3
Plataforma MASP

Conforme descrito no item 1.1.1, a pesquisa planejada exige a realização de diversos experimentos e consequente coleta de dados para análise posterior. Esta necessidade causou a definição da primeira questão secundária de pesquisa:

QS1: *Obter uma ferramenta de suporte que permita a rápida instanciação dos experimentos para a pesquisa em curso, na qual a modelagem da simulação possa ser realizada através de arquivos de configuração.*

Para atender os experimentos a serem realizados, uma capacidade indispensável é a possibilidade de instanciar agentes que possuam controle de seu avanço no tempo de simulação, o que implica em um ambiente *multi–thread*. Esta capacidade é necessária para que os agentes possuam autonomia para atualizar seu estado no sistema de acordo com sua programação, independentes, portanto, de um mecanismo externo da *engine* de simulação.

A fim de atender a realização destes experimentos, foram verificados os principais *framework* e *toolkit* de simulação orientados a agentes, notadamente o Mason (MASON, 2013), o SWARM (SWARM, 2013) e o NETLOGO (Wilensky, 2013).

Um conjunto de requisitos de alto nível foi definido para a ferramenta que apoiaria à pesquisa:

RANS01 - Representação de atores da simulação em *thread* independentes, que fossem capazes de controlar seu avanço do tempo de simulação de forma individual: A implementação dos agentes de forma individual, com o controle não somente de seu estado, mas de sua dinâmica na simulação, é necessária para os testes futuros das soluções do problema.

RANS02 - Possibilidade de alterar rapidamente a descrição do cenário e o comportamento dos agentes, sem necessidade de alterar código: Admitia-se que seriam necessárias várias execuções de experimentos com variações entre eles para permitir análise dos resultados. A adoção de uma ferramenta que permitisse configurar o modelo de execução sem alterar o código atenderia dois

objetivos: inicialmente permitiria a rápida instanciação dos experimentos, e em segundo lugar, evitaria a inserção de código que poderia afetar os resultados, ou seja, garantiria que os modelos se execução sofreriam o mesmo tratamento.

RANS03 - Uso de Linguagem Java: Como já apresentado em no item 1.1, o uso de linguagem Java foi admitido como restrição da pesquisa por representar uma instancia mais complexa do problema.

RANS04 - Acesso ao código fonte da ferramenta: Este acesso é necessário para assegurar-se que os agentes realmente estão com o controle de suas linhas de execução, e sob a forma em que é feito o agendamento das ações a serem executadas.

RANS05 - Possibilidade de executar número expressivo de agentes simultaneamente: O framework deveria ser capaz de instanciar milhares de agentes simultaneamente, para permitir a avaliação da abordagem.

RANS06 - Capacidade de coleta de informações em tempo de execução para análise: Possibilidade de realizar registro (i.e *Log*) dos eventos em um formato que permitisse análise estatística dos resultados.

RANS07 - Possibilidade de alterar o código fonte: considerada uma necessidade futura durante a execução da pesquisa, para testar as abordagens.

Após uma análise das ferramentas citadas, verificou-se que elas não atenderiam a representação de atores da simulação em *thread* independentes, que fossem capazes de controlar seu avanço do tempo de simulação de forma individual. Isto porque, nestas ferramentas, não importando se o avanço de tempo de simulação adotado pela ferramenta usa a abordagem *next-event* ou *time–stepped*, o fato é que o avanço de cada agente é controlado totalmente pela plataforma através de chamadas de métodos. Os agentes de simulação são implementados como objetos e não alteram seu estado autonomamente. A *engine* de simulação mantém referência explícita a todos os objetos que representam atores.

Ao implementar os agentes como objetos aos quais se mantém referencia explicita na *engine*, que se torna responsável por determinar a atualização destes através de mensagens do paradigma de Orientação a Objetos, ie. chamada de método, elimina-se a autonomia que um agente deve possuir para gerir seu estado no sistema: o estado é gerido por elemento exógeno. Contudo, para que o agente atualize seu estado, é necessário que ele conheça o avanço do tempo de simulação,

uma vez que o estado é função do avanço deste. A adoção de uma abordagem na qual os agentes controlem seu estado é coerente com o paradigma de Orientação a Agentes, na qual o agente tem autonomia no sistema, e mais ainda, é uma solução de engenharia de software que permite o desenvolvimento de simuladores nos quais os agentes possuem menor acoplamento à *engine*, ampliando possibilidades de evolução e reuso.

Descartada a possibilidade de utilizar suporte de ferramentas de simulação existentes na condução dos experimentos desta pesquisa, decidiu-se por desenvolver uma ferramenta própria, específica para realizar os experimentos e coletar dados, mas que pudesse apoiar também o desenvolvimento de MABS. A solução recebeu o nome genérico de Multi Agent Simulation Platform (MASP).

Durante a concepção da arquitetura, especial atenção foi dada a possibilidade de alterar rapidamente a descrição do cenário e o comportamento dos agentes, sem necessidade de alterar código (RANS02). Como consequência, admitiu-se a seguinte hipótese, também definida em 1.1.1:

H1: *O uso de ontologias, associada a técnicas de inferência lógica e de injeção de dependências, permitiria a rápida construção e alteração de modelos de execução.*

O processo de desenvolvimento foi iterativo, e o entendimento sobre qual arquitetura atenderia as necessidades evoluiu ao longo de todo trabalho. Este capítulo apresenta a arquitetura prescritiva, ou seja, a arquitetura conceitual que foi implementada (Taylor et al., 2009).

As próximas seções descrevem, sequencialmente, a abordagem, a arquitetura e sua implementação

3.1 Abordagem

No projeto de modelos de execução, ou seja, de programas destinados a executar simulações, parte considerável do código destina-se a representar conceitos do sistema que está sendo simulado e suas relações. Especificamente em simulações baseadas em agentes, isto significa codificar os atores da simulação, seus comportamentos, os elementos passivos do sistema e como estes elementos interagem entre si.

Uma vez que o MASP foi criado para instanciar diversas simulações com modificações entre si, buscou-se uma forma de evitar a codificação e compilação sucessiva dos construtos. Isto para evitar um esforço paralelo e não direcionado ao problema principal, relacionado ao controle de atrasos em tempo de execução e para mitigar o risco de inserir erros nos modelos de execução que pudessem afetar a análise de resultados. A solução concebida buscou, então, reduzir ao máximo o

acoplamento entre o motor da simulação e os componentes associados ao sistema simulado.

