra a representação do papel do agente. É um retângulo com a borda curvada para baixo e com três compartimentos: a parte superior tem o nome da instância; o centro tem objetivos e crenças; e o inferior tem deveres, direitos e detalhes do protocolo de comunicação. A figura 2 (C) mostra a representação da organização. É uma união de um retângulo e uma elipse com três compartimentos: o superior tem o nome da instância; o meio tem objetivos, crenças e axiomas; e o

inferior tem planos e ações. A figura 2 (D) mostra a representação da função do objeto. É um retângulo com um corte no canto superior esquerdo e com três compartimentos semelhantes à representação da classe UML.

A figura 2 (E) mostra a representação do ambiente. O compartimento superior possui o nome da instância, o compartimento intermediário possui propriedades e o compartimento inferior possui métodos (Operações). Um quarto compartimento foi adicionado para representar os elementos que habitam o ambiente.

O MAS-ML representa quatro relacionamentos em diagramas estáticos: Habitação, Propriedade, Brincadeira, Controle e Dependência. A relação de habitação pode ser observada na Fig. 2 (E): o compartimento mais abaixo é a relação de habitação.

A figura 3 (A) mostra a relação de propriedade. É usado para vincular uma organização e as funções que podem ser desempenhadas na organização (função de agente ou função de objeto) e é representado por uma linha dupla. A Fig. 3 (C) mostra a relação Play. É um relacionamento ternário que liga um agente, um papel e uma organização. Isso significa que o agente pode desempenhar esse papel na organização. Esse relacionamento é representado por uma linha que conecta um agente a um relacionamento de propriedade. A figura 3 (B) mostra a relação de controle. O relacionamento define que uma função de agente controla outra função de agente. Ele modela um relacionamento social entre funções de agente, como um mestre-escravo. Uma única linha com um círculo representa esse relacionamento; o círculo indica o controlador.

2.3.2. Diagramas Dinâmicos

Os diagramas dinâmicos definidos no MAS-ML são versões estendidas do Diagrama de Sequência e Diagrama de Atividades da UML (Silva et al., 2008b). As entidades modeladas em um diagrama de sequência são mostradas na figura 4, onde objeto (figura 4 (A)), agente (figura 4 (B)), organização (figura 4 (C)) e ambiente (figura 4 (D)) estão representados. Um diagrama de sequência identifica as entidades (por exemplo, o agente ou a organização), os papéis que eles estão desempenhando (se for o caso), as organizações nas quais os papéis estão sendo desempenhados - é importante, pois as entidades podem desempenhar papéis diferentes em diferentes organizações (Odell, Parunak e Bauer, 2003) - e também os ambientes em que estão imersos (D'Inverno e Luck, 2001). Representar o ambiente é particularmente interessante, pois pode mostrar a distribuição do sistema.

Os relacionamentos criados pelo MAS-ML com o diagrama de sequência da UML são mostrados na Fig. 5: os dois últimos estão relacionados a agentes e os outros estão relacionados a agentes que desempenham papéis. Os estereótipos relacionados à ativação de função, desativação de função, comprometimento de função e cancelamento de função podem ser usados para conectar agentes a funções de agente.

O diagrama de atividades do MAS-ML é capaz de representar o comportamento do agente por meio de elementos do diagrama de atividades UML. Além disso, o plano de estereótipos, crença, mensagem, organização e ambiente podem ser usados para representar seus respectivos elementos. As raias de natação representam agentes, ambientes e funções. A Fig. 6 mostra um exemplo de um diagrama de atividades do MAS-ML.

3. Trabalhos Relacionados

Esta seção envolve trabalhos relacionados a (i) estruturas conceituais e (ii) linguagens de modelagem, tanto no contexto do MAS quanto considerando sistemas que envolvem agentes com arquitetura interna diferente, apresentados na Seção 2.1.

3.1. Frameworks Conceituais

Algumas estruturas conceituais foram propostas para o MAS. No entanto, eles fornecem suporte limitado a arquiteturas internas. Nosso objetivo é analisar três estruturas conceituais, considerando o suporte fornecido à modelagem das entidades típicas do MAS, juntamente com suas propriedades e relacionamentos. Além disso, será analisado o suporte dado à modelagem das arquiteturas internas do agente.

A estrutura de D'Inverno e Luck (2001) define uma hierarquia composta por quatro camadas com entidades, objetos, agentes e agentes autônomos. No entanto, apresenta as seguintes limitações: (i) a entidade ambiental possui apenas características estruturais sem transações, (ii) nenhum aspecto dinâmico associado às entidades propostas é definido e (iii) não fornece elementos para definir os diferentes agentes internos arquiteturas.

Yu e Schmid (2001) propõem uma estrutura conceitual para a definição de MAS orientado a agentes baseado em função. Os agentes são mostrados como uma entidade que desempenha papéis em qualquer organização. Como pontos fracos, destacamos os seguintes aspectos: (i) embora os agentes sejam definidos como uma entidade desempenhando papéis, essa estrutura conceitual não define as propriedades do agente e os relacionamentos entre agentes e papéis; (ii) embora os autores declarem que os papéis são desempenhados nas organizações, a proposta não define as propriedades da organização e os relacionamentos entre elas e os papéis; (iii) também não define a entidade ambiental que contém agentes e organizações; e (iv) restringe apenas o comportamento do agente no contexto de uma função.

Centeno et al. (2008) definem uma estrutura conceitual para mecanismos organizacionais em MASs. Eles usam agentes, ambiente e organização para definir os conceitos organizacionais relacionados à sua organização e operação. Eles argumentam que os mecanismos organizacionais podem ser classificados em dois tipos básicos: (i) aqueles que fornecem informações adicionais sobre o meio

ambiente (seu estado, possíveis ações, seus resultados esperados etc.) e (ii) aqueles que manipulam adequadamente o ambiente. Argumentamos que uma característica chave para um MAS ser “baseado em organização” é fazer uso de pelo menos um desses mecanismos.

Shirazi e Barfouroush (2008) definem uma estrutura conceitual para modelar e desenvolver sistemas de negociação automatizados. Essa estrutura representa e especifica todos os conceitos e entidades necessários para o desenvolvimento de um sistema de negociação, bem como os relacionamentos entre esses conceitos. Essa estrutura também pode ser usada para modelar cenários de negociação humana para analisar esses tipos de negociação e simulá-los com sistemas com vários agentes.

O trabalho de Lavinal et al. (2006) apresenta uma estrutura genérica baseada em agentes como um primeiro passo para a concepção de sistemas autogerenciados. Essa estrutura conceitual consiste em grupos de agentes de gerenciamento acoplados a recursos gerenciados e dotados de habilidades específicas de gerenciamento.

Beydoun et al. (2009) introduzem um metamodelo relativamente genérico, orientado a agentes, cuja adequação ao suporte ao desenvolvimento da linguagem de modelagem é demonstrada avaliando-o em relação a vários metamodelos específicos da metodologia existentes. Primeiro, o metamodelo é construído por uma combinação de análise de baixo para cima e de cima para baixo e melhores práticas. Os conceitos assim obtidos e seus relacionamentos são avaliados por mapeamento para dois metamodelos orientados a agentes: TAO e Islander.

Além disso, as estruturas conceituais apresentadas nesta seção não apresentam uma descrição da arquitetura interna dos agentes, seus elementos internos e os respectivos papéis.

3.2. Linguagens de Modelagem

Várias linguagens foram propostas para a modelagem do MAS. No entanto, eles não suportam a modelagem de diferentes arquiteturas internas de agentes disponíveis em Russell e Norvig (2003) e Weiss (1999). Além disso, eles têm várias desvantagens que justificaram a escolha do MAS-ML a ser estendido para modelar diferentes arquiteturas de agentes.

O trabalho de Odell et al. (2000) apresenta a linguagem AUML. Essa linguagem de modelagem visa fornecer semântica semiformal e intuitiva através de uma notação gráfica amigável. A AUML não fornece elementos para representar a próxima função, planejar, formular a função de problema, formular a função de meta e função de utilidade.

Wagner (2003) propõe a linguagem de modelagem AORML, baseada no metamodelo AOR. Essa linguagem não suporta a modelagem dos elementos das arquiteturas internas do agente. Portanto, não é possível diferenciar agentes com arquiteturas reativas e proativas na AORML.

Além disso, os dois idiomas mencionados acima não definem o ambiente como uma abstração, portanto, não é possível modelar a migração do agente de um ambiente para outro. Essa capacidade é inerente à modelagem de agentes móveis (Silva e Mendes, 2003).

Choren e Lucena (2005) apresentam a linguagem de modelagem ANote que envolve um conjunto de modelos chamados visualizações. No ANote, não é possível diferenciar agentes com arquiteturas reativas das proativas. Além disso, o ANote não suporta objetos convencionais usados para modelar entidades não autônomas. A linguagem define vários conceitos relacionados aos agentes, mas o conceito de função do agente não é especificado.

Esse conceito é extremamente importante ao modelar sociedades nas quais os agentes podem desempenhar papéis diferentes ao mesmo tempo. A linguagem de modelagem de agentes (AMOLA) (Spanoudakis e Moraitis, 2008) fornece a sintaxe e a semântica para a criação de modelos de sistemas multiagentes, cobrindo as fases de análise e design do processo de desenvolvimento de software. Ele suporta uma abordagem modular de design de agentes e introduz os conceitos de controle intra e inter-agentes baseados em gráficos de estados. A AMOLA lida com os aspectos individuais e sociais dos agentes. Como o AMOLA não modela percepções, planejamento, função seguinte, formula-problema-função e formula elementos de função de objetivo, a arquitetura interna que usa esses elementos não pode ser representada.

AML (Cervenka, 2012) é uma linguagem de modelagem baseada em um metamodelo que permite a modelagem de unidades organizacionais, relações sociais, papéis e propriedades de papéis. Os agentes AML são compostos por lista de atributos, lista de operações, partes e comportamentos, e sensores e atuadores podem ser modelados próximo aos recursos. O planejamento, a próxima função, a formulação da função do problema, a formulação da função de objetivo e os elementos da função de utilidade não são semanticamente representados; portanto, a arquitetura interna que utiliza esses elementos não é representada. Vale ressaltar que os aspectos semânticos da comunicação são modelados como especializações de elementos existentes na UML, como a invocação de métodos, que não é adequada para modelar a comunicação do agente, por exemplo.

