2
Referencial Teórico

Este capítulo do texto apresenta, dividido por tópicos, o referencial teórico associado ao trabalho. Cada um dos tópicos se atém as especificidades relacionadas à pesquisa.

2.1
Sistemas Multiagentes

O conceito de agentes de software tem origem nos estudos de inteligência artificial (AI), onde encontramos a definição de agente como sendo tudo que é capaz de perceber seu ambiente por meio de sensores e de agir neste ambiente por meio de atuadores (Russel & Norvig, 2003), que é coerente com a arquitetura *Subsumption* (Brooks et al., 1986). Neste ramo da ciência, surgiram expressões como Agentes Inteligentes, *SoftBots* e outros. Agentes também são definidos como sendo sistemas computacionais situados em um ambiente e capazes de empreender ações de forma autônoma sobre este ambiente a fim de cumprir suas metas de projeto (Zambonelli & Jennings, 2001; Wooldridge, 2002). A decisão sobre qual ação será empreendida é tomada pelo agente seguindo sua programação, podendo utilizar informações provenientes de sensores e de sua representação interna do ambiente neste processo. O conjunto de influências no processo de tomada de decisão e a arquitetura interna do agente que apoia este processo criam a classificação de agentes reativos e agentes cognitivos, que usam a arquitetura *Belief-Desire-Intention* (BDI) (Rao & Georgeff, 1997).

Sistemas Multiagentes (SMA) são sistemas nos quais agentes interagem entre si e com o ambiente (Zambonelli & Jennings, 2001; Wooldridge & Jennings, 1995). Esta comunicação pode ocorrer através de mensagens explicitas, como as estabelecidas na *Agent Communication Language* (ACL), ou de forma indireta, como apresentado nos trabalhos (Weyns et al., 2005), nos quais informações disponibilizadas no sistema são acessíveis aos demais agentes em uma solução semelhante a um *blackboard*.

Em SMA, o ambiente no qual os agentes estão imersos e interagem agrega novas responsabilidades como: estruturar o SMA, fornecer abstrações para acesso a recursos e serviços pelos agentes, manter a dinâmica do sistema e definir regras que

regulem o sistema, mediando as interações realizadas pelos agentes (Weyns et al., 2007).

SMA que representem uma metáfora do mundo real podem possuir ambientes que representem o mundo físico, contendo representação no espaço virtual. Essa representação pode ser referenciada em coordenadas no espaço virtual, i.e. geo-referenciadas, e devem ser interpretadas corretamente pelos agentes. O uso de SMA para representação de dinâmicas espaciais é um campo de pesquisa ativo e que já possui algumas soluções em uso, tal como o acoplamento fraco entre SMA e sistemas de informação geográficas (GIS) (Parker, 2005; Gonçalves, 2004; Brown et al., 2005).

Devido à autonomia dos agentes para decidir individualmente sobre as ações que executam, não é possível prever, em tempo de projeto, o conjunto de todas as interações entre agentes ou entre estes e o ambiente (Jennings, 2000). Desta forma, não se pode garantir, *a priori*, o tempo necessário para a execução do sistema. Caso a corretude do sistema dependa não somente de sua lógica de execução, mas também de sua capacidade de executá-la no tempo previsto, surge um problema de confiança na solução implementada.

Em (von Staa, 2006) confiança no software é apresentada como uma das características da fidedignidade (do inglês *dependable*) que é um conceito mais amplo e tem relação com a fé que se possui no software, baseado em características como a já citada confiança, disponibilidade, segurança, manutenibilidade, depurabilidade, robustez, entre outros. Para que SMA sejam aceitos na indústria, e não somente na academia, é necessário que existam recursos suficientes para garantir-lhes a fidedignidade.

Para construção de sistemas fidedignos é necessário limitar a incerteza (Neumann, 1995). Porém esta tarefa pode ser dificultada pela natureza do próprio sistema modelado e implementado, como é o caso da representação de sistemas complexos utilizando-se SMA com agentes autônomos.