Resumidamente, o motor de simulação, no momento da carga da simulação, lê um modelo conceitual que descreve o sistema a ser simulado, com seus atores, comportamentos e relacionamentos, e com estes dados passa a instanciar os agentes e suas dependências. A técnica adotada, portanto é Injeção de Dependências (Prasanna, 2009). Esta abordagem oferece ainda outra vantagem, que é a possibilidade de definir-se um modelo de domínio que represente conceitos básicos de uma família de simulações, e que poderá ser estendido ou modificado para casos particulares.

Adicionalmente, uma série de diretivas específicas para controle das simulações, tais como tempo do passo, duração da simulação, abordagem de controle, e outros, também são necessários para a execução. Uma vez que estas informações não possuem um grafo de relacionamentos, como no modelo conceitual da simulação, e conceitualmente não se referem a mesma tarefa, decidiu-se por usar uma representação mais simples: um arquivo textual de propriedades.

Para que o motor de simulação possa carregar o modelo, módulos de software que representem os conceitos em tempo de execução devem ser codificados. O motor terá então a tarefa de ler o modelo conceitual, identificar os módulos de software que correspondem aos conceitos representados, carregá-los em memória e realizar a injeção das dependências que também são extraídas do modelo. Uma visão de alto nível da proposta é apresentada na figura 3.1.

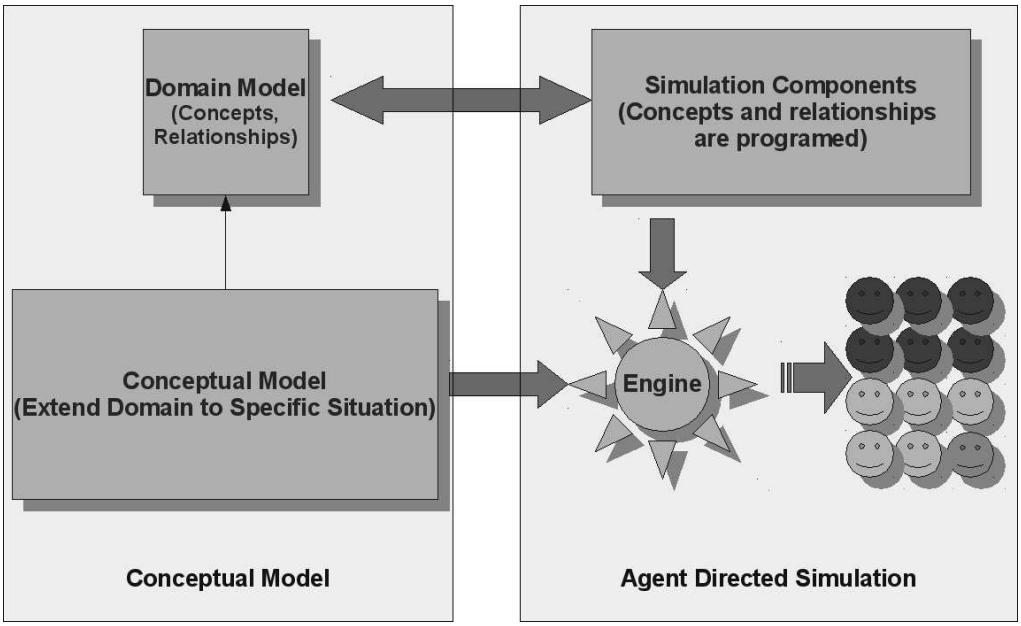

Figura 3.1: Arquitetura em alto nível de abstração

3.2 Representando o Modelo Conceitual em Ontologia

Os modelos conceituais, em simulação, são projetados para representar os conhecimentos estáticos e dinâmicos de um sistema. Estes modelos podem não capturar informação quantitativa, porém sua expressividade deve ser tal que permita aos engenheiros de software compreender o sistema a ser simulado. Redes semânticas, modelos orientados a objetos e modelos de dados são alguns exemplos de modelos conceituais usados no campo da simulação para modelar um sistema (Fishwick, 1995).

A abordagem proposta nesta pesquisa foi realizar a representação do modelo conceitual utilizando ontologias. O uso de ontologias para representar modelos conceituais de simulações foi apresentado em (Benjamin et al., 2006; Okuyama et al., 2006; Silver et al., 2007), e discutido no item 2.4.2.

Ontologias podem ser representadas em diferentes linguagens, porém a Ontology Web Language (OWL) (W3C OWL Working Group, 2009), que é uma extensão da RDF, vem se consolidando como um padrão e possui características como a possibilidade de representar Lógica Descritiva (DL) (Casanova et al., 2007) em seu subconjunto, a OWL-DL. Estas sentenças podem ser processadas automaticamente por computador, aonde se aproveita o suporte de *parser* existente para XML e suas extensões.

A existência de API que permitem realizar inferências sobre as ontologias descritas em OWL-DL em tempo de execução também é um fator importante na avaliação sobre seu uso para representar modelos conceituais de simulação. Motores de Inferência como a biblioteca Jena (JENA, 2007) ou Pellet (Pellet, 2009) podem ser utilizados como componentes em uma arquitetura desenvolvida para apoiar desenvolvimento de simulações. Desta forma o programa poderá ler, interpretar e modificar a simulação em tempo de execução.

Ao representar o modelo conceitual de uma simulação em uma ontologia, especificamente em OWL-DL, obtém-se melhorias expressivas, tais como:

- desenvolvimento cooperativo do modelo, através de técnicas de edição colaborativa de ontologias;
- ferramental para edição, fusão e alinhamento de ontologias, como o existente em (Protege, 2013);
- reuso de modelos existentes, seja pela especialização de modelos mais gerais, seja pela sua modificação. Por exemplo, o modelo de engenharia de tráfego pode utilizar em sua ontologia uma taxonomia sobre veículos terrestres.
- A ontologia pode ser utilizada como uma linguagem comum entre os engenheiros de software e os especialistas no domínio.

– Uma vez que os conceitos da ontologia possuam representação em construtos de software, a engine poderá instanciá-los sem modificação de código, utilizando técnicas de injeção de dependência. Isto permitirá que a simulação seja alterada sem intervenção de engenheiro de software[1] .