4. Estendendo o Framework TAO

De acordo com Silva et al. (2003), o TAO descreve um agente como um elemento autônomo, adaptável e interativo que tem como características estruturais suas metas e crenças e como características comportamentais de suas ações e planos. Os recursos estruturais de um agente são os usados para armazenar informações sobre outros agentes e o ambiente, sobre os estados que os agentes desejam alcançar e qualquer outra informação útil. Recursos comportamentais são

aqueles relacionados à execução do agente, como as tarefas que o agente pode executar e o mecanismo usado pelo agente para selecionar essas tarefas.

A extensão do TAO foi baseada na inclusão de conceitos usados para definir as arquiteturas internas dos agentes (Seção 2.1). Devido à arquitetura interna escolhida ao desenvolver um agente, alguns dos recursos estruturais e comportamentais definidos anteriormente no TAO para todos os agentes não podem ser modelados ou definidos explicitamente. Nesse contexto, uma nova definição para agente foi formulada da seguinte maneira:

Um agente é um elemento autônomo, adaptável e interativo. Suas características comportamentais e estruturais são predefinidas por sua arquitetura interna da seguinte maneira:

- Os agentes reflexos simples não possuem características estruturais. Suas características comportamentais são caracterizadas por suas percepções e ações. As ações a serem executadas são escolhidas pelas regras de condição-ação.
- Os agentes reflexos baseados em modelos têm crenças que representam suas características estruturais. Suas características comportamentais são compostas de percepções, ações (orientadas por regras de condição-ação) e uma função seguinte.
- Os agentes do MAS-ML têm objetivos e crenças como características estruturais e têm planos e ações como características comportamentais. Os planos são executados para atingir metas e são compostos por ações.
- Os Agentes Baseados em Objetivos têm objetivos e crenças como características estruturais e têm percepções, uma função seguinte, uma função de formulação de objetivos, uma função de formulação de problemas, uma função de planejamento e ações como características comportamentais.
- Os agentes baseados em utilidade têm objetivos e crenças como características estruturais e têm percepções, uma função seguinte, uma função de formulação de objetivos, uma função de formulação de problemas, um planejamento e ações de funções de utilidade como características comportamentais. A Tabela 1 lista os recursos estruturais e comportamentais relacionados a cada arquitetura de agente interno.

Ainda é necessário mudar o conceito da função do agente em relação aos agentes reativos. As características estruturais do papel do agente no TAO são definidas como sendo compostas por crenças e objetivos (Silva, 2004). Podemos acrescentar a esse conceito o fato de que, no caso de Agentes Reflexos Simples, não existem crenças e objetivos, e no caso de agentes reativos baseados no conhecimento, as crenças existem apenas como uma característica estrutural.

Além da abordagem conceitual, Silva et al. (2003) apresentam os conceitos em uma abordagem semi formal que usa modelos para ajudar a descrever os conceitos. Usamos a mesma notação apresentada por Silva et al. (2003) para representar o conceito de agente para cada arquitetura de agente diferente e a função de agente para agentes reflexos simples e agentes reflexivos baseados em modelo. Isso é mostrado porque a representação inicial do MAS-ML no Agente e na Função do Agente, apresentada na Seção 2.2, não é suficiente para representar todas as arquiteturas do agente.

O primeiro modelo define a classe Simple Reflex Agent. A classe do agente Simple Reflex descreve as percepções, ações e relacionamentos iguais para todas as instâncias do agente. O modelo do agente também lista os eventos que os agentes, instâncias da classe SimpleReflexAgent, podem gerar e perceber, e as funções que os agentes podem desempenhar.

A classe Agente Reflex com base em modelo descreve crenças, percepções, próxima função, ações e relacionamentos iguais para todas as instâncias do agente. O modelo do agente também lista os eventos que os agentes, instâncias da classe ModelBasedReflexAgent, podem gerar e perceber, e as funções que os agentes podem desempenhar.

A classe Agente Baseado em Objetivo descreve objetivos, crenças, percepções, ações, função seguinte, função de formulação de objetivo, função de problema de formulação e relações que são iguais para todas as instâncias de agente. O modelo do agente também lista os eventos que os agentes, instâncias da classe GoalBasedAgent, podem gerar e perceber, e as funções que os agentes podem desempenhar.

A classe Agente Baseado em Utilitários descreve objetivos, crenças, percepções, ações, próxima função, função de objetivo de formulação, função de problema de formulação, função de problema de função, função de utilitário e relacionamentos que são os mesmos para todas as instâncias do agente. O modelo do agente também lista os eventos que os agentes, instâncias da classe UtilityBasedAgent, podem gerar e perceber, e as funções que os agentes podem desempenhar.

A representação inicial da classe de função do agente não foi alterada e essa representação é usada para todos os agentes proativos. Os modelos usados para representar as funções de agente para a representação de novos agentes são os seguintes.

O modelo de função do Simple Reflex Agent apresenta a classe de função do Simple Reflex Agent e os deveres, direitos, protocolos e compromissos que definem as interações. Ele também identifica os relacionamentos das funções de agente, ou seja, seu proprietário, os agentes e organizações que podem desempenhar a função, os objetos associados à função e as associações com outras funções. Todas as instâncias de função da classe de função têm as mesmas propriedades e relacionamentos.

O modelo de Função do agente reflexo baseado em modelo apresenta a classe Função do agente baseado em modelo e as crenças, deveres, direitos, protocolos e compromissos que definem as interações. Ele também identifica os relacionamentos das funções de agente, ou seja, seu proprietário, os agentes e organizações que podem desempenhar a função, os objetos associados à função e as associações entre as funções. Todas as instâncias de função da classe de função têm as mesmas propriedades e relacionamentos.

Os Agentes Baseados em Objetivos e os agentes utilitários usam a mesma notação apresentada inicialmente por Silva et al. (2003) e apresentado na Seção 2.2.

5. Estendendo a Linguagem MAS-ML

Esta seção apresenta as extensões do MAS-ML para apoiar a modelagem de agentes usando diferentes arquiteturas internas: Reflexo Simples, Reflexo Baseado em Modelo, Baseado em Objetivo e Baseado em Utilidade. A nova versão do MAS-ML é chamada MAS-ML 2.0.

Segundo UML (2009), valores marcados, estereótipos e restrições são mecanismos de extensão. Além disso, a adaptação das metaclasses existentes e a definição de novas metaclasses também podem ser usadas. Os estereótipos e a definição de novas metaclasses foram usados para representar agentes reflexos simples, agentes reflexivos baseados em modelos, agentes baseados em objetivos com agentes planejados e baseados em serviços públicos. Seguindo as definições de arquitetura apresentadas na Seção 2, sentimos a necessidade de definir as seguintes características: percepção, função seguinte, função formular meta, função formular problema, função formular problema, planejamento e função utilidade.

A Fig. 7 ilustra o metamodelo MAS-ML 2.0 e destaca usando retângulos brancos de linha dupla as extensões feitas no metamodelo MAS-ML. Um subconjunto de metaclasses UML é desenhado como retângulos de linha única branca, as metaclasses MAS-ML são representadas como retângulos de linha única cinza e os estereótipos MAS-ML são representados com um retângulo cinza com bordas arredondadas.

As percepções de um agente obtêm informações sobre o ambiente e / ou outros agentes. Como não há nenhuma metaclasse no MAS-ML que possa ser usada para representar esse conceito, a metaclasse AgentPerceptionFunction foi criada para representar a percepção do agente. A metaclasse AgentPerceptionFunction está relacionada à metaclasse Environment, pois o agente percebe o ambiente e também à metaclasse Constraint para restringir as informações que podem ser percebidas pelos sensores do agente.

A tarefa de planejamento resulta em uma sequência de ações para atingir uma meta (Russell e Norvig, 2003). Além disso, são observadas as seguintes propriedades: (i) diferentemente de um plano (representado por Agent-Plan no

metamodelo MAS-ML original), a sequência de ações é criada em tempo de execução; e (ii) ao contrário de uma ação simples (representada por AgentAction no metamodelo MAS-ML), a ação de planejamento tem um objetivo associado a ela. Assim, a nova metaclasse AgentPlanningStrategy foi criada para representar a funcionalidade de planejamento. Um relacionamento de associação entre o AgentPlanningStrategy e o AgentPlan foi definido para representar a ação de criação de planos. As metaclasses AgentPerceptionFunction e AgentPlanningStrategy estendem a metaclasse BehaviouralFeature.

A próxima função, função de objetivo de formulação, função de problema de formulação e função de utilidade são ações especiais do agente que dependem da arquitetura interna do agente. Os estereótipos <<próxima função>> <<função do objetivo do formulário>>, <<função do problema de formulário>> e <<função da facilidade>> estereótipos foram criados e relacionados à metaclasse do AgentAction. Finalmente, as regras de ação de condição usadas pelos agentes reflexos simples baseados em modelo dos agentes reflexos podem ser representadas usando a representação de ação do agente (Silva et al., 2008a), portanto, não é explicitamente representada no metamodelo.

5.1. Representação estática do AgentClass

Os novos recursos estruturais e comportamentais introduzidos no metamodelo MAS-ML são usados para modelar os diferentes tipos de agentes e impactam a representação da metaclasse AgentClass em diagramas estáticos. Nas próximas seções, a nova representação é mostrada. Cada agente desta seção foi criado usando o nome padrão: Agent-Class InstanceName. Esse nome pode ser substituído quando os agentes do domínio do aplicativo estão sendo definidos (a Seção 7.2.1 descreve os agentes que usam a notação genérica apresentada nas seções a seguir).

5.1.1. Estrutura do Agente Reflexivo Simples

A representação para um Agente Reflexo Simples (Fig. 8) não inclui nenhum elemento estrutural no compartimento do meio, mas no compartimento inferior as percepções e ações, conduzidas pelas regras de condição-ação e não por um plano específico, são representadas.