2.1.1 Representação de Sistemas Complexos com SMA

O paradigma de SMA pretende prover os engenheiros de software com ferramental e conceitos necessários para analisar, projetar e implementar sistemas de software complexos (Jennings, 2001).

Por utilizarem conceitos do mundo real na fase de *design* e implementação, o paradigma de orientação a agentes (POA) oferece uma facilidade inerente para modelagem de sistemas reais (Dignum & Dignum, 2001). Esta facilidade tem destaque na modelagem de sistemas complexos, que são sistemas de comportamento não-linear, nos quais pequenas alterações que ocorram entre os elementos de me-

nor granularidade do sistemas podem se propagar e como consequência, alterar o estado geral do sistema em nível macro (Moffat, 2003), situação similar a descrita em (Lorenz, 1963) . Os trabalhos apresentados em (Moffat, 2003; Pidd, 2004) sugerem que para estes sistemas complexos, o uso de uma abordagem multiagentes é a mais adequada por permitir a representação do nível micro do sistema, de maior granularidade. Desta forma torna-se possível a observação as interações neste nível e suas consequências sobre o comportamento global do sistema.

Apesar destas facilidades que a POA oferece para modelar sistemas complexos, o suporte ao desenvolvimento ainda é limitado e de uso restrito a pessoal especializado. Especificamente sobre a implementação de sistemas que possuem restrições temporais a serem respeitadas, o suporte em SMA é ainda mais escasso, conforme discutido no trabalho (Dignum, 2011).

2.1.2
Suporte de Java para desenvolvimento de SMA

Para apoiar o desenvolvimento de Sistemas multiagentes um número de metodologias, toolkits, frameworks e plataformas têm sido desenvolvidos, conforme apresentado em (Bellifemine et al., 2005; Bordini et al., 2006). Estas ferramentas, conquanto sejam desenvolvidas com uso de diversas linguagens e tecnologias, possuem um considerável subconjunto baseado na tecnologia Java. São exemplos deste conjunto: Jack (Evertsz et al., 2004), JADE (Bellifemine et al., 2007), Cougaar (Helsinger et al., 2004), Jadex (Pokahr et al., 2005), Jason (Bordini et al., 2007) e BDI4Jade (Nunes et al., 2011).

Os sistemas desenvolvidos utilizando tecnologia Java compartilham características inerentes a esta tecnologia e a sua linguagem de programação: Java é fortemente tipada e criada para implementar sistemas Orientados a Objetos (OO), conquanto esta característica possa ser escondida do programador pelas ferramentas de desenvolvimento de SMA.

Outra característica comum é que as soluções desenvolvidas em Java são executadas sobre uma maquina virtual (JRE). Ocorre que existem diversas implementações de JRE no mercado, com características de desempenho diferentes.

Outros pontos relativos ao uso da tecnologia Java aplicada a SMA é que esta permite a criação de soluções distribuídas com alguma facilidade; existe considerável suporte de bibliotecas de apoio ao desenvolvimento. Contudo, as soluções de SMA desenvolvidas são submetidas às mesmas restrições que a tecnologia Java possui, a exemplo da não garantia de execução de tarefas agendadas no tempo previsto. Estas restrições tornam o uso de Java para MABS um caso mais complexo no que tange o problema de *tardiness*. Uma solução para este caso provavelmente teria potencial para ser generalizada.

2.2 Simulação

Simulações computacionais são execuções de modelos que representam o comportamento dinâmico de sistemas específicos, e que são realizadas para verificar ou explorar o comportamento destes sistemas. O desenvolvimento de simulações está fortemente ligado ao setor de defesa, e mais recentemente, pesquisa e entretenimento (Fujimoto, 2000).