A abordagem proposta usa uma ontologia para representar o sistema a ser simulado. Uma vez que o trabalho é direcionado a Orientação a Agentes, esta modelagem considera os conceitos inerentes ao paradigma. A abordagem prevê o desenvolvimento incremental de modelos, no qual uma ontologia inicial é fornecida aos desenvolvedores, esta ontologia contendo os conceitos de modelagem orientada a agentes com um conjunto de relacionamentos definidos. Este conjunto inicial não deve ser alterado, pois possui relação à forma com que a injeção de dependências ocorrerá. Esta ontologia inicial deverá ser estendida para representar os conceitos específicos dos sistemas simulados. Ontologias mais genéricas podem ser utilizadas como base para vários modelos, bem como a união de ontologias é possível através de técnicas específicas de *merge* de ontologias (Choi et al., 2006).

Uma limitação resultante do uso da ontologia é a falta de expressividade para tratar questões ou relações temporais, resultante do vocabulário da lógica utilizada. Ressalta-se que os únicos modelos conceituais que possuem representação do templo são baseados em funções matemáticas, tais como sistemas de equações diferenciais. Uma vez que a formulação matemática não se mostra adequada a modelagem de sistemas multiagentes nos quais se deseje modelar o nível de granularidade fina para observar o comportamento global do sistema, estes modelos não são adequados ao problema. O MASP foi desenvolvido aceitando esta restrição sobre os modelos conceituais que seriam criados.

A ontologia inicial da arquitetura proposta foi criada utilizando a ferramenta Protégé-OWL Editor (Protege, 2013), e uma representação gráfica desta, obtida através do *plug-in* Ontoviz, é apresentada na figura 3.2.

Esta ontologia deverá ser estendida para representar o modelo conceitual de simulações particulares, e a mesma é uma evolução da ontologia de domínio apresentada no trabalho Dominium (Taranti & Choren, 2007a). Os conceitos representados incluem: organizações sociais; conceitos espaciais; estruturas normativas; atores; papéis e comportamentos, conforme o modelo AGR (Ferber et al., 2003).

Os principais conceitos incluídos na ontologia são:

– *Agent* – define um ator da simulação. Para cada agente descrito na ontologia, a *engine* de simulação instanciará agentes de simulação. A *engine* permite instanciar múltiplos agentes de simulação para cada instância de agente descrita na ontologia. Este número de agentes a ser instanciado é informado

[1] Considerando-se apenas alteração de dependências dos agentes de simulação

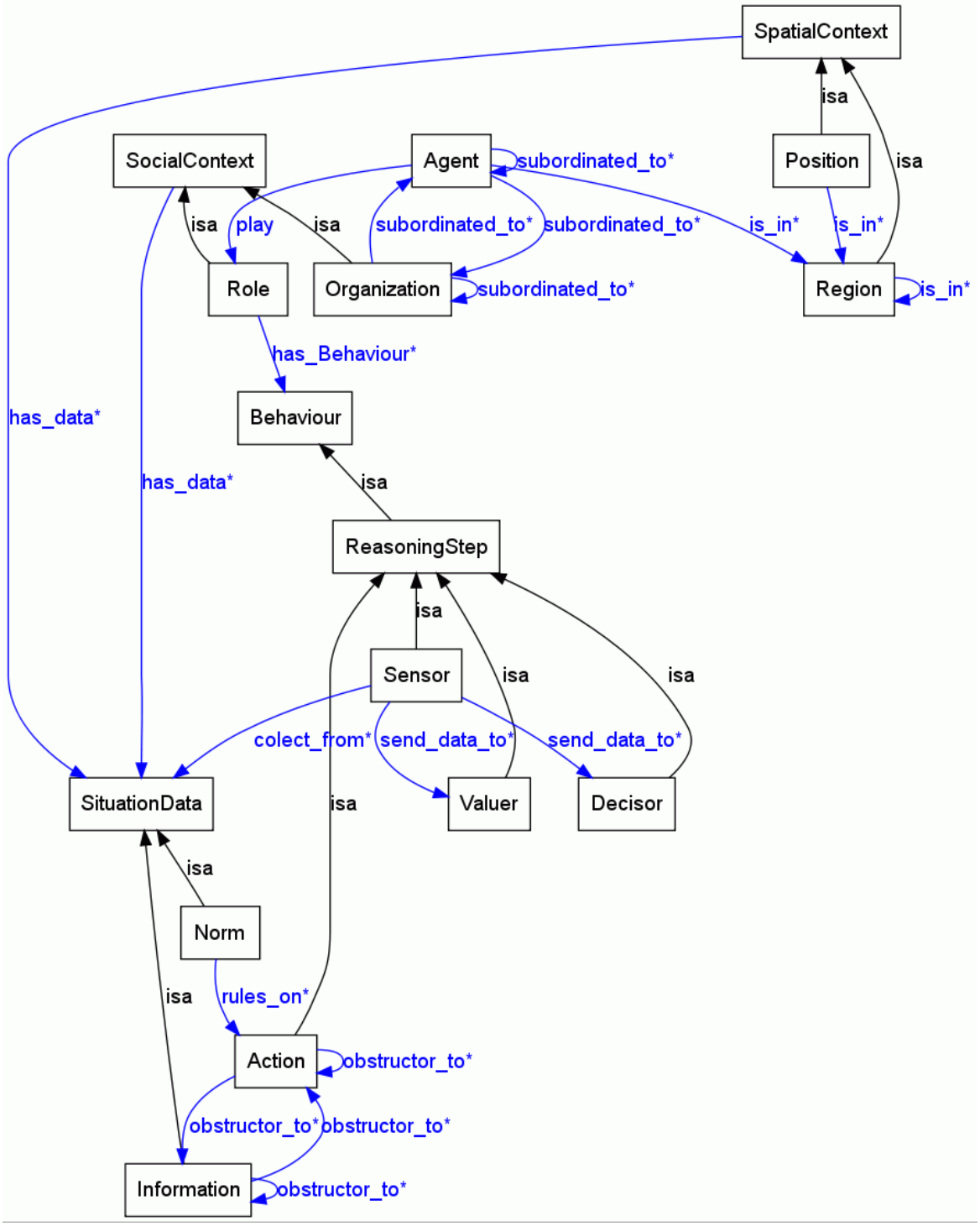

Figura 3.2: Ontologia do modelo conceitual

pelo desenvolvedor na própria ontologia e é um recurso útil para realização de testes de carga no simulador.