5.1.2. Estrutura do Agente Reflexivo Baseado em Modelo

O Agente Reflex baseado em modelo representa uma atualização sobre o Simple Reflex Agent. Assim, a definição para o elemento de ação é mantida a mesma. Além disso, crenças representando o estado e a próxima função estão incluídas. A Fig. 9 apresenta a representação gráfica do AgentClass para um Agente Reflex baseado em modelo.

5.1.3. Estrutura do Agente Baseado em Objetivos com Planos

Os Agentes Baseados em Metas com plano possuem a mesma estrutura proposta inicialmente por Silva e Lucena (2004),

incluindo metas, crenças, ações e plano. A figura 2 (A) mostra a representação gráfica desse agente.

5.1.4. Estrutura do Agente Baseado em Objetivos com Planejamentos

Os agentes baseados em metas com planejamento incorporam complexidade adicional na representação do agente. Primeiramente, as metas são consideradas para orientar o comportamento do agente. Para manipular consistentemente metas e estados, o comportamento do agente é aprimorado com os elementos <<perceives>>, <<formulate-goal- function>> e <<formulate-problem-function>>. O elemento <<next-function>> já existente é mantido. Esta função recebe a percepção atual e as crenças que devem ser atualizadas (estado).

Além disso, em vez de representar planos pré-estabelecidos, a atividade de planejamento é incorporada. Esta atividade envolve uma meta e usa as ações disponíveis para criar uma sequência de ações. A Fig. 10 ilustra a AgentClass para um agente baseado em metas usando o planejamento.

5.1.5. Estrutura do Agente Baseado em Utilidade

A representação para o Agente Baseado em Utilidade consiste em uma especialização do Agente Baseado em Objetivo com o planejamento. Durante o planejamento, os agentes podem ser vinculados para atingir mais de uma meta. Nesse caso, é possível a ocorrência de objetivos conflitantes ou a existência de vários estados que atendem aos objetivos. Portanto, a função de utilidade é incorporada à estrutura do agente para avaliar o grau de utilidade dos objetivos associados. Assim, o elemento <<utility-function>> é adicionado para representar a função responsável pela otimização do desempenho do agente. A representação gráfica do AgentClass para um agente proativo com base na utilidade é ilustrada na Fig. 11.

5.2. Representação Estática AgentRoleClass

Um AgentRoleClass no MAS-ML é representado por um retângulo sólido com uma curva na parte inferior. Semelhante à representação de classe, possui três compartimentos separados por linhas horizontais. O compartimento superior contém o nome da função do agente exclusivo em seu espaço para nome. O compartimento intermediário contém uma lista de objetivos e crenças associadas ao papel e, abaixo, uma lista de deveres, direitos e protocolos.

Os agentes reativos não têm objetivos explícitos e, mais particularmente, os agentes reflexos simples não têm crenças. Assim, sua representação de papéis deve ser adaptada e sua representação é ilustrada na Fig. 12.

Além da representação dos papéis dos agentes reflexos simples, os papéis dos agentes reflexivos baseados em modelo incluem crenças para lidar parcialmente com ambientes observáveis. A representação da função de agente neste caso é ilustrada na Fig. 13.

Os recursos das funções de agente em outras arquiteturas permanecem inalterados, pois ambos definem crenças e objetivos. As mudanças estruturais em relação à entidade AgentRoleClass afetam os diagramas de Organização e Funções.

5.3. Representação de AgentClass no diagrama de sequência

Semelhante aos diagramas estáticos, as novas definições da Classe de Agente influenciam novas representações de seus recursos comportamentais. Na Seção 5.3.1, apresentamos as representações dos novos elementos que foram definidos para ser possível modelar a execução das diferentes arquiteturas de agentes internos, modeladas nas Seções 5.3.2-5.3.5.

5.3.1. Apresentando os novos elementos no Diagrama de Sequência

A percepção do agente é representada no diagrama de sequência MAS-ML por uma seta com a cabeça aberta deixando o agente para o ambiente, juntamente com o estereótipo <<perceives>>, o nome da percepção e os elementos que o agente percebe (Fig. 14)

Para representar as ações executadas por um agente reativo, é necessário representar suas condições associadas. A Fig. 15 ilustra uma ação que um agente reativo pode executar.

A próxima função dos agentes reativos é representada no diagrama de seqüências do MAS-ML 2.0 por uma seta fechada com a cabeça cheia, que começa no agente e termina no agente. O estereótipo <<next-function>> é usado seguido pelo nome da função. A Fig. 16 ilustra a próxima função no diagrama de sequência.

Portanto, se um Simple Reflex Agent for modelado, primeiro teremos sua percepção e depois suas ações guiadas pelas regras de condição-ação. No caso de um agente reflexo baseado em modelo, primeiro temos a percepção, depois a próxima função e, finalmente, suas ações guiadas pelas regras de condição-ação.

A próxima função dos agentes pró-ativos é executada antes da função de objetivo-formulado e é usada por dois tipos de agentes proativos neste artigo: função de objetivo-formulado e função de problema, conforme ilustrado nas Figs. 17 e 18.

No caso de agentes capazes de planejar em tempo de execução, a sequência de ações que o agente executará não poderá ser modelada no tempo de design. Nesse caso, o planejamento é representado por uma ponta de seta fechada que começa e termina no agente adornado com o estereótipo <<planeamento>>. As ações que podem ser usadas para planejar e atingir o (s) objetivo (s) são representadas como em Silva e Lucena (2004). Opcionalmente, uma nota textual

pode especificar o critério ou algoritmo usado para executar o planejamento. A Fig. 19 ilustra o planejamento no diagrama de sequência MAS-ML 2.0.

O elemento da função utilidade é representado no diagrama de sequência por uma seta com cabeça cheia que começa no agente e termina em si mesma, junto com o estereótipo <<utility-function>>. A Fig. 20 ilustra a representação da função de utilidade no diagrama de sequência MAS-ML 2.0.

No MAS-ML 2.0, as ações do agente são modeladas usando o elemento de iteração e o fragmento combinado, que já estão definidos na UML. Essa representação permite a modelagem de qualquer combinação de ações. Um exemplo é mostrado na Fig. 21. Como a sequência de ações para os agentes com planejamento é gerada em tempo de execução, a modelagem dessa sequência não é necessária.

O agente baseado em metas com planejamento utiliza a representação proposta por Silva e Lucena (2004), bem como o plano definido durante a fase de projeto. No caso do Agente Baseado em Objetivos com planejamento, ele inicialmente executa a percepção, depois a próxima função, a função de meta de formulação e a função de problema de formulação. A função de planejamento é executada e sua saída é uma sequência de ações possíveis.

Por fim, o Agente de Utilidade precisa da percepção, próxima função, função de objetivo de formulação, função de problema de formulação, planejamento, função de utilitário e resulta em ações que são executadas na ordem predefinida.

5.3.2. Representação do Agente Reflexivo Simples no Diagrama de Sequência

O Simple Reflex Agent inicia executando sua percepção seguida pelas ações executadas de acordo com as regras de condição-ação. A Fig. 22 mostra o Simple Reflex Agent em um diagrama de sequência. O agente percebe o ambiente e age executando uma ação que altera o ambiente ou envia uma mensagem para outro agente.

5.3.3. Representação do Agente Reflexivo Simples Baseado em Modelo no Diagrama de Sequência

O Agente Reflex baseado em modelo começa executando sua percepção. Em seguida, a próxima função é executada, seguida pelas ações executadas de acordo com as regras de condição-ação. A Fig. 23 mostra o agente reflexo baseado em modelo em um diagrama de sequência. O agente percebe o ambiente e age executando uma ação que altera o ambiente ou envia uma mensagem para outro agente.

5.3.4. Representação do Agente Reflexivo Simples Baseado em Objetivo no Diagrama de Sequência

O agente baseado em objetivos inicia sua execução percebendo o ambiente. Em seguida, a próxima função é executada, seguida pela função formular objetivo e a função formular problema. As ações são executadas de acordo com a meta e o plano selecionados. A Fig. 24 mostra o agente baseado em objetivos em um

diagrama de sequência. O agente percebe o ambiente e age executando uma ação que altera o ambiente ou envia uma mensagem para outro agente.

5.3.5. Representação do Agente Reflexivo Simples Baseado em Utilidade no Diagrama de Sequência

O Agente Baseado em Utilidade inicia sua execução percebendo o ambiente, executando a próxima função e executando a função de objetivo de formulação e a função de problema de formulação. As ações são executadas de acordo com o plano escolhido que considera a função de utilidade relacionada. A Fig. 25 mostra o agente baseado em utilidade em um diagrama de sequência. Como nos agentes anteriores, o agente percebe o ambiente e age alterando o ambiente ou enviando uma mensagem.

5.4. Representação do AgentClass no Diagrama de Atividade

As características propostas por Silva et al. (2005) e utilizados nos diagramas de atividades foram reutilizados. Assim, cada atividade é representada por um retângulo arredondado. As crenças do agente são representadas por um quadrado com a identificação do agente usado pelas crenças e objetivos no canto superior direito através de uma descrição textual denotada pelo estereótipo <<Goal>>.

5.4.1. Representação de Elementos de Agentes Reativos no Diagrama de Atividades

O diagrama de atividades de agentes reflexos simples e baseados em modelo representa o comportamento da percepção à ação. O comportamento de um Simple Reflex Agent é representado da seguinte forma: a atividade inicial é a percepção do agente e, com base na percepção atual, as regras de ação da condição são usadas para selecionar uma das ações possíveis. Finalmente, a ação selecionada é executada. A Fig. 26 mostra o Simple Reflex Agent no diagrama de atividades.

Por outro lado, o comportamento de um Agente Reflex baseado em modelo é representado da seguinte forma: a atividade inicial é a percepção do agente que pode ser usado pela próxima função para atualizar suas crenças. Depois disso, as regras de ação-condição são responsáveis por selecionar uma das ações possíveis. Finalmente, a ação selecionada é executada. A Fig. 27 mostra o Agente Reflex baseado em modelo no diagrama de atividades.