Simulações podem ser usadas por diversos motivos, pois permitem, por exemplo:

- realizar experiências *in silico* que não podem ser realizadas em ambiente real (ex. avaliar as consequências do rompimento de um dique de represa);
- testar sistemas e planos antes de sua implementação;
- realizar avaliações com rapidez, prospectando cenários diferentes (ex. avaliar influência de uma variável em um sistema estocástico); e
- treinar pessoas em sistemas que não estão disponíveis ou com grande redução de custos.

O processo de construção de simulações é composto por três diferentes processos, que são fortemente acoplados entre si (Fishwick, 1995):

- Design do modelo;
- Execução do Modelo; e
- Análise do modelo.

O Modelo inicial normalmente não representa características dinâmicas do modelo, mas os conceitos do mesmo e os relacionamentos que os associam.

Esse modelo pode ser representado sob diversas instâncias, sendo exemplos: Modelos Conceituais, Modelos Declarativos, Modelos Funcionais, Modelos de Restrições , Modelos Matemáticos e Modelos Espaciais (Fishwick, 1995).

O modelo conceitual é amplamente utilizado para modelagem de simulações (Birta & Arbez, 2007), e conta com suporte de metodologias e linguagens de representação, a exemplo da UML.

Na visão do autor desta pesquisa, o uso de modelos conceituais para fase de design é adequado para simulações baseadas em sistemas multiagentes, uma vez que estes modelos são utilizados para representar domínios de aplicação de sistemas de software e possuem expressividade superior aos demais para representação de sistemas complexos.

O modelo conceitual é, então, uma representação do problema em estudo, cuja implementação é o modelo de execução (Sargent, 2005).

A tarefa de verificação é realizada comparando-se o modelo conceitual com o modelo de execução, ou seja, pretende confirmar se o que foi implementado é o que se especificou. Já a tarefa de validação visa comparar os resultados obtidos da simulação com o sistema original, a fim de verificar a coerência entre os dois sistemas. A figura 2.1 apresenta graficamente as relações de verificação e validação entre os modelos utilizados em simulação (Drogoul et al., 2003).

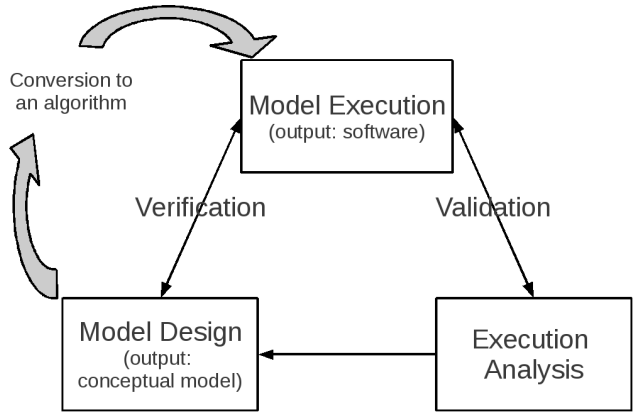

Figura 2.1: Verificação e validação @ (Drogoul et al. 2003)

2.2.1 Tempo de simulação e Classificação de Simulações

Uma vez que simulações representam a dinâmica de um sistema ao longo do tempo, a modelagem temporal é primordial neste campo. Para representar esta dinâmica, uma abstração do tempo real (i.e do tempo físico) é usada na construção do modelo de execução. Esta abstração é o *tempo da simulação*, cujo avanço durante a execução da simulação não possui necessariamente uma relação funcional com o tempo real (Fujimoto, 2000).

As opções sob como representar conceitos e dinâmicas em um modelo de execução estão relacionadas ao propósito que este modelo possui e às características do sistema real a ser simulado. As abordagens de representação do tempo real em tempo de simulação, e como realizar seu avanço também se sujeitam a estas restrições.

Abaixo são apresentadas as principais classificações apresentadas na literatura.