– *Behavior* – descreve conceitos relacionados as ações executadas por um ator no sistema simulado. Suas subclasses são associadas ao paradigma de computação autonômica - ou seja: monitorar, analisar, planejar e executar [2].

– *ReasoningStep* – é uma superclasse que descreve um padrão de ciclo de processo de decisão;

– *Action* – classe da ontologia que representa ações que os atores executam e que podem afetar o contexto de outros atores ou o ambiente. Estas ações podem estar submetidas a normas ou possuir interferência com outras ações.

– *Sensor* –representa comportamentos associados a coleta de informações oriundas do contexto do agente;

– *Valuer* – representam os comportamentos que recebem informações dos sensores e lhes atribuem ordem de prioridade ou valor relativo. Os resultados de sua computação, na representação de código, aciona um comportamento de decisão.

– *Decisor* – representa comportamentos que realizam a tomada de decisão propriamente dita, ou seja, sua representação em código conterá os algoritmos que indicarão as ações externas a serem executadas pelos agentes.;

– *SpatialContext* – é a superclasse de representações de contextos espaciais no sistema simulado. A ontologia prevê subclasses que apoiam tanto a representação em espaço vetorial quanto em espaço discreto (ex. células). A estrutura da ontologia também permite associar propriedades a contexto espacial, tais como informações ou normas que vigorem nestes contextos somente;

– *Region* – define uma área na qual agentes possam ser localizados ou moverem-se;

– *GeographicObject* – define um recurso passivo do ambiente. Útil para descrever obstáculos ou elementos que não sejam agentes, mas que influenciem o processo de decisão destes. É subclasse de contexto espacial (não está representado na figura 3.2 devido à simplificação da imagem);

– *SocialContext* – é a superclasse para as representações de contexto social. O contexto social se relaciona com os papéis que um agente desempenha no sistema, suas relações com grupos a quais pertença ou possua relação de subordinação;

[2] Ciclos semelhantes podem ser observados na literatura que descreve o processo de decisão, tais como o ciclo OODA utilizado pelos militares ou o ciclo cognitivo, descrito na interação homem-máquina

- *Organization* – definem as organizações de agentes e suas subordinações;
- *Role* – é um papel que o agente desempenha na simulação. O uso de restrições em lógica descritiva permite definir comportamentos específicos para papéis, e a transitividade destas restrições auxilia a modelagem destes comportamentos para as subclasses de um papel;
- *SituationData* – é a superclasse de todas as classes que representam a situação de um agente;
- *Information* –são informações que os agentes podem utilizar em seu processo de tomada de decisão;
- *Norm* – Definem normas que regulam o comportamento dos agentes da simulação. As normas podem ser associadas a contextos espaciais ou sociais. Não foi contemplado na ontologia, porém em simulações que utilizem estrutura normativa, este conceito pode ser estendido seguindo a lógica deôntica (von Wright, 1951), com a consequente criação de subclasses de Proibição, Permissão e Obrigação. Normas afetam as ações dos agentes.

Os relacionamentos que são expressos na ontologia têm papel especial no reconhecimento e carga do modelo pela engine de simulações. Como já citado, estes relacionamentos entre as classes não devem ser alterados para uso da solução proposta, pois as consultas realizadas sobre a ontologia a fim de verificar as dependências existentes se apoiam nestas propriedades e em propriedades de dados.

A ontologia apresentada nesta seção representa o estagio final do processo evolutivo inerente a pesquisa. Os conceitos inseridos na ontologia e seus relacionamentos buscam oferecer abstrações para a representação completa se um sistema, com exceção da representação temporal, cuja limitação já foi discutida. Desta forma se representam os contextos social e espacial, com inspiração nos trabalhos (Ferber et al., 2004) e (Weyns et al., 2007), que discutem cada um destes contextos, respectivamente. Esta ontologia foi construída com apoio da plataforma Protégé na versão 3.4, utilizando a linguagem OWL-DL, e sua consistência foi frequentemente verificada com o *reasoner* Pellet 1.5 (Pellet, 2009).

3.3 Arquitetura de Implementação

A proposta de alto nível (fig. 3.1) foi detalhada na arquitetura de componentes apresentada na figura 3.3. A arquitetura priorizou o baixo acoplamento entre componentes e o uso de injeção de dependências para permitir o desenvolvimento de simulações com o mínimo de intervenção em código.