5.4.2. Representação de elementos de agentes proativos no diagrama de atividades

O diagrama de atividades do agente baseado em metas com planejamento representa o comportamento do agente, da percepção à ação. O comportamento de um agente baseado em objetivos é representado no diagrama de atividades do

MAS-ML 2.0 da seguinte forma: a atividade inicial é a percepção do agente e, depois disso, a próxima função atualiza as crenças com base na percepção atual. A meta de formulação e as funções de problema de formulação são executadas. O planejamento é realizado para determinar as ações que devem ser executadas. Por fim, as ações selecionadas são executadas. A figura 28 mostra o agente baseado em objetivos no diagrama de atividades.

O comportamento de um agente baseado em utilidade é representado no diagrama de atividades da seguinte forma: a atividade inicial é a percepção; a próxima função atualiza as crenças com base na percepção atual. A função formular objetivo e a função formular problema são executadas. O planejamento é realizado para determinar quais ações devem ser tomadas. A função utilidade ajuda na escolha da ação, e as ações selecionadas são executadas. A Fig. 29 mostra o agente baseado em utilidade no diagrama de atividades.

6. Ferramenta MAS-ML

Esta seção apresenta as funções, tecnologias, ferramentas de modelagem existentes para o MAS e detalhes relacionados ao desenvolvimento do ambiente de modelagem do MAS chamado ferramenta MAS-ML. Inicialmente, o processo de geração de ferramentas é descrito. Isto é seguido por uma descrição do desenvolvimento de cada diagrama com os respectivos elementos representados. Por fim, é mostrada uma visão geral da ferramenta MAS-ML.

6.1. Antecedentes do MDA

MDD (desenvolvimento orientado a modelo) é caracterizado pelo foco na modelagem e não na implementação (França e Rumpe, 2007). Essa abordagem fornece automação por meio da implementação de mudanças que podem envolver modelo a modelo ou modelo a texto.

Com base no MDD, o OMG (grupo de gerenciamento de objetos) define uma arquitetura chamada MDA (Model-Driven Architecture) que visa padronizar e facilitar a integração dos recursos utilizados. Essa arquitetura é baseada em um conjunto de padrões, como UML (Unified Modeling Language) com MOF (Meta Object Facility), XMI (XML Metadata Interchange) e CWM (Common Warehouse Metamodel), que oferece alguns modelos ou perfis principais para o desenvolvimento (OMG, 2014a). O OMG também define padrões para o processo de geração de código, que utiliza um conjunto de instruções de entrada (podem ser modelos) e fornece como saída o código-fonte relacionado. Esse padrão estabelece que os conceitos do OMG podem ser aplicados a diferentes linguagens de modelagem, ou seja, não estão

necessariamente vinculados a uma plataforma específica, como a UML (OMG, 2014b).

O processo de geração de código pode ser dividido em estágios. Os conceitos de cada fase do processo de geração de código são ilustrados na Fig. 30 e descritos a seguir:

- CIM (Modelo Independente de Computação): Concentra-se no entendimento de requisitos para especificar o domínio do aplicativo.
- PIM (Platform Independent Model): define as entidades do sistema e seus relacionamentos associados.
- PSM (Platform Specific Model): A geração de código será realizada em uma linguagem de programação específica e pode usar componentes, estruturas, middleware e bibliotecas. Portanto, é necessário definir definições a esse respeito.
- PDM (Platform Definition Model): como o PSM é escolhido, é necessário criar relações entre os elementos presentes no PIM e no PSM.

6.2. Ferramentas de modelagem para MAS

Com base na abordagem MDA, algumas ferramentas de modelagem foram propostas para modelar o MAS. O MAS-ML tem uma ferramenta disponível; foi proposto por De Maria et al. (2005) para uma abordagem MDA. No entanto, esta ferramenta não possui código disponível; consequentemente, a aplicação da extensão proposta neste trabalho e quaisquer alterações não podem ser aplicadas nesta ferramenta. Cervenka (2012) descreve perfis definidos para IBM Rational Rose, Enterprise Architect e StarUML; todas as três ferramentas fornecem os mecanismos de customização para definir e aplicar perfis UML nos modelos de aplicativos; portanto, a implementação de perfis UML para AML é direta. AUML cita um conjunto de ferramentas que podem suportar AUML; no entanto, é necessária uma definição de perfil para introduzir a sintaxe AUML nas ferramentas UML.

Além disso, as ferramentas de modelagem apresentadas nesta seção não apresentam uma descrição da arquitetura interna dos agentes, seus elementos internos e as respectivas funções. A ferramenta MAS-ML pode representar as arquiteturas internas de agentes reflexos simples, agentes reflexivos baseados em modelo, agentes baseados em objetivos e agentes utilitários.

6.3. Implementação da ferramenta

A ferramenta MAS-ML é um ambiente de modelagem que suporta a modelagem de sistemas multi-agente com base no metamodelo do MAS-ML. Depois de definir o metamodelo da linguagem, ou seja, a sintaxe abstrata da linguagem, sentimos a necessidade de implementar uma ferramenta de modelagem para dar suporte à sintaxe concreta da linguagem. A ferramenta foi desenvolvida com base na abordagem orientada a modelo proposta pelo Graphical Modeling Framework (GMF, 2011). O processo de desenvolvimento proposto pela estrutura é

dividido em seis etapas (Modelo de Domínio, Modelo Gráfico de Definição, Modelo de Definição de Ferramenta, Modelo de Mapeamento e Modelo de Geração de Domínio). Essas seis etapas foram aplicadas no desenvolvimento da ferramenta MAS-ML 2.0, da seguinte maneira:

Modelo de domínio. Primeiro, o metamodelo MAS-ML foi especificado usando o EMOF (Essential Meta-Object Facility), uma linguagem de definição de metamodelo. Os estereótipos predefinidos foram adicionados ao ActionClass pelo recurso ActionSemantics, da seguinte forma: 0 - sem estereótipo, 1 - próxima função, 2 - função de utilidade, 3 - formular problema-função e 4 - formular objetivo-função.

Modelo Gráfico de Definição. Nesta etapa, as entidades, suas propriedades e relacionamentos foram definidos seguindo o metamodelo.

Modelo de definição de ferramenta. Os elementos usados em cada paleta são definidos nesta etapa. Esta etapa recebe o modelo de domínio e o modelo de definição citados anteriormente.

Modelo de Mapeamento. Nesta etapa, um mapeamento vinculando os modelos de domínio, modelo gráfico e modelo de ferramenta é construído. O mapeamento gerado é usado como entrada do processo de transformação, que cria uma plataforma de modelagem específica. A ferramenta MAS-ML incorpora um mecanismo de verificação de modelo para validar a construção dos diagramas. Um conjunto de seis regras de validação definidas usando a OCL (Object Constraint Language) (OCL, 2011) é usado para verificar se o modelo foi formado corretamente (Tabela 2). Os diagramas de classe, organização, função, sequência e atividade usam essas regras.

Geração de ferramentas. O próximo passo segue a abordagem generativa proposta em Czarnecki e Eisenecker (2000), onde o código é gerado com base no modelo. Assim, o GMF é usado, pois fornece um componente generativo e uma infraestrutura de tempo de execução para desenvolver editores gráficos. O processo de desenvolvimento de cada diagrama é descrito abaixo.

- Diagrama de classe. O diagrama de classes da ferramenta MAS-ML foi desenvolvido de acordo com as etapas listadas na Seção 6. Seu modelo de domínio foi criado com as entidades e os relacionamentos já definidos
- no MAS-ML 2.0, e outras etapas foram seguidas, culminando na geração da ferramenta. O diagrama de classes contempla os seguintes elementos: (i) nós: classe, AgentClass, OrganizationClass, EnvironmentClass, ActionClass, PlanClass, propriedade, operação, objetivo, crença, percepção e planejamento; (ii) Relacionamentos: Associação, Habitação, Dependência, Generalização, Agregação e Composição; e (iii) notas.
- Diagrama de organização. O Modelo de Domínio gerado ao criar o Diagrama de Classes foi usado para criar o Diagrama da Organização

que contempla o seguinte: (i) a propriedade e o relacionamento dos relacionamentos, (ii) a função do agente e a função do objeto e (iii) o agente, a organização e o ambiente. Esses novos elementos são abordados no modelo de domínio e no modelo gráfico que foram usados no diagrama de organização. No entanto, alguns ajustes no modelo de domínio foram necessários. A relação de habitar teve sua semântica alterada para permitir que agentes e organização habitassem o ambiente. A associação, dependência, generalização, agregação e composição foram removidas porque não fazem parte do Diagrama da Organização.

- Diagrama de Funções. O mesmo modelo de domínio foi usado para criar o diagrama de funções. Os elementos que aparecem nos diagramas de organização e função foram preservados. As entidades preservadas são Função do Agente e Função do Objeto e os relacionamentos preservados são os de Associação, Controle, Dependência, Generalização e Agregação. A representação gráfica dos elementos, relacionamentos e diagramas da ferramenta MAS-ML é apresentada através de um estudo de caso nas próximas seções.
- Diagrama de sequência. A criação do diagrama de sequência foi iniciada através do mapeamento de todos os elementos apresentados no metamodelo MAS-ML para a linguagem Emfatic 2.0 (Emfatic, 2014). Neste processo, associamos os elementos apresentados no diagrama de seqüência com suas representações gráficas. Os elementos apresentados no diagrama são os seguintes: Classe, AgentClass, OrganizationClass, EnvironmentClass, ControlStructures, AgentRoleClass, PlanClass, ActionClass, Planning e Perception. As representações gráficas de Class, AgentClass, AgentRoleClass, OrganizationClass e EnvironmentClass no diagrama são ilustradas na Fig. 31.