Modelos contínuos e discretos

A classificação mais aceita para sistemas dinâmicos é a de sistemas discretos e contínuos. Em sistemas discretos, a transição entre estados é executada instantaneamente, ou seja, o sistema pode ser representado em tuplas que associam estados a um instante no tempo de simulação. Em sistemas contínuos, o estado do sistema

evolui em função do tempo, portanto, para cada instante T_s do tempo de simulação corresponde um conjunto de estados (Law & Kelton, 1991; Gordon, 1977)[1].

Para simular sistemas contínuos em ambientes virtuais com apoio de computação, estes sistemas são modelados como sistemas discretos, ou seja, o estado do sistema é atualizado em passos regulares do tempo, com perda de informações e precisão. Esta perda é relativa aos períodos entre as atualizações, que não são representados ou avaliados. Quanto menor o passo adotado, menor a perda de informações e mais próxima ao sistema original a simulação estará, porém existe uma relação de custo-benefício em detrimento a carga computacional adicional. A palavra *discretização* é usada para representar este processo.

O projetista da simulação é responsável por determinar o maior erro aceitável para representar o sistema de forma discreta.

Modelos de avanço do tempo de simulação

Uma vez que se possua um sistema com características discretas, e conhecendo seu comportamento ao longo do avanço do tempo real, é possível modelar o conceito de tempo real em tempo da simulação. O conceito de tempo da simulação pretende representar o quanto o universo simulado (i.e virtual) avançou em relação a dimensão temporal.

A abordagem utilizada para avançar o tempo de simulação depende do tipo de simulação executada e de sua finalidade. Historicamente, simulações de eventos discretos são divididas em duas classes diferentes em relação a forma em que o avanço do tempo da simulação é realizado, seja por avanço ao próximo evento ou avanço do tempo de simulação por passos (Law & Kelton, 1991; Zeigler et al., 1999).

A abordagem de avanço ao próximo evento utiliza uma fila de eventos, que pode ser atualizada ao longo da simulação. A cada novo evento executado, o tempo da simulação é acrecido do tempo necessário a execução deste evento no mundo real. As simulações que usam esta técnica são ditas Simulações Dirigidas por Eventos ou DEVS.

Na abordagem de avanço de tempo por passos, o estado de todos os elementos do sistema é atualizado sequencialmente para cada incremento realizado no tempo da simulação. Estes acréscimos no tempo da simulação podem ser realizados imediatamente após a computação da última atualização e determinam o avanço do modelo no tempo de simulação.

Ambas as abordagens de avanço permitem, respeitados certos limites de erro e utilizada a engenharia de software correta, a construção de simulações com

[1] T_s é um número Real

abordagem *fast–as–possible*, que tem por objetivo a produção de dados para análise analítica.

Um caso particular de avanço do tempo de simulação ocorre quando este possui relação linear com o tempo real: nesta situação o avanço de tempo da simulação é proporcional ao tempo real. Estas simulações são ditas *paced–time* ou *escaled–time*. Quando a proporção entre tempo de simulação e tempo real assume valor 1:1, observa-se uma simulação em tempo real (Fujimoto, 2000).

Simuladores de voo e simuladores de apoio a jogos de guerra e de empresas são exemples de aplicações que fazem uso de avanço do tempo de simulação proporcional ao tempo real.

Este trabalho tem escopo nas simulações baseadas em sistemas multiagentes *paced-time* e que possuem flexibilidade para permitir a variação do passo de avanço. Desta forma, as simulações *Real-Time* estão fora do escopo.

Simulações locais, paralelas e distribuídas

As simulações podem também ser classificadas conforme o local de execução. A forma mais comum é a execução em uma única máquina ou local. Já as simulações paralelas e as distribuídas utilizam o processamento distribuído por mais de um *host*. Conquanto o conceito não esteja pacificado na literatura, o termo simulações paralela é normalmente aplicado ao uso de *clusters* em redes locais de alta velocidade, já simulações distribuídas seriam as que utilizam-se de vários *host* distribuídos geograficamente, normalmente fazendo uso da Internet para tramitação de dados (Fujimoto, 2000).