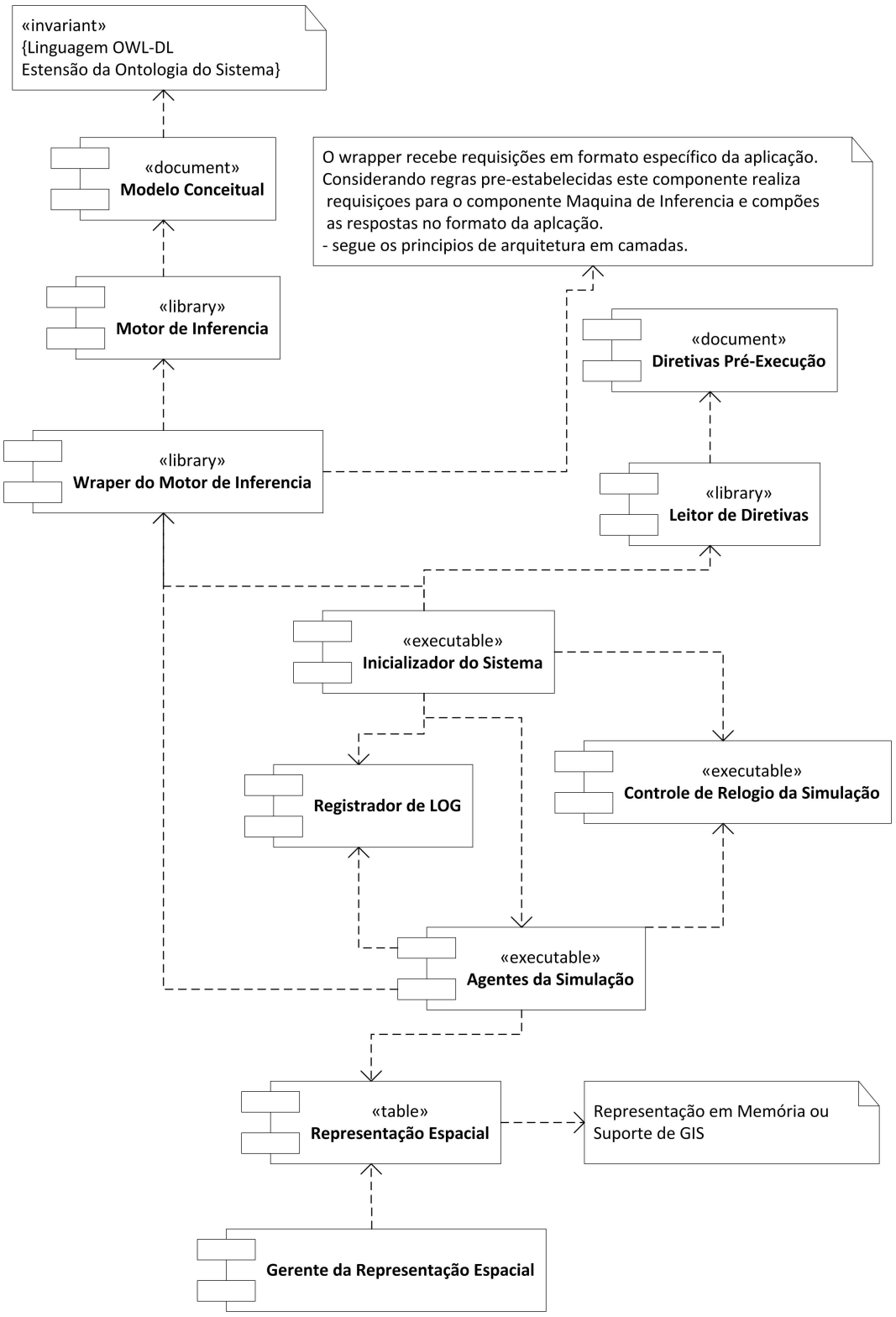

Figura 3.3: Arquitetura do MASP - Diagrama de componentes

Durante o desenvolvimento evolutivo percebeu-se que seria uma boa solução de engenharia não incluir na ontologia diretivas sobre como a simulação deveria ser executada, pois estas não compõe a modelagem do domínio do sistema que estaria sendo simulado. Esta decisão visou manter a integridade conceitual da arquitetura.

A seguir, os componentes da arquitetura são descritos, incluindo alguns detalhes de implementação e como estes componentes se comunicam entre si.

3.3.1 Inicializador do Sistema

Elemento responsável pela carga da simulação. O Inicializador é responsável por instanciar os elementos ativos da simulação, sejam: o Controle do Relógio da Simulação, o Gerente da Representação Espacial e os Agentes de Simulação, que representam os atores do sistema simulado. Estes elementos são implementados como agentes de software.

Para cumprir sua tarefa, o Inicializador verifica as Diretivas Pré-Execução, que são registradas em uma lista de variáveis e valores. Esta lista permite ao pesquisador alterar dados que não sejam relativos ao modelo conceitual do sistema de forma não-invasiva, ie. sem alteração de código. Um exemplo de diretiva que altera o comportamento do componente é a realização de ciclo de aquecimento, que é uma corrida curta da simulação realizada a fim de verificar uma taxa de execução inicial que seja adequada ao ambiente de execução, ou seja, que permita executar dentro dos limites máximos de tardiness estabelecidos. Estes limites também constam nas diretivas pré execução.

Adicionalmente, o componente Inicializador do Sistema utiliza os serviços do componente Wrapper do Motor de Inferência para obter a lista de todos agentes de simulação que serão instanciados. A instanciação destes ocorre por técnica de reflexão, utilizando também a informação dos papéis que serão desempenhados por estes agentes na simulação. É necessário manter uma relação entre o nome das classes da ontologia e as classes de código que implementam os conceitos, a fim de permitir o correto funcionamento do MASP.

3.3.2 *Wrapper* do Motor de Inferência

O MASP depende do uso de componente de prateleira (COTS) para realizar a inferência sobre o modelo conceitual, que é o Motor de Inferência. Para permitir a evolução deste componente no futuro, optou-se por encapsulá-lo em um *Wrapper* usando o padrão *façade*, que disponibiliza interfaces para que os demais componentes possam consultar a ontologia que mantém o modelo conceitual sem a necessi-

dade de tratar detalhes, tais como as consultas sequenciais sobre triplas RDF devido aos relacionamentos entre conceitos.

3.3.3 Motor de Inferência

A complexidade relativa à realização de inferências lógicas não é objetivo desta pesquisa, porém seu uso é uma necessidade para permitir a representação de modelo conceitual da simulação em uma ontologia OWL-DL. Para suprir esta necessidade verificou-se quais projetos se encontravam com um nível de maturidade que permitisse o seu uso. O objetivo era evitar que a pesquisa descrita nesta tese pudesse ser afetada por instabilidade ou erro do motor de inferência. A decisão foi utilizar Biblioteca JENA, atualmente mantida pela Apache Software Foundation.

3.3.4 Modelo Conceitual

O modelo conceitual representa a modelagem da simulação sob forma estática. A arquitetura prevê a representação sob ontologia escrita em OWL-DL, conforme descrito no item 3.2.

A instanciação da arquitetura oferece uma ontologia que deverá ser estendida para as simulações em particular.

3.3.5 Leitor de Diretivas

É um conector que oferece uma interface de acesso as Diretivas de Pré-Execução, que permite realizar consultas sobre as mesmas através de chamadas de métodos.

3.3.6 Diretivas Pré-Execução

Conjunto de chaves/valor que é editado pelo usuário em arquivos de texto simples, tais como editores como o notepad ou vi. Seu acesso pelos demais componentes é realizado através do conector Leitor de Diretivas, conforme já descrito.

Abaixo são apresentados os principais parâmetros que um usuário pode alterar nas simulações. Enfatiza-se que a seleção não visava apenas atender execução de simulações, mas principalmente possibilitar a realização dos experimentos nas etapas posteriores desta pesquisa.

– *Porta lógica usada para comunicação* – O MASP utiliza comunicação por RMI. Para permitir várias instâncias de simulação na mesma máquina, o usuário deverá selecionar uma porta lógica para cada execução.