Os relacionamentos presentes no Diagrama de Sequência são os seguintes: Ação, AgentMessage (Ativar, Cancelar, Compromisso, Criar, Desativar, Padrão, Destruir), Alterar (DeactivateReactivate, DeactivateCreate, DestroyCreate, DestroyReactivate), For, If, Else, ObjectMessage, Perception , Plano e Planejamento, conforme ilustrado na Fig. 32.

- Diagrama de atividades. O diagrama de atividades foi criado de acordo com a mesma abordagem seguida na criação do diagrama de seqüências. O metamodelo do MAS-ML 2.0 foi transcrito para a linguagem Emfatic, os elementos foram identificados e suas representações definidas. Os elementos são Ambiente, Organização, Agente de Função, Estado Inicial, Estado Final, Decisão, Intercalação, Junção, Forquilha e Ação. Ambiente, Organização, Função e Agente. Estado inicial e estado final marcam o início e o final da execução da

atividade, respectivamente. Decisão representa o condicional em que o fluxo pode seguir um determinado caminho, dependendo de um valor predeterminado. O elemento de intercalação determina o final de uma decisão, onde os caminhos que podem ser percorridos por uma Decisão convergem. Fork representa o início de uma execução em paralelo e Join representa o ponto de sincronização entre execuções em paralelo. O elemento Action representa a ação executada pelo agente e o fluxo de relacionamento indica o fluxo de ações que o agente executa. Na Fig. 33, mostramos os principais elementos apresentados no diagrama de atividades usando a mesma representação da própria ferramenta.

6.4. Visão geral da ferramenta MAS-ML

O ambiente implementado é um plug-in da plataforma Eclipse (Eclipse, 2013). Assim, o usuário pode usar o mesmo ambiente para criar modelos e artefatos de código, utilizando os recursos oferecidos pela plataforma. Dado que muitas estruturas, incluindo JADE, Jadex (Braubach et al., 2004) e Jason (Bordini et al., 2007), são baseadas na plataforma Java, a integração de estruturas de diferentes fontes e propósitos ajuda os desenvolvedores a implementar o MAS e favorece a geração de código no mesmo ambiente de desenvolvimento. A Fig. 34 mostra uma vista do diagrama de classes da ferramenta MAS-ML. Os principais componentes da interface são destacados por letras. Cada componente tem uma função específica na ferramenta, de acordo com o seguinte:

Explorador de Pacotes (A). Para cada novo projeto de modelagem, os arquivos usados durante o processo de modelagem são criados e importados, por exemplo, bibliotecas, documentos de texto e imagens. O explorador de pacotes incorpora as principais funcionalidades permitidas em uma estrutura em árvore de arquivos, a fim de melhorar seu gerenciamento e manipulação.

Vista de modelagem (B). Os modelos criados devem necessariamente ser visualizados para atender a dois requisitos básicos: entendimento e comunicação. Nesse sentido, a visualização de modelagem permite que os desenvolvedores visualizem e editem os modelos interativamente.

Nós da paleta (C). Os construtores que fazem parte dos diagramas propostos pelo MAS-ML são a paleta de nós e relacionamentos. Assim, os desenvolvedores podem criar instâncias desses construtores que são exibidas na visualização de modelagem. É possível identificar na Fig. 34 alguns desses elementos, como ActionClass e AgentClass, por exemplo.

Paleta de Relacionamento (D). Os relacionamentos que podem ser estabelecidos entre os fabricantes presentes nos nós estão disponíveis no relacionamento da paleta. Na Fig. 34, é possível identificar algumas dessas relações, por exemplo, a associação estabelecida entre esses dois agentes (isto é, AgentA e AgentB) mostrada na visualização de modelagem.

Vista de propriedades (E). As preocupações sempre presentes durante o desenvolvimento do metamodelo MAS-ML são: (i) a identificação e representação das características das propriedades das metaclasses MAS-ML; e (ii) a organização e o gerenciamento das propriedades herdadas das metaclasses são estendidos da UML para evitar a criação de conflitos ou inconsistências. Essas propriedades definem exatamente os modelos e são usadas, por exemplo, para distinguir os modelos apresentados nos diagramas. Observando a Figura 34, a diferença entre os dois agentes apresentados na visualização de modelagem é a diferença entre as Strings "AgentA" e "AgentB" atribuídas à sua propriedade name. Assim, a importância da visualização de propriedades é evidente, pois permite a manipulação das propriedades do modelo. Essa visualização exibe as propriedades definidas no metamodelo MAS-ML quando o modelo é exibido e selecionado na visualização de modelagem. Na Fig. 34, as propriedades do AgenteB.goal02 podem ser vistas.

Visualização de problemas (F). Dada a necessidade de validar os modelos criados, a ferramenta fornece aos desenvolvedores uma funcionalidade de validação de modelo. Ou seja, a ferramenta permite que os desenvolvedores verifiquem se o modelo criado possui (ou não) inconsistências, considerando o metamodelo MAS-ML. Se alguma inconsistência for detectada, elas serão relatadas na visualização do problema. A figura 26 mostra alguns problemas capturados. Nesse caso, as seguintes regras bem formadas não são respeitadas: (i) todo agente deve ter uma ação, (ii) todo agente deve ter uma meta, (iii) todo agente deve ter um plano e assim por diante. Essas inconsistências são comparadas com o modelo apresentado em (B). Esse recurso é particularmente importante para permitir o uso da modelagem de MASs no contexto do desenvolvimento orientado a modelos, no qual os modelos são vistos como artefatos de primeira ordem. Modelos inconsistentes prejudicam a transformação de modelos no desenvolvimento de software centralizado em modelos.

Vista de estrutura de tópicos (G). Uma visão geral da distribuição dos elementos do modelo pode ser vista na vista de estrutura de tópicos.

O código da ferramenta MAS-ML e os plug-ins gerados estão disponíveis em https://sites.google.com/site/uecegessi/masmltool.

7. Estudo de Caso

O aplicativo TAC-SCM é usado para ilustrar os benefícios do TAO e do MAS-ML 2.0, onde agentes com diferentes arquiteturas são solicitados para modelar estratégias diferentes para a solução do problema. A Seção 7.1 descreve o ambiente TAC-SCM, a Seção 7.2 mostra os agentes e as funções de agente usando os modelos TAO (os modelos TAO foram apresentados na Seção 4) e a Seção 7.3 mostra os modelos MAS realizados na ferramenta de modelagem MAS-ML que segue a especificação do MAS-ML 2.0.

7.1. TAC-SCM

O TAC (Trading Agent Competition) (Wellman et al., 2002) é um ambiente que permite a realização de leilões simultâneos, técnicas de teste, algoritmos e heurísticas para uso em negociação. Existem dois tipos de jogos em competição: TAC-Classic (Wellman et al., 2002) e TAC-SCM (Sadeh et al., 2003). O TAC-SCM refere-se ao planejamento e gerenciamento das atividades da organização em uma cadeia de suprimentos. O cenário TAC-SCM foi projetado para capturar os desafios em um ambiente integrado para aquisição de matérias-primas, produção e entrega de produtos acabados aos clientes. Esse ambiente é altamente dinâmico, estocástico e estratégico (Arunachalam, 2004).

O jogo inicia quando um ou mais agentes se conectam a um jogo do servidor. O servidor simula fornecedores e clientes, fornecendo um banco, fabricação e serviço de armazenamento de mercadorias para agentes individuais. O jogo termina em um número fixo de dias simulados e, no final, o agente com maior quantia em dinheiro no banco é o vencedor (Collins et al., 2006).

7.2. Modelando TAC-SCM com TAO e MAS-ML 2.0

Em um ambiente TAC-SCM, é possível identificar a General Store da organização principal e suas duas suborganizações, Livraria importada e Livraria de segunda mão, desempenhando as funções Mercado de bens especiais e Mercado de bens usados, respectivamente. A modelagem do ambiente TAC-SCM e da organização General Store usa modelos TAO e são mostrados abaixo. Além disso, neste sistema são identificados cinco tipos de agentes: DeliveryAgent, SellerAgent, BuyerAgent, SupplierAgent e ManagerAgent. Todos esses agentes, descritos em detalhes na próxima seção, habitam o ambiente TAC-SCM.

A Fig. 35 representa o diagrama de organização para o TAC-SCM MAS. Este diagrama representa a TacOrganization e descreve os agentes e funções de agente no ambiente específico e seus relacionamentos de habitar (todos os agentes, todas as funções de agente e a organização habitam TacEnvironment), a TacOrganization possui cada função de agente e cada agente desempenha sua respectiva função de agente em TacOrganization contexto.

Os agentes e suas funções de agente são apresentados nas próximas subseções. O DeliveryAgent é apresentado na Seção 7.2.1, o BuyerAgent é apresentado na Seção 7.2.2, o SellerAgent é apresentado na Seção 7.2.3, a Seção 7.2.4 mostra o ProductionAgent e a Seção 7.2.5 apresenta o ManagerAgent. Esta seção apresenta o modelo TAO e a modelagem de cada agente em diagramas estáticos, seguidos pelo diagrama de sequência e, finalmente, no diagrama de atividades.

7.2.1. Modelagem DeliveryAgent no MAS-ML 2.0

O TAC (Trading Agent Competition) (Wellman et al., 2002) é um ambiente que permite a realização de leilões simultâneos, técnicas de teste, algoritmos e heurísticas para uso em negociação. Existem dois tipos de jogos em competição: TAC-Classic (Wellman et al., 2002) e TAC-SCM

(Sadeh et al., 2003). O TAC-SCM refere-se ao planejamento e gerenciamento das atividades da organização em uma cadeia de suprimentos. O cenário TAC-SCM foi projetado para capturar os desafios em um ambiente integrado para aquisição de matérias-primas, produção e entrega de produtos acabados aos clientes. Esse ambiente é altamente dinâmico, estocástico e estratégico (Arunachalam, 2004). O jogo inicia quando um ou mais agentes se conectam a um jogo do servidor. O servidor simula fornecedores e clientes, fornecendo um banco, fabricação e serviço de armazenamento de mercadorias para agentes individuais. O jogo termina em um número fixo de dias simulados e, no final, o agente com maior quantia em dinheiro no banco é o vencedor (Collins et al., 2006).