Diversos fatores sugerem o uso de simulações paralelas e/ou distribuídas, entre eles:

- Redução do tempo de computação;
- Necessidade de distribuição geográfica;
- Integração de simuladores que executam em host de diferentes fornecedores. Exemplificando, A simulação de um modelo marítimo pode utilizar o simulador de radar que executa no laboratório de seu fabricante, que autoriza o uso do modelo, mas não sua retirada das instalações da empresa;
- Aumento de tolerância a falhas; e
- Jogos *multi-players*.

A complexidade das simulações atuais, aliada a fatores externos, como questões comerciais e limitação de recursos, favorece a criação de infraestruturas distribuídas e modelos que possam ser reusados em várias simulações.

2.2.2 Simulações de Ambientes Virtuais

Uma classe especial de simulações é destinada a criação de ambientes virtuais (VES). Estas simulações são tipicamente desenvolvidas para treinamento ou entretenimento.

Um ponto de destaque no conceito de VES é que se busca criar um ambiente "suficientemente realista" para que os objetivos da simulações sejam atingidos. Por exemplo, em um jogo via Internet, um retardo de 0,5 segundos pode ser desagradável, porém normalmente não comprometerá o entretenimento. Já em um simulador de voo, após o piloto acionar seu manche, caso a imagem não se rotacione em menos de 200 milissegundos, o piloto perceberá o comportamento anômalo. Contudo, dentro do mesmo ambiente do simulador de voo, caso o piloto esteja perseguindo um caça inimigo e durante a perseguição a imagem do solo apresentar pequenas falhas de renderização, isto não afetará o objetivo final.

Com isso destaca-se que dentro de um mesmo ambiente (VES), os elementos da simulação possuem limites de erro por atraso diferentes entre si, limites estes que podem variar durante a execução da simulação. Citando novamente o exemplo do simulador de voo, caso um elemento aumente sua velocidade no ambiente virtual, sua capacidade de detecção deve ser ampliada, ou o erro de detecção reduzido, para evitar-se que alvos que deveriam ser detectados sejam descartados em consequência dos atrasos.

Ambientes Virtuais Distribuídos

A necessidade de realizar simulações de treinamento distribuídas na área da Defesa dos EUA levou ao desenvolvimento de padrões. Os dois padrões abertos utilizados são o *Distributed Interactive Simulation* (DIS) e o *High Level Architecture* (HLA). Ambos são utilizados para simulações real–time.

O DIS normalmente é aplicado em redes locais, trafegando pacotes UDP em *broadcast*. O controle do tempo depende de um relógio local, ao qual as máquinas envolvidas sincronizam usando protocolos externos a simulação (ex. NTP).

Já o HLA, atualmente na versão HLA Evolved (HLAe), que é um padrão ISO, utiliza o padrão *publish/subscribe* e uma abstração de tempo chamada tempo lógico. Somente dados solicitados trafegam na rede. Este padrão depende de um barramento de infraestrutura, o RTI (run-time environment). Um detalhe importante sobre o HLA, que afeta esta pesquisa, é que caso um simulador não seja capaz de acompanhar a simulação por falta de recursos, o avanço geral do tempo de simulação é mantido e o elemento em atraso é excluído do sistema, ou des-federado.

HLA e DIS podem ser utilizados em composição: um batalhão de cavalaria pode treinar em simuladores de tanques de guerra que são conectados em rede

local por DIS, e em outro estado um grupo de aviação pode estar da mesma forma com uma esquadrilha de helicópteros simulados em rede local por DIS. As duas simulações podem interoperar através de um barramento HLA, de forma que a esquadrilha aérea possa engajar o batalhão de cavalaria em solo do ambiente virtual, apesar dos simuladores estarem distantes por centenas de quilômetros. A literatura possui diversos exemplos deste tipo de treinamento, como os apresentados em (Smith 1998), (Fujimoto, 2000) e (Strassburger et al., 2008).