- *Nome da ontologia* – identifica a ontologia que contém o modelo conceitual.
- *Taxa inicial da simulação* – valor inicial da relação entre o Tempo de Simulações e o Tempo Real.
- Modelo de Avanço do Tempo de Simulação – define se os agentes de simulação realizarão avanço do seus tempos de simulação por passos de tempo ou ao próximo evento (sempre em abordagem *paced-time*, i.e *scaled–time*)
- *Passo de tempo padrão* – para uso em simulações que operem com avanço de passo
- *Duração da simulação* – determina a duração do experimento.
- *Tempo do Ciclo de Aquecimento* – caso algum valor seja informado, a simulação executa durante o período estipulado para buscar uma taxa inicial de execução a ser utilizada no ciclo principal.
- *Ativar Registro de LOG* – Permite habilitar ou não o registro de log da simulação. O conjunto de informações registrada objetiva medições de tardiness.
- *debug* – Permite habilitar ou não o envio para console de uma série de informações para manutenção do MASP.
- *Controle Automático de* Tardiness – ativa os algoritmos de controle de *tardiness*, que são objeto principal desta pesquisa. A razão para habilitar ou não estes algoritmos é permitir a realização de comparações. Este controle permite desacelerar ou acelerar automaticamente o avanço do tempo da simulação em relação ao tempo real.
- *Parada Automática* – habilita a parada da simulação caso o limite máximo de *tardiness* seja atingido, encerrando a simulação por falha.
- *Abordagem de Controle de Tempo* – permite selecionar qual algoritmo irá gerenciar o avanço do tempo da simulação. Esta funcionalidade foi adicionada para permitir o aprimoramento do algoritmo e a comparação entre suas várias versões.
- *Conjunto de Variáveis para Ajuste do Controle de Tardiness* – Diversos parâmetros que influenciam o controle do avanço do tempo são diretamente relacionados ao sistema a ser simulado. Um conjunto de variáveis permite o ajuste destes parâmetros, dentre as quais:
  - *Nível de Tardiness Máximo* – quando a *tardiness* a o sistema deve encerrar a simulação por erro, uma vez que implicaria em estados inconsistentes com o sistema real.

- *Nível de Erro Espacial Máximo* – Registra o maior erro espacial permitido em uma simulação. Esses erros não controlados pelo MASP, e possuem relação com o tardiness do sistema e com o passo de execução.
- *Gatilho para ativar correção* – indica um nível de *tardiness* para o qual os algoritmos de correção devem ser ativados para evitar possíveis estados inconsistentes da simulação.
- *Gatilho para ativar correção agressiva* – indica um nível de *tardiness* para o qual correções mais fortes devem ser ativadas.
- *Parâmetros de Controle*– substituem constantes no algoritmo de controle do avanço do tempo de simulação. Seu ajuste permite estabilizar a simulação, evitando oscilações bruscas no sistema. É possível atribuir valores diferentes para aceleração ou desaceleração do sistema.
- *Número de Ciclos Estáveis para Acelerar*– Registra um número mínimo de ciclos de controle que devem ocorrer sem tendência de crescimento de tardiness para que o mecanismo de aceleração seja acionado.
- *Número de Ciclos de Controle para Coleta de Dados*– O MASP realiza coleta de dados para percepção de tendências (i.e bias ou viés). Este parâmetro define quantos *n* últimos ciclos devem ter seus dados de tardiness registrados em memória para uso no algoritmo de controle.
- *Período para Execução do Ciclo de Controle de tardiness* – O controle de *tardiness* é realizado em frequência pré-estabelecida nas diretivas pre-execução. Existe uma relação de custo-beneficio envolvida: períodos pequenos consomem muitos recursos de computação e períodos grandes dificultam a redução de oscilações na taxa existente entre o tempo de simulação e o tempo real.

3.3.7 Registrador de LOG

As simulações que utilizam o MASP são *multi-thread*, pois cada agente é uma *thread* individual. O registro de informações destes agentes em um único local exige controle de concorrência, tarefa executada pelo componente Registrador de LOG.

Os registros são realizados em arquivos do sistema, utilizando o padrão csv, uma tabela em texto estruturado sem formatação. Estes dados são posteriormente acessados com ferramentas de análise estatística.

3.3.8 Representação Espacial

Componente que mantém os dados de representação espacial. O MASP possui suporte a representação em duas dimensões, podendo-se optar por representação em memória ou em sistema GIS.

3.3.9
Gerente da Representação Espacial

É um agente de software que atualiza o posicionamento dos elementos com representação espacial. Para permitir esta atualização, os agentes que possuem representação espacial periodicamente informam sua posição, rumo e velocidade. O Agente Gerente da Representação Espacial utiliza estes dados para estimar a posição corrente, e corrige sua estimativa a cada informação recebida.

3.3.10
Agentes da Simulação

Estes agentes representam, em tempo de execução, os atores do sistema simulado. Para sua implementação devem ser utilizados os modelos oferecidos pelo MASP.

3.3.11
Templates de Agentes de Simulação e Comportamentos

Estes componentes não são apresentados na figura 3.3. São bibliotecas oferecidas aos desenvolvedores com classes que devem ser estendidas para implementar Agentes de Simulação e seus Comportamentos. Seu uso é essencial, uma vez que o mecanismo de Injeção de Dependência implementado utiliza reflexão, e está implementado nos modelos oferecidos.

3.3.12
Controle do Relógio da Simulação

Este componente de software é responsável pelo avanço do tempo de simulação. Ele possui o contador que representa o relógio de simulação, cujo avanço é realizado em função do avanço do tempo real. Neste componente é que se encontram implementados os algorítimos que controlam o tardiness do sistema em tempo de simulação, foco principal da pesquisa realizada.