7.2.2. VendedorModelagem em MAS-ML 2.0

Em um ambiente TAC-SCM, é possível identificar a General Store da organização principal e suas duas suborganizações, Livraria Importada e Livraria de Segunda Mão, desempenhando as funções Mercado de Bens Especiais e Mercado de Bens Usados, respectivamente. A modelagem do ambiente TAC-SCM e da organização General Store usa modelos TAO e são mostrados abaixo. Além disso, neste sistema são identificados cinco tipos de agentes: DeliveryAgent, SellerAgent, BuyerAgent, SupplierAgent e ManagerAgent. Todos esses agentes, descritos em detalhes na próxima seção, habitam o ambiente TAC-SCM.

Fig. 35 depicts the Organization Diagram for TAC-SCM MAS. This diagram represents the TacOrganization and describes the agents and agent roles in the specific environment and their relationships of inhabit (all agents, all agent roles and the organization inhabit TacEnvironment), TacOrganization ownership each agent role and each agent plays its respective agent role in TacOrganization context. The agents and their agent roles are presented in next subsections. The DeliveryAgent is presented in Section 7.2.1, the BuyerAgent is presented in Section 7.2.2, SellerAgent is presented in Section 7.2.3, Section 7.2.4 shows the ProductionAgent and Section 7.2.5 presents the ManagerAgent. This section presents the TAO template and the modelling of each agent in static diagrams, followed by the Sequence Diagram and finally in the Activity Diagram.

A arquitetura de cada agente, escolhida de acordo com o papel desempenhado pelo agente no jogo, é discutida a seguir.

O DeliveryAgent foi modelado com a representação do agente MAS-ML original e deve atingir o objetivo de entregar produtos aos consumidores, para que ele tenha um objetivo específico que possa ser alcançado com uma sequência de ações. Conseqüentemente, a classe DeliveryAgent pode ser representada pelo modelo do agente MAS-ML, onde os elementos são identificados para DeliveryAgent. Observe que não há eventos gerados e percebidos (o mesmo se aplica a outros agentes desta seção). A representação do modelo DeliveryAgent é mostrada abaixo.

Sua representação nos diagramas estáticos do MAS-ML 2.0 é mostrada na Fig. 36. Ela é definida de acordo com a arquitetura do MAS-ML e possui uma meta denominada deliveryProductsClient que está relacionada ao plano de deliveryProducts, o deliveryBeliefs é definido para o agente, as ações CheckDeliveryCurrentDate, CheckDeliveryAvailableProducts e DeliveryProduct estão disponíveis e estão associados ao plano deliveryProducts nessa mesma sequência.

O DeliveryAgent tem uma função de agente associado, o Deliverer. Sua representação nos modelos de TAO é mostrada abaixo, onde são descritas suas crenças, ações e relacionamentos. O protocolo de comunicação foi definido com a Fipa e os compromissos são definidos para a entrega.

Sua representação nos diagramas estáticos do MAS-ML 2.0 é mostrada na Figura 37. As crenças da função do agente são definidas como deliveryBeliefs e o Deliverer possui três ações de tarefas: checkDeliveryCurrentDate, checkDeliveryAvailableProducts e deliveryProducts. A troca protocolo-mensagem tem o rótulo chamado request e o conteúdo tem o nome da ação respectiva, e a função que recebe as mensagens é Producer. e deliveryProductsClient. Por fim, o DeliveryAgent envia uma mensagem ao TAC_Environment para entregar um computador com um ID específico. O DeliveryAgent está associado à função de agente do Deliverer.

O diagrama de atividades do DeliveryAgent mostra a execução do plano deliveryProduct e, associada a esse plano, uma sequência de ações é executada na seguinte ordem: checkDeliveryCurrentDate, checkAvailableProducts e deliveryProductsClient. Esse plano está associado ao objetivo de deliveryProductsToClient e o agente deve estar associado à

função de agente do Deliverer para executá-lo. A Fig. 39 mostra o diagrama de atividades do DeliveryAgent.

7.2.3. Modelagem BuyerAgent

O BuyerAgent escolhe quando fazer novos pedidos de componentes e efetuar o pagamento. Como os agentes reflexos respondem rapidamente às percepções (Weiss, 1999), o BuyerAgent deve ser modelado como agentes reflexos devido à necessidade de respostas rápidas. Os modelos TAO da classe BuyerAgent (agente reflexo baseado em modelo) são mostrados abaixo:

A Fig. 44 mostra o BuyerAgent (agente reflexo baseado em modelo) em diagramas estáticos do MAS-ML 2.0. Observe que os agentes reflexos não têm plano e objetivos, e um novo componente percebe que está presente em ambos. BuyerAgent tem as seguintes ações: requestQuotation; licitar e pagar; ele tem uma próxima função nextFuncBuyer que atualiza suas crenças com as percepções e realiza inferência e pode perceber supplierQuotation, supplierConfirmation, manager_requirement. As crenças deste agente são salvas em um arquivo de prólogo chamado opinionsBuyer.pl.

O BuyerAgent tem uma função de agente associada, a função de agente do Comprador. Sua representação nos modelos de TAO é mostrada abaixo, onde são descritas suas crenças, ações e relacionamento com o TacOrganization. O protocolo de comunicação foi definido com a Fipa e os compromissos são definidos para fazer compras.

Suas representações nos diagramas estáticos do MAS-ML 2.0 são mostradas na Figura 45. As crenças do comprador são definidas como crenças do comprador e o comprador tem três ações corretas: requestQuotes, offer e pay. O protocolo para troca de mensagens SimpleNegotiation tem o rótulo chamado Request, o conteúdo tem o nome da ação respectiva e a função que recebe as mensagens é Supplier.

O diagrama de sequência da figura 46 ilustra o comportamento do BuyerAgent. Inicialmente, a percepção é executada e as informações de SupplierQuotation, supplierConfirmation e managerRequest são percebidas; a próxima função (nextFuncBuyer) é executada e as crenças atualizadas com as percepções e inferências feitas; finalmente, uma das ações possíveis é executada sempre que a regra de ação da condição é verdadeira. Observe que o BuyerAgent está relacionado à função de agente do Buyer.

O diagrama de atividades do BuyerAgent mostra a percepção desse agente e a próxima função executada a seguir. A próxima etapa deste agente é executar uma ação de acordo com as regras de condição-ação. Este agente não possui objetivos e um plano associado a uma sequência de ações. A Fig. 47 mostra o diagrama de atividades do BuyerAgent.

7.2.4. Modelagem do ProductionAgent no MAS-ML 2.0

O agente ProductionAgent precisa atender à demanda atual montando computadores e gerenciando o estoque. Para atingir esse objetivo, ele não pode usar um plano predefinido, porque esse ambiente dinâmico requer um conjunto diferente de ações, dependendo da demanda atual. Assim, SupplierAgent pode ser definido como um agente baseado em metas.

O modelo da classe ProductionAgent é descrito a seguir. Seu objetivo, crenças, percepções, ações e relacionamento com o TacOrganization são descritos. O protocolo de comunicação foi definido pelo FIPA e funções de planejamento e especiais (nextFunction, Formulate-GoalFunction e formateProblemFunction) também são definidas.

A Fig. 48 mostra o ProductionAgent e seus componentes internos em diagramas estáticos do MAS-ML 2.0. É composto pelas seguintes ações: check_available_parts, request_necessary_parts, check_parts, build_parts, test_computer, computer_available, uma próxima função nextFuncProduction que atualiza suas crenças com as percepções e realiza inferência e percebe requestProduction. As crenças desse agente são salvas em um arquivo de prólogo chamado opinionsProduction.pl e tem o objetivo de satisfazer a demanda. O ProductionAgent possui um planejamento de productComputers e está relacionado ao objetivo de satisfazer a demanda, mas uma sequência de ações não foi definida porque é possível apenas em tempo de execução. O formGoalManager e o formProbManager também estão disponíveis.

O ProductionAgent tem uma função de agente associado, o Produtor. Sua representação nos modelos de TAO é mostrada abaixo, onde suas crenças, ações e relacionamentos são descritos. O protocolo de comunicação foi definido com a Fipa e os compromissos são definidos para produzir.

Sua representação em diagramas estáticos do MAS-ML 2.0 é mostrada em Fig. 49. As crenças da função do agente são

definidas como crençasProdução, e o Produtor tem seis funções: checkAvailableParts, requestNecessaryParts, checkParts, buildParts, testComputer e computerAvailable. O protocolo para troca de mensagens possui o rótulo chamado request, o conteúdo foi definido como requestNecessaryParts e a função que recebe as mensagens é comprador.

A Fig. 50 mostra o diagrama de sequência do ProductionAgent. Observe que o comportamento adotado pelo agente começa com sua percepção de productionRequest, após perceber que a próxima função é executada, objetivando atualizar as crenças desse agente. goalFuncProduction e probFuncProduction são executados para fornecer a meta e o problema formulado para o planejamento. O planejamento é executado e a sequência de ações resultantes do planejamento é representada por um loop e as ações associadas às condições permanecem no loop e serão executadas de acordo com o planejamento.

O diagrama de atividades do ProductionAgent mostra a percepção desse agente; a seguir, ele executa o nextFuncProduction para atualizar suas crenças; goalFuncProduction e probFuncProduction são executados para fornecer a meta e o problema formulado para o planejamento. O planejamento é executado e a sequência de ações resultantes do planejamento é representada por um loop, e as ações associadas às condições permanecem no loop e serão executadas de acordo com o planejamento. A Figura 51 mostra o diagrama de atividades do ProductionAgent.

7.2.5. Modelagem ManagerAgent no MAS-ML 2.0

Por fim, o agente ManagerAgent é responsável por gerenciar toda a equipe e alocar os recursos. Esse agente tenta maximizar o lucro e as vendas: observe que esses objetivos podem estar em conflito. Então, a arquitetura mais apropriada nesse caso é baseada no utilitário. O modelo da classe ManagerAgent é mostrado abaixo: seu objetivo, crenças, percepções, ações e relacionamento com o TacOrganization são descritos. O protocolo de comunicação foi definido com a Fipa, e funções de planejamento e especiais (próxima função, formular função de objetivo, formular função de problema e função de utilidade) também são definidas.