2.2.3 Simulação Baseada em sistemas Multiagentes

As facilidades que a orientação a agentes oferece para modelar sistemas complexos, já citada no item 2.1.1, sugere também seu uso no campo da simulação. Diversas áreas vem aplicado o paradigma em seus experimentos (Drogoul et al., 2003), como negócios (North & Macal, 2007), militar (van Doesburg et al., 2005; Cares, 2002) e testes de software (Uhrmacher & Kullick, 2000; Taranti et al., 2009).

Além da citada vantagem para modelagem de sistemas complexos, o paradigma de orientação a agentes também permite que ao se implementar os atores modelados como agentes de software, criem-se sistemas que respeitem os conceitos de modularização e encapsulamento, mantendo coerência sintática e semântica com o modelo conceitual. O trabalho (Uhrmacher, 2007) discute o uso dos agentes para modelagem.

Avanço do Tempo de Simulação em MABS

Ferramentas e frameworks para desenvolvimento de MABS, tais como Mason (MASON, 2013), Swarm (SWARM, 2013) oferecem suporte para avanço do tempo de simulação para o próximo evento ou por incremento de passo e ao próximo evento. O NetLogo (Wilensky, 2013) atende especificamente a especificação de sistemas de tempo discreto (DTSS). Nestes *frameworks*, o suporte para avanço de passo se dá pela técnica que impõe o uso de um máximo denominador comum entre as ações cíclicas, sem previsão de alteração de passo ao longo da execução. As técnicas de avanço normalmente são a adoção de avanço *fast-as-possible* ou *paced-time*. No primeiro determina-se aos agentes atualizarem seu estado ao novo tempo da simulação, e assim que todos completam a atualização, um novo incremento é aplicado, e assim sucessivamente até o término da simulação. A segunda abordagem prevê que a cada T_r milissegundos do tempo real, os agentes deve se atualizar T_s milissegundos, seguindo uma função linear $T_S = K * T_r$, sendo que K é o coeficiente linear que relaciona o tempo real e o da simulação. Essa abordagem considera, otimisticamente, que o tempo real T_r é suficiente para os agentes executarem o

processamento necessário para atualizarem seu estado ao instante T_s do tempo de simulação.

Outro ponto de destaque é que não cabe aos agentes a decisão sobre o avanço do tempo de simulação. Este é incrementado pela *engine*. Da mesma forma, quando a técnica de avanço de tempo de simulação adotada é a de avanço ao próximo evento (DEVS), cabe a *engine* gerenciar a fila e definir os avanços.

2.3 Tempo Real

Sistemas de Tempo Real são sistemas nos quais a corretude lógica depende tanto da corretude dos algoritmos quanto da precisão em executar as ações no tempo previsto (Laplante, 2004).

As aplicações de Tempo Real podem ser classificadas como *hard*, *firm* e *soft*, de acordo com o nível de acurácia exigida em relação ao cumprimento dos tempos agendados (Baskiyar & Meghanathan, 2005; Cedeno & Laplante, 2007):

- Os sistemas *hard real-time* entram em estado de falha caso os tempos de resposta esperados não sejam atingidos.
- As aplicações *firm real-time* possuem uma pequena flexibilidade quanto a estes limites de tempo.
- Os sistemas *soft real time* são os sistemas que têm sua performance degradada, mas que continuam operando mesmo quando falham em cumprir as restrições de tempo.

A qualidade que um sistema possui em cumprir suas restrições temporais no instante previsto é a *timeliness*, e a medida de atraso entre a execução e o que foi agendado é a *tardiness*.

2.3.1 Java e tempo real

A tecnologia Java fornece um padrão para apoiar o desenvolvimento de sistemas de tempo real: a especificação *Java Real Time* (RTJ), registrada como *Java Specification Requests* 001 (JRS001). Esta especificação modifica o mecanismo de agendamento de *thread* da Java Runtime Environment (JRE) e a forma como as *thread* com restrições de tempo devem ser implementadas. As alterações impostas pelo uso da especificação incluem alterações no mecanismo de gerencia de memória do Java, incluindo a inserção de acesso físico à área específica de memória. Esta especificação permite a criação de executáveis binários que são dependentes da plataforma de execução (Bollella et al., 2000).