3.4
Tecnologias utilizadas

Ao longo da pesquisa algumas tecnologias foram substituídas, e novas versões adotadas. A lista abaixo representa o conjunto das principais tecnologias utilizadas no MASP:

- OpenJava 7 (SDK e JRE) – bibliotecas de desenvolvimento e máquina virtual de execução. O uso de tecnologia Java é uma restrição ao problema.
- JADE 4.2.0 (Java Agent DEvelopment Framework) – *Midleware* de desenvolvimento de sistemas multiagentes em Java, que implementa a especificação

FIPA (Bellifemine et al., 2005). O MASP reescreve as classes *Agent* e *Behaviour* para adequá-las a uso em simulações, alterando, por exemplo, a referencia temporal para o relógio de simulação em vez de realizar uma *system call* ao sistema operacional. O mecanismo de agendamento de comportamentos do JADE é composto principalmente pelas classes *TickerBehaviour* e *WakerBehaviour*. A classe *TickerBehaviour* implementa um mecanismo que permite executar um comportamento de forma periódica. A classe *WakerBehaviour*, por sua vez,implementa um agendador que executa um comportamento uma única vez em um instante no futuro. Estes comportamentos são programados seguindo uma política de agendamento de tarefas não–preemtivo no qual cada agente possui o controle de sua *thread* e coordena a execução de seus comportamentos. O MASP reescreve as classes de agendamento de comportamento e de agentes para adaptá-las as RANS e ao uso de tempo de simulação ao invés de tempo real.

– Biblioteca JTS (Java Topology Suite) – implementação em Java da conhecida libgeos, que oferece recursos para cálculos geográficos.
– PostgreSQL 9.1 /Postgis 2 – O Postgis é uma extensão do PostgreSQL que acrescenta recursos para sistemas de informações geográficas ao SGBD.
– JENA 2.7.2 – Biblioteca para manipulação e consulta à ontologias, incluindo o suporte a OWL.

3.5 Exemplo de Uso

Esta seção apresenta um exemplo de uso do MASP, no que tange ao modelo conceitual e instanciação de simulações. O exemplo é uma simulação hipotética de Jogo de Guerra que utiliza conceitos de Guerra Anti-Submarino.

A ontologia fornecida pelo MASP foi estendida para criar o modelo conceitual da simulação, representando o domínio do sistema simulado e seu estado inicial. A figura 3.4 apresenta a ontologia resultante, simplificada para possibilitar a leitura.

O modelo conceitual foi criado por um especialista no domínio (de guerra anti-submarino) com auxílio de um engenheiro de software. Foi utilizada a plataforma Protégé para editar a ontologia e o *reasoner* Pellet para verificar sua consistência.

No exemplo, o especialista criou uma taxonomia de papéis sobre o conceito de unidade militar (Unit). O conceito *Role* possui uma relação explícita com *Behaviour* – a relação hasBehavior.

Restrições são criadas, tais como: “has_Behavoiur has Kinematics” para Role, a qual por transitividade é transferida para todas subclasses e suas instâncias.

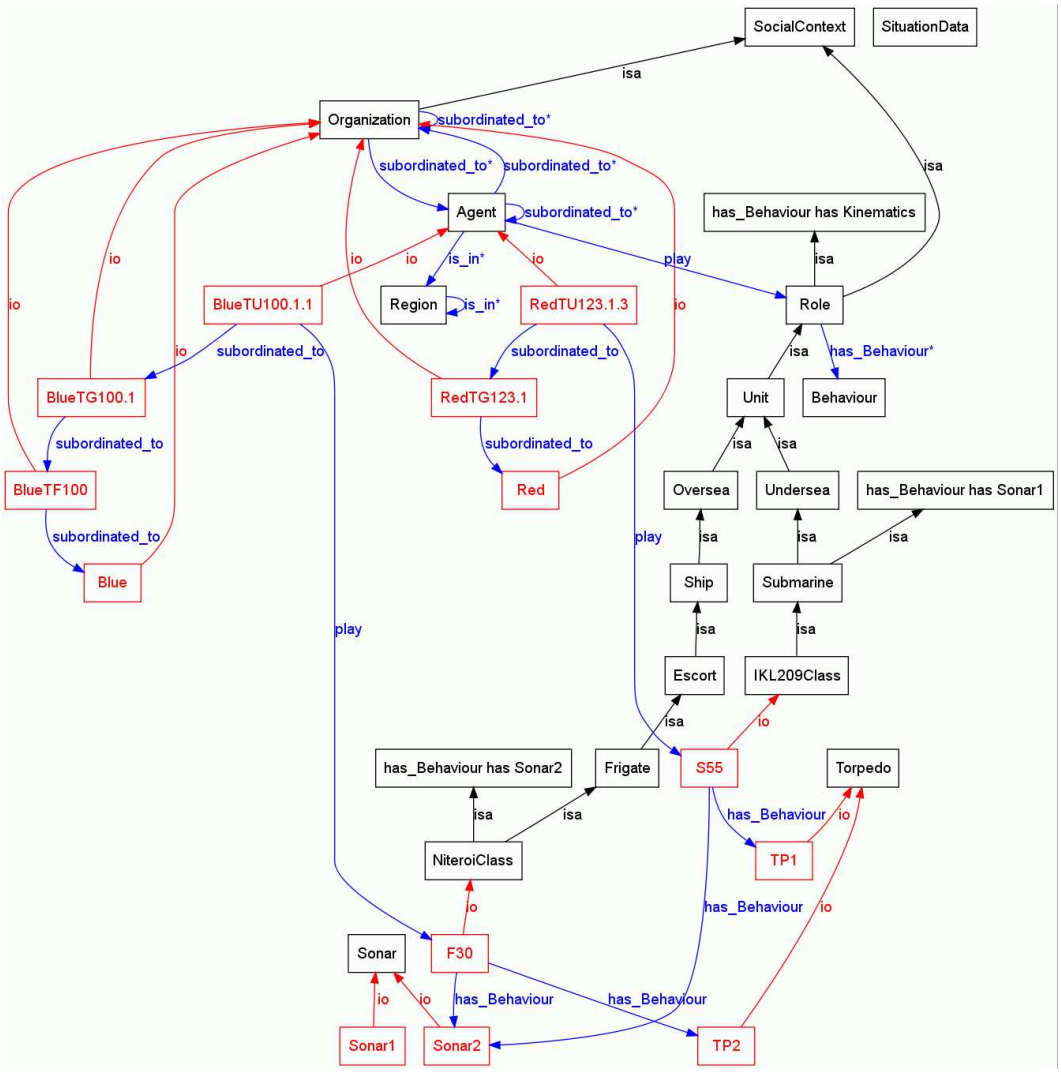

Figura 3.4: Ontologia: modelo Conceitual (parte de) de Guerra Anti-Submarina

A ontologia define que *Task Unit 100.1.1*, que é uma instância da classe Agent, também é subordinada ao *Task Group 100.1*, que por sua vez é uma instância de SocialContext, e pode possuir normas e informações associadas ao mesmo.