A figura 52 mostra o ManagerAgent e seus componentes internos em diagramas estáticos do MAS-ML 2.0. É composta pelas seguintes ações: requestBuyingPrice, requestSellingPrice,

requestSellingComiceer, estimativaBuyingPrice, valuSellingPrice, uma próxima função nextFuncManager que atualiza suas crenças com as percepções e realiza inferência; ele pode perceber BuyerRequest, SellerRequest, ProductionRequest e DeliveryRequest. As crenças deste agente são salvas em um arquivo de prólogo chamado opinionsManager.pl e tem os objetivos de maximizeGain e maximizeSale. O ManagerAgent possui um managerPlanning e está relacionado aos objetivos maximizeGain e maximizeSale, mas uma sequência de ações não foi definida porque é possível apenas em tempo de execução. O formGoalManager, formProbManager e utilityFuncManager também estão disponíveis.

O ManagerAgent tem uma função de agente associado, o Manager. Sua representação nos modelos de TAO é mostrada abaixo, onde suas crenças, ações e relacionamentos são descritos. O protocolo de comunicação foi definido com a Fipa e os compromissos são definidos para produzir.

Sua representação nos diagramas estáticos do MAS-ML 2.0 é mostrada na Fig. 53. As crenças do papel do agente são definidas como crençasManager; o gerente tem quatro funções (requestBuyingPrice, requestSellingPrice, estimativaBuyingPrice e estimativaSellingPrice) e um direito (requestSellingComputer). O protocolo para troca de mensagens (ManagerNegotiation) possui o rótulo chamado request e o conteúdo foi definido como requestBuyingPrice ou requestSellingPrice e a função que recebe as mensagens é comprador ou vendedor, respectivamente.

A figura 54 mostra o diagrama de sequência do ManagerAgent. Observe que o comportamento adotado pelo agente começa com sua percepção de sellerRequest, compradorRequest, productionRequest e deliveryRequest, a próxima função é executada após a percepção e objetiva atualizar as crenças do agente. GoalFuncManager e probFuncManager são executados para fornecer a meta e o problema formulado para o planejamento. O planejamento é executado e a sequência de ações resultantes do planejamento é representada por um loop; o primeiro passo disso
loop é executar a função de utilitário para calcular o grau de utilitário associado a cada ação, e as ações associadas a condições permanecem no loop e serão executadas após o utilityManager.

O diagrama de atividades do ManagerAgent mostra a percepção desse agente; a seguir, ele executa o nextFuncManager para atualizar suas crenças; goalFuncManager e probFuncManager são executados para fornecer a meta e o problema formulado para o planejamento. O planejamento é executado e a sequência de ações resultantes do planejamento é representada por um loop, e as ações associadas às condições permanecem no loop e serão executadas de acordo com o planejamento. A figura 55 mostra o diagrama de atividades do ManagerAgent.

7.3. Modelagem de agentes TAC-SCM com outras linguagens de modelagem MAS derivadas de UML

Nesta subseção, será descrita a modelagem de agentes apresentada neste estudo de caso do TAC-SCM usando diagramas estáticos de outras linguagens que estendem a UML. O objetivo principal é demonstrar o aumento da expressividade com a extensão proposta por este artigo em relação a outras linguagens de modelagem existentes. Obviamente, as lacunas existentes na modelagem de diagramas estáticos têm impacto em outros diagramas de linguagens de modelagem.

As seguintes linguagens de modelagem serão usadas nessa comparação: MAS-ML (em sua forma original, como proposto por Silva, Choren e Lucena (2008a)), AML e AUML. Os agentes SellerAgent, BuyerAgent, DeliveryAgent, ProductionAgent e ManagerAgent serão apresentados para que possam ser comparados com a modelagem do MAS-ML 2.0 no estudo de caso apresentado.

7.3.1. Modelagem de agentes TAC-SCM com versão original do MAS-ML

Como proposto por Silva et al. (2008a), o agente deve ser representado em diagramas estáticos do MAS-ML, contendo objetivos, crenças, planos e ações. Portanto, os agentes aqui foram modelados com essas características. O agente DeliveryAgent, mostrado na Fig. 36, foi modelado com essa arquitetura originalmente proposta pelo idioma, portanto não será modelada novamente.

A figura 56 mostra o BuyerAgent com sintaxe disponível na versão original do MAS-ML; portanto, esse agente tem o objetivo de comprar peças e crenças, além de ter um plano predefinido e as ações de requestQuotation, bid e pay. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um Agente Reflex com base em modelo, para que metas e planos predefinidos não possam

fazer parte dele. Além disso, os agentes reflexos baseados em modelo têm percepções e uma próxima função, que não é possível representar.

A figura 57 mostra o SellerAgent com sintaxe disponível na versão original do MAS-ML; portanto, esse agente tem como objetivo vender produtos de clientes e crenças associadas, além de ter um plano predefinido e as ações de offerProduct, ReceivePayment e mountDeliveryProduct. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um Agente Reflexo Simples, para que objetivos, crenças e planos predefinidos não possam fazer parte dele. Além disso, os agentes reativos têm percepções que não são possíveis de representar.

O ProductionAgent com a sintaxe disponível na versão original do MAS-ML é mostrado na Fig. 58. Esse agente é composto por uma meta de Satisfação da demanda, crençasProdução, um plano predefinido chamado productComputers e as ações: check_available_parts, request_necessary_parts, check_parts, build_parts, test_computer e computer_available . De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um agente baseado em metas, para que planos predefinidos não possam fazer parte dele. Além disso, os Agentes Baseados em Objetivos têm percepções, planejamento, próxima função, função de formulação de objetivos e função de formulação de problemas, que não são possíveis de representar na versão original do MAS-ML.

O ManagerAgent modelado com a sintaxe disponível na versão original do MAS-ML é mostrado na Figura 59. Esse agente é composto pelos objetivos de maximizeGain e maximizeSale, crenças, um plano predefinido chamado managerPlan e request_buying_price actions, request_selling_price, request_selling_computer, estimado_buying_price, e estimativa_venda_venda. De acordo com aquilo apresentado no estudo de caso, esse agente deve ser modelado como um agente baseado em utilitários, para que planos predefinidos não possam fazer parte dele. Além disso, os Agentes de Utilidade têm percepções, planejando a próxima função, a função de objetivo da formulação, função de problema de formulação e função de utilidade, que não é possível representar no MAS-ML original.

7.3.2. Modelagem de agentes TAC-SCM com AUML

Como proposto por Odell et al. (2000), o agente no Diagrama de Classes da AUML é uma entidade que pode consistir em atributos, operações, capacidades, percepções, protocolos, organizações relacionadas e funções relacionadas. Além disso, os agentes podem ter crenças, classes, objetivos e estratégias associadas a ele. Assim, os agentes modelados nesta subseção serão compostos dos recursos propostos no AUML.

A Fig. 60 mostra a modelagem de DeliveryAgent no diagrama de classes da linguagem AUML, e podemos identificar a função de entregador desse agente, além da organização à qual ele pertence (TacOrganization). DeliveryAgent tem crenças, presentes em DeliveryBeliefs, e a meta DeliveryProductsClient. Também possui a estratégia DeliveryProducts, que funciona como um plano associado ao agente. Em DeliveryProducts, as ações associadas estão presentes no compartimento Actions. Nas próximas
função não pode ser representada.

A Figura 61 traz o BuyerAgent modelado com o Diagrama de classes da AUML, e podemos identificar a função de comprador desse agente, além da organização à qual ele pertence (TacOrganization). O BuyerAgent está relacionado às suas crenças presentes no BuyerBeliefs além do BuyerAct e opera como um plano associado ao agente. No BuyerAct, as ações associadas estão presentes no compartimento de ações. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um agente reativo com base no conhecimento, para que planos predefinidos não possam fazer parte dele. Além disso, a próxima função não pode ser representada.

O SellerAgent modelado com o diagrama de classes da AUML é mostrado na Figura 62, onde podemos identificar a função de vendedor desse agente, além da organização à qual ele pertence (TacOrganization). O SellerAgent possui o SellerAct e opera como um plano associado ao agente. No SellerAct, as ações associadas estão presentes no compartimento Ações. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um simples agente reativo, para que planos predefinidos não possam fazer parte dele.

O ProductionAgent modelado no diagrama de classes AUML é mostrado na Figura 63, onde podemos identificar a

função de produtor desse agente, além da organização à qual ele pertence (TacOrganization). As crenças do agente são representadas por ProductionBeliefs e seu objetivo é representado por satisfazerDemand. O ProductionAgent possui os ProduceComputers e opera como um plano associado ao agente. No ProducerComputers, as ações associadas estão presentes no compartimento Ações. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um agente baseado em metas, para que planos predefinidos não possam fazer parte dele. Além disso, a próxima função, a função de objetivo de formulação e a função de problema de formulação não podem ser representadas.

O ManagerAgent modelado com o diagrama de classes da AUML é mostrado na Figura 64, onde a função de gerente para esse agente pode ser identificada além da organização à qual ele pertence (TacOrganization). As crenças do agente são representadas por ManagerBeliefs e as metas são representadas por maximizeSale e MaximizeGain. O ManagerAgent possui o ManagerPlanning, que opera como um plano associado ao agente. No ManagerPlanning, as ações associadas estão presentes no compartimento Ações. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um Agente de Utilidade, para que planos predefinidos não possam fazer parte dele. Além disso, a próxima função, a função objetivo-formulação, a função problema de formulação e a função utilidade não podem ser representadas.

Em geral, os agentes reativos não possuem estratégia (plano), no entanto, foram modelados com as ações em uma classe adornada com o estereótipo de estratégia. Nesta linguagem de modelagem, as crenças são representadas como uma classe, portanto, podem ter operações relacionadas.