Algumas JRE que implementam a especificação RTJ são disponíveis no mercado, contudo todas localizadas nesta pesquisa são software proprietário com custo associado ao número de JRE utilizadas. O corrente cenário é que a especificação é pouco usada: mesmo na indústria não existe conhecimento difundido sobre a especificação, produtos e a programação em Java Real–Time. A especificação apresenta dificuldade elevada de uso, exigindo do desenvolvedor habilidades em administrar o gerenciamento de memória, habilidade que não é de amplo conhecimento.

Como resultado, restrições temporais em Java são comumente implementadas utilizando o *software development kit* (SDK) padrão do Java (JDK), no qual existem mecanismos para tratar restrições temporais utilizando agendamentos de ações e que são capazes de atender sistemas que não possuam fortes restrições temporais. Estes mecanismos incluem a capacidade de agendar a execução de ações para um dado instante ou suspender a execução de uma ação/*thread* durante período determinado.

O Java SDK 5 e versões posteriores passaram a disponibilizar este suporte no pacote *java.util.concurrent*, que possui diversas interfaces e classes. Especificamente os objetos criados a partir da classe *ThreadPoolExecutor* podem agendar comandos para execução futura ou periódica.

Um fator muito importante sobre estes agendamentos, e que é responsável por um atraso intrínseco aos mesmos, em adição aos atrasos decorrentes do controle não-preemptivo de *thread* e do garbage collector (GC), é que as ações são agendadas para executar não antes de determinado instante. O trecho do algoritmo que verifica o instante de execução verifica se o instante atual é maior ou igual ao instante agendado para a execução. Em resumo, o caso comum é que o instante é maior e a diferença entre o agora e o agendado é um atraso. Ocorre que duas *thread* agendadas para executarem em um mesmo instante são transferidas do estado de pronto para o estado de execução seguindo a ordem de fila FIFO. Caso estejam executando em uma máquina monoprocessada, nunca executarão simultaneamente.

Para controlar a execução de *thread* Java, pode-se optar por especificar a prioridade de cada *thread* do software. Esta abordagem preemptiva é possivel em Java, conquanto não seja usualmente aplicada. As *thread* definidas em Java, em tempo de execução, serão mapeadas pela JRE para *thread* do Sistema Operacional (SO), em um mapeamento *n* para *n*. Em última instância, o controle de suspensão e ativação de *thread* cabe ao SO (Silberschatz et al., 2005; Oaks & Wong, 2004).

Adicionalmente, a quantidade de níveis de prioridade para *thread* dos SO é usualmente maior que a quantidade de níveis existentes em Java, existindo dúvidas sobre como ocorrerá efetivamente o mapeamento não somente das *thread*, mas também de suas prioridades.

Caso um computador se encontre com uso intensivo de processador, o que

pode ocorrer devido a rotinas de *backup*, por exemplo, as thread do *kernel* do sistema operacional passarão a competir por recursos e possivelmente as *thread* que são mapeadas para thread java poderão sofrer suspensões, dando origem a aumento do nível de tardiness no sistema Java em execução. Em (Oaks & Wong, 2004) é apresentado um experimento que mostra que o mesmo código executando em diferentes thread Java apresentou tempos diferentes.

O GC insere outro fator de risco para ocorrência de *tardiness*, pois pode suspender a execução de um sistema implementado em Java, que parecerá não responder durante o período em que o GC varre a *heap* para liberar memória. A execução do GC é controlada pela JRE, sem influência do programa em execução, e pode causar atrasos em todas as *thread* em execução.