A Task Unit 100.1.1 executa o papel *Frigate 30*, que é uma instância de *Niteroi Class*. A instância Frigate30 tem uma relação do tipo *hasBehaviour* com *TorpedoType2* e *Sonar1*.

A instância de Role, Frigate30, também possui o comportamento *kinematics*, pois sua superclasse *Unit* possui uma restrição de existência sobre este conceito de comportamento.

Depois de estender a ontologia, esta pode ser modificada para acomodar *data properties*, associadas a indivíduos específicos da ontologia. Estas informações podem ser capturadas pela engine de simulação, e utilizadas para iniciar variáveis dos objetos instanciados pela engine na inicialização da simulação. Exemplificando, a data property *Armor* é associada a todas instâncias da superclasse Unit. Outras

propriedades de dados como *Damage*, *Range*, *HitProbability* também foram criadas no exemplo.

Uma vez que a ontologia esteja pronta, representando o modelo conceitual, o próximo passo é a implementação das classes em código Java que representarão as classes da ontologia. Para cada Papel ou Comportamento definido na ontologia, uma classe deverá ser criada, estendendo os modelos fornecidos pelo MASP. Durante a carga do programa, o componente de inicialização verificará quais papéis os agentes executam e os instanciará. Os agentes, por sua vez, verificam quais comportamentos lhe são associados e criam as respectivas instâncias, incluindo a inicialização de variáveis com base nas *data properties*.

A inicialização de uma simulação com muitos agentes (ex. 2.000) demora alguns segundos. Para não afetar a execução da simulação, o MASP aguarda que todo sistema esteja em memória antes de iniciar o avanço do tempo de simulação.

Alterações posteriores na ontologia serão refletidas em tempo de execução, a cada nova iniciação do sistema. Isto inclui a criação de novos agentes desempenhando os papéis existentes, modificação nos papeis, alteração nas propriedades de unidades em função da criação de restrições a propriedades, entre outras possibilidades.

O mecanismo para que as alterações sejam percebidas e instanciadas pela simulação é baseado em injeção de dependência por *setter injection* (Prasanna, 2009): A lista de agentes da simulação é obtida diretamente da ontologia inferida por um *reasoner* de logica descritiva, junto aos papéis e consequentes comportamentos e propriedades. Os agentes são instanciados de acordo com suas classes e, subsequentemente, seus comportamentos, que são injetados utilizando a técnica citada. Ao final, as propriedades de dados podem ser utilizadas para ajuste de valores iniciais da simulação. Ressalta-se que o avanço do tempo da simulação somente inicia após a carga completa dos agentes em memória, ie. a instanciação da simulação.

Esta ontologia, criada para representar um subconjunto do domínio de guerra anti-submarino, poderá ser estendida para representar elementos do teatro de guerra de superfície ou aérea. Isto descreve como o reuso deste componente que representa o modelo conceitual permitiria acelerar o desenvolvimento de novas simulações. Especificamente para a execução da pesquisa, esta qualidade permitiu modificar rapidamente os cenários e elementos que influenciavam no tardiness do sistema.

3.6 Trabalhos Relacionados

O trabalho apresentado em (Drogoul et al., 2003) discute o vazio (*gap*) existente entre os especialistas de domínio e os engenheiros de software, assim como a dificuldade em compartilhar o conhecimento e utilizar uma representação con-

sensual entre os grupos. O MASP propõe o uso de uma ontologia para realizar a modelagem que: possui um formalismo que possibilita sua verificação quanto a consistência; tem uma expressividade superior aos modelos comumente utilizados, tais como diagramas UML ou modelos de dados; permite processamento automático, que o MASP utiliza para instanciar a simulação.

Outros trabalhos sobre aplicação de ontologias em simulações foram apresentados, tais como (Chistley et al., 2005). Porém este utiliza ontologia para modelar o processo de simulações, e não o sistema simulado. Em (Silver et al., 2007) é apresentada uma ontologia para simulações de eventos discretos, que utiliza uma ferramenta para gerar a ontologia em uma extensão de XML. A ontologia possui um acoplamento forte com o sistema nesta abordagem. A proposta implementada no MASP utiliza uma ontologia descrita em uma linguagem padronizada que possa ser alterada sem afetar a engine de execução. A ontologia gerada em (Silver et al., 2007) também não considera modelagem orientada a agentes, e portanto não é adequada a pesquisa proposta.

Em (Benjamin et al., 2006), os autores apresentaram as vantagens em utilizar ontologias para modelagem conceitual de simulações, contudo não apresentam uma arquitetura ou engine que possa acessar a ontologia de forma automática. O trabalho não tem como objetivo simulações dirigidas por agentes ou modelagem de atores.

O uso de ontologias para modelar ambientes, contextos e domínios em sistemas multiagentes foi apresentado em (Felicissimo et al., 2006a; Felicissimo et al., 2006) e (Taranti & Choren, 2007b; Taranti & Choren, 2007a). Estes trabalhos têm como objeto o estudo de regulação de SMA, não sendo diretamente aplicáveis a simulações.

3.7 Conclusão do Capítulo

Este capítulo apresentou os resultados da primeira fase da pesquisa, cujo objetivo foi definido na primeira questão secundária de pesquisa, e restrito pela hipótese H1, apresentados em 1.1.1, sejam:

QS1: *Obter uma ferramenta de suporte que permita a rápida instanciação dos experimentos para a pesquisa em curso, na qual a modelagem da simulação possa ser realizada através de arquivos de configuração.*

H1: *O uso de ontologias, associada a técnicas de inferência lógica e de injeção de dependências, permitiria a rápida construção e alteração de modelos de execução.*

Conquanto a arquitetura do MASP tenha sofrido contínua evolução desde sua versão inicial, os resultados iniciais foram apresentados em (Taranti et al., 2010),

com enfoque nas vantagens em se acoplar o modelo conceitual da simulação à engine de simulação.

O MASP foi utilizado em todas as atividades posteriores da pesquisa, nos experimentos e testes de conceito. Conquanto não tenha sido o enfoque principal desta etapa, o resultado que mais despertou interesse acadêmico foi a ontologia apresentada na figura 3.2.

A fase seguinte da pesquisa, que objetivava confirmar ou negar as hipóteses apresentadas no item 1.1.2 é apresentada no próximo capítulo.