7.3.3. Modelagem de agentes TAC-SCM com AML

Na LBC, os agentes consistem em planos, fragmentos de comportamento, percebedores, atuadores, crenças e objetivos. As informações que podem ser percebidas são representadas junto com os percebedores dos agentes, e as ações que os agentes podem executar são representadas junto com seus atuadores. Assim, os agentes modelados nesta subseção devem ter os elementos de LBC necessários para a representação de sua arquitetura.

A Fig. 65 mostra a modelagem de DeliveryAgent no diagrama de classes da linguagem de modelagem AML, onde é possível identificar seu objetivo (deliveryProductsClient) e crenças (deliveryBeliefs), além do plano (DeliveryProducts), relacionado aos fragmentos de comportamento checkDeliveryAvailableProducts, deliveryProducts e checkDeliveryCurrentDate. As pernas dos atuadores também estão disponíveis. A próxima função não pode ser representada na AML e o plano não está relacionado a uma meta específica. Além disso, objetivos e crenças, que são características estruturais dos agentes, podem ser representados em classes específicas que permitem que essas classes tenham operações.

A Figura 66 mostra a modelagem do BuyerAgent no diagrama de classes AML, onde suas crenças (deliveryBeliefs) e fragmentos de comportamento RequestQuotation e Pay and Bid podem ser identificados. O atuador da perna e o olho do observador também estão disponíveis. No entanto, a próxima função não pode ser representada na AML e as crenças, que são características estruturais, são representadas por classes específicas, que permitem ter operações.

O SellerAgent modelado no diagrama de classes AML é mostrado na Figura 67, onde os fragmentos de comportamento offerProduct, ReceivePayment e mountDeliveryProduct podem ser identificados. O atuador da perna e o olho do observador também estão disponíveis. Essa modelagem é apropriada para os Simple Reflex Agents.

O ProductionAgent modelado no diagrama de classes AML é mostrado na Figura 68. As crenças do agente são representadas por BeliefsProduction e seu objetivo é representado por satisfazerDemand. Esse agente tem olho como percebedor, pernas como atuadores e os seguintes fragmentos de comportamento: checkAvailableParts, requestNecessaryParts, checkParts, buildParts, computerAvailable, TestComputer, productComputers, nextFunctionProduction, goalFuncProduction e probFuncProduction. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um agente baseado em metas; portanto, a próxima função a função objetivo e a função problema de formulação deveriam ter sido representadas. Essas funções foram modeladoras como fragmentos de comportamento com o respectivo nome, mas não

há semântica disponível para diferenciá-las. Também não foi possível representar o planejamento desse agente.

O ManagerAgent modelado no diagrama de classes AML é mostrado na Figura 69. As crenças do agente são representadas pelo BeliefsManager, e os objetivos são representados pelo maximizeSale e MaximizeGain. O ManagerAgent possui os seguintes fragmentos de comportamento: requestBuyingPrice, requestSellingComputer, requestSellingPrice, estimativaBuyingPrice, valuSellingPrice, managerPlanning, nextFuncManager, formGoalManager, formProbManager e utilityFunctionManager. De acordo com o apresentado no estudo de caso, esse agente deve ser modelado como um Agente Baseado em Utilidades, de modo que a próxima função, função de objetivo de formulação e função de problema de formulação deve ser representada. Embora alguns fragmentos tenham sido definidos com nomes dessas funções, não há semântica disponível na AML para diferenciá-las. Também não foi possível representar o planejamento desse agente.

7.4. Discussão

Nas seções acima, usamos quatro linguagens de modelagem para modelar o TAC-SCM. Nesta seção, destacamos os recursos estruturais e comportamentais que podem e não podem ser modelados em cada linguagem de modelagem. Nossa intenção é demonstrar que apenas o MAS-ML 2.0 é capaz de modelar as arquiteturas de agentes internas descritas neste documento, porque é a única linguagem capaz de modelar todos os recursos estruturais e comportamentais cobertos por essas arquiteturas. A Tabela 3 mostra que todos os recursos estruturais exigidos pelas arquiteturas são modelados pelas quatro linguagens de modelagem. As características comportamentais da ação, plano e percepção são modeladas pela maioria delas. Isso indica que a arquitetura do agente interno Simple Reflex Agents e MAS-ML agentes só podem ser modelados usando o MAS-ML 2.0. pode ser modelado por AUML, AML e MAS-ML 2.0. Essas arquiteturas não podem ser modeladas pelo MAS-ML porque não são capazes de modelar a percepção. No entanto, apenas o MAS-ML 2.0 é capaz de modelar planejamento, próxima função, função de formulação de problemas, função de formulação de objetivos e função de utilidade. Assim, o agente interno arquitetou agentes reflexivos baseados em modelo, agentes baseados em objetivos e agentes baseados em utilidades

8. Conclusões e Trabalhos Futuros

Semelhante ao desenvolvimento orientado a objetos, a modelagem é uma atividade essencial para o desenvolvimento do MAS. Portanto, os agentes precisam ser modelados de forma consistente com sua implementação. De acordo com Russell e Norvig (2003), o ambiente pode ser classificado como determinístico ou estocástico, totalmente observável ou parcialmente observável, estático ou dinâmico, contínuo ou discreto, episódico ou seqüencial, e um ambiente específico.
a arquitetura do agente pode ser escolhida dependendo do tipo de ambiente que o sistema multi-agente executará. Russell e Norvig (2003) classificam os agentes em quatro tipos de arquiteturas internas: agentes reflexos simples, agentes reflexivos baseados em modelos, agentes baseados em objetivos e agentes baseados em utilidades.

Dependendo do tipo de tarefa que o agente executará, uma arquitetura pode ser mais apropriada que outra; por exemplo, agentes que têm metas conflitantes ou que podem não ser totalmente alcançadas devem ser implementados como agentes baseados em utilitários, para que possam usar a função de utilidade para auxiliar na tomada de decisões em relação ao processo de suas metas. Os agentes reflexos são adequados para situações que precisam de respostas rápidas a mudanças contínuas em seu ambiente. No entanto, em um sistema com vários agentes, geralmente há uma combinação de agentes, cada um com uma arquitetura diferente, como na declaração de uso fornecida neste artigo.

Existem várias linguagens de modelagem MAS, no entanto, nenhuma delas suporta totalmente a modelagem de diferentes arquiteturas de agentes propostas por Russell e Norvig (2003). Portanto, um agente reflex simples, agente reflex baseado em modelo, agente baseado em meta e agente baseado em utilidade devem ser modelados com características específicas que devem possuir internamente. Este artigo apresentou uma extensão do metamodelo TAO e da linguagem MAS-ML para permitir a modelagem de diversas arquiteturas internas de agentes publicadas na literatura de agentes.

O TAO e o MAS-ML foram originalmente projetados para apoiar a modelagem de agentes pró ativos baseados em metas com o plano. Assim, alguns problemas foram detectados ao tentar usar a linguagem para modelar agentes reativos e outras arquiteturas proativas. Nesse sentido, a evolução do TAO incorpora novos conceitos relacionados ao agente, e a evolução do MAS-ML proposta neste trabalho envolve a definição de duas novas metaclasses, AgentPerceptionFunction e AgentPlanningStrategy, a fim de agregar a representação de diferentes comportamentos do agente.

Além disso, novos estereótipos para descrever o comportamento de agentes de arquiteturas específicas foram definidos e associados à metaclasse AgentAction.

As estruturas estáticas das entidades AgentClass e AgentRoleClass também foram modificadas.

Em seguida, os diagramas de classe, organização, função, sequência e atividade foram alterados em consistência. Os diagramas estáticos foram alterados para modelar cinco tipos de agentes, os Agentes Baseados em Objetivos com plano foram mantidos e os Agentes Reflexos Simples, Agentes Reflexos Baseados em Modelo, Agentes Baseados em Objetivos e Agentes Baseados em Utilidades, cada um com os respectivos elementos. A representação inicial das funções de agente foi mantida e duas novas representações foram fornecidas, respectivamente, associadas a Agentes Reflexos Simples e Agentes Reflexos Baseados em Modelo. Os diagramas dinâmicos foram desenvolvidos com uma nova representação para cada tipo de agente.

Uma ferramenta de modelagem foi criada para suportar a nova versão da linguagem MAS-ML; foi nomeado ferramenta MAS-ML. É um domínio de ambiente específico para modelar o MAS, implementado como uma plataforma de plug-in do Eclipse. A ferramenta MAS-ML suporta a modelagem de diagramas estáticos e dinâmicos de acordo com o MAS-ML 2.0 e permite realizar a modelagem, validando os modelos criados e persistentes. As regras da OCL foram estabelecidas para permitir a verificação do modelo de diagramas criados com esta ferramenta e fornecer aos projetistas a possibilidade de analisar a formação dos poços de seus diagramas.

Finalmente, foi apresentado um estudo de caso com a modelagem de um MAS para o TAC-SCM. Agentes com diferentes arquiteturas internas foram modelados usando quatro linguagens de modelagem: MAS-ML 2.0, MAS-ML original, AUML e AML. As arquiteturas foram escolhidas de acordo com as atividades a serem executadas pelo agente. Como é demonstrado na Seção 7.3 e retomado na Seção 7.4, o AUML e a AML são capazes de modelar apenas duas arquiteturas internas do agente, e o MAS-ML original é capaz de modelar apenas uma arquitetura do agente. Portanto, eles não são adequados para modelar problemas como o TAC-SCM, onde é necessário modelar agentes mais complexos que devem, por exemplo, ser capazes de planejar, formular problemas e objetivos e avaliar suas utilidades. A única linguagem de modelagem capaz de modelar essas características é o MAS-ML 2.0.

Como trabalhos futuros, outros estudos de caso estão sendo conduzidos para fornecer validação adicional para este trabalho, e novos estudos experimentais serão realizados para avaliar a técnica proposta. Além disso, a possibilidade de extensões TAO e MAS-ML para outras arquiteturas internas é um trabalho possível. Em relação à ferramenta MAS-ML, algumas melhorias podem ser feitas na representação gráfica dos construtores para representar com mais fidelidade a representação proposta no metamodelo MAS-ML. A geração de código para uma estrutura é uma possibilidade interessante.