Finalmente, implementações com restrições de tempo em Java são comuns, tanto nas especificações Java SE quanto Java EE. Contudo a tecnologia não foi desenvolvida para sistemas de tempo real, a exceção da especificação JRT. Isto não impede o desenvolvimento de aplicações Java para sistemas soft real–time, ou mesmo firm real–time, quando não se tratarem de sistemas críticos. Contudo, a capacidade do sistema atender os agendamentos realizados depende de vários fatores, e isto deverá ser considerado no projeto.

Uma solução para a questão principal de pesquisa que atenda simulações baseadas em sistemas multiagentes e implementadas em Java apresenta a vantagem de atender um caso mais complexo, que apresenta as dificuldades já descritas da tecnologia no que tange a sistemas de tempo real. A generalização posterior da solução para tecnologias que possuam acesso direto ao sistema operacional e a memória primária é possível, pois estas são um cenário mais simples que o estudado.

2.4 Ontologias

As ontologias utilizam um formalismo declarativo para permitir a representação de conceitos, de suas relações e propriedades (Gruber, 1991; Gruber, 1993). Além disto, as ontologias são uma ferramenta para construir a base de conhecimento sobre domínios específicos.

A definição para ontologia de Gruber é amplamente utilizada na literatura, na qual ontologia é *uma especificação formal e explícita de uma conceitualização compartilhada* (Gruber, 1991; Gruber, 1993) , onde:

"*..**conceitualização** representa um modelo abstrato de algum fenômeno que identifica os conceitos relevantes para o mesmo. **Explícita** significa que os elementos e suas restrições estão claramente definidos; **formal** significa que a ontologia deve ser passível de processamento automático, e **compartilhada** reflete a noção*

de que a ontologia captura conhecimento consensual, aceito por um grupo de pessoas." (Breitman, 2005).

2.4.1
Uso de Ontologias para modelar sistemas

O uso de ontologias para modelar sistemas é alvo de estudos há mais de duas décadas, sendo que os trabalhos de (Fox et al., 1996), (Wegmann & Naumenko, 2001), (Evermann & Wand, 2005) são representativos sobre o tema.

Ressalta-se que nesta época não havia uma linguagem de representação de ontologias padronizada e que permitisse o processamento por computadores. Linguagens como a RDF , SHIQ, DAML, OIL (McGuinness et al., 2002; Horrocks et al., 2003) surgiram e ganharam corpo em pesquisas associadas a Web Semântica. A principal linguagem da área, a *Ontology Web Language* (OWL), recebeu a recomendação da W3C somente em 2004 (W3C, 2004).

Atualmente existe suporte para modelagem de ontologias através de plataformas maduras, como o Protégé (Protege, 2013). Adicionalmente, estão disponíveis *framework* para manipulação das ontologias em memória e realização de inferências.

O uso de ontologias para modelagem de sistemas não se atém a Engenharia de Software, sendo os exemplos mais representativos encontrados na área médica, como, por exemplo, a modelagem de células tronco.

2.4.2
Uso de ontologias para modelar simulações

A possibilidade de utilizar ontologias em apoio a simulações é explorada em diversos trabalhos. Resumidamente, observa-se trabalhos nos quais a ontologia representa o processo da simulação, como em (Chistley et al., 2005). Em (Benjamin et al., 2006) o uso de ontologias para modelar o sistema a ser simulado é explorado, utilizando uma ontologia de domínio que é estendida. O trabalho (Okuyama et al., 2006) apresenta a *Environment Description Language for Multi-Agent Simulation* (ELMS) e o uso de uma ontologia para modelar o sistema. Esta trabalho, em sua fase final apresentava suporte geoposicionamento.

O trabalho apresentado em (Silver et al., 2007) discute uma abordagem na qual o sistema a ser simulado é modelado em uma ontologia, a qual é utilizada posteriormente para gerar o modelo executável.

No contexto desta tese, ontologias serão utilizadas como ferramentas para modelagem e instanciação de simulações, como será apresentado no próximo capítulo.