4
Verificação de hipóteses

Recordando a principal questão de pesquisa abordada por este trabalho, apresentada em 1.1 é:

QP: *Obter uma solução que permita a execução de simulações baseadas em sistemas multiagentes nas quais os atrasos medidos em tempo de simulação sejam controlados a níveis pré–estabelecidos e nas quais cada agente possua o controle de sua linha de execução.*

A pesquisa se atém ao subconjunto de sistemas multiagentes desenvolvidos em tecnologia Java que implementem simulações nas quais cada agente possua o controle de seu avanço no tempo de simulação.

A segunda fase da pesquisa, apresentada neste capítulo, foi dedicada a comprovar a relação de atrasos em tempo de simulação com abstrações utilizadas em sistemas multiagentes. O objetivo era, então, identificar as variáveis que fossem candidatas a uso em uma abordagem para controle dos citados atrasos, para, numa etapa posterior, definir-se como se atuaria nos parâmetros destas variáveis.

Uma consideração importante ao objetivo é que os atrasos em tempo de simulação são reflexos de atrasos ocorridos em tempo real nos sistemas de simulação que adotam a abordagem *paced-time* : em sistemas que usam esta abordagem, existe a relação $\Delta t_s = K.\Delta t_r$, aonde t_r representa o tempo real, ou físico, e t_s representa o tempo de simulação.

Portanto, para agendar uma tarefa para execução em determinado instante tempo de simulação t_{s1}, a *engine* verifica o instante no tempo real correspondente (t_{r1}) e agenda a tarefa para aquele instante. Em tempo de execução, o instante t_{r1} é facilmente verificável pela engine de simulação através do uso de chamadas ao sistema operacional.

Considerando a relação acima, foi conduzido um experimento para, então, isolar conceitos de SMA que pudessem influenciar no nível de *tardiness* do sistema medido em t_r.

Este capítulo busca associar quais abstrações de agentes possuem relações com atrasos de execução em tempo real, e consequentemente, em tempo de simulação para simulações *scaled-time*, ie. *paced-time*.

4.1 Sistemas Multiagentes Implementados em Java e Tardiness

Diversas aplicações que usam Programação Orientada a Agentes têm sido apresentadas na literatura, tais como sistemas para controle de trafego aéreo, de leilão, para controle de linhas de produção em fábricas, logísticos e de simulação. Restrições temporais são um tipo de requisito não-funcional frequente em tais sistemas, que pretendem controlar a execução de processos.

Uma vez que algumas ações destes sistemas são agendadas para execução em instantes no futuro, a corretude lógica do sistema em execução depende também de sua capacidade em executar estas ações no tempo programado. Em outras palavras, a corretude do sistema depende também de sua precisão no tempo, ou *timeliness*.

Esta característica classifica os ditos sistemas como Sistemas de Tempo Real (vide item 2.3). O indicador usado neste trabalho para verificação de *timeliness* é o *tardiness* do sistema, que indica a diferença entre o instante para o qual uma ação foi agendada e o instante no qual ela efetivamente foi executada.

A construção de sistemas com características de tempo real que utilizem o paradigma de Orientação a Agentes é dificultada por características, inerentes a estes sistemas:

- Frequentemente não é possível determinar-se o estado corrente do sistema em nível macro, em virtude dos diversos estados que os agentes podem assumir em nível micro , situação análoga à apresentada por sistemas complexos apresentada em (Moffat, 2003);
- Os agentes se comunicam com mensagem assíncronas, não havendo certeza sobre quando a resposta será enviada. Exemplificando, o uso do padrão ACL da FIPA (FIPA, 2013) considera que cabe ao agente decidir sobre a resposta que dará a uma requisição recebida;
- Devido a autonomia dos agentes, um dos fundamentos do paradigma, não é possível, *a priori*, afirmar qual será a decisão de um agente em função de mensagem recebida ou como consequência da percepção de uma alteração no ambiente; e
- SMA são sistemas intensivos em concorrência, fator que, por si, dificulta a construção de sistemas com características de tempo real.

Estas características de complexidade dos SMA impedem que os arquitetos de sistemas, durante a fase de projeto, conheçam todos estados que o sistema poderá assumir.

Outra dificuldade encontrada em projetos de SMA com restrições temporais é a ausência de as metodologias que apoiem a modelagem destes sistemas. As principais metodologias se atém a conceitos essenciais do paradigma, tais como atores,

comportamentos, ações, metas e planos, e não possuem sequer representação para restrições temporais, como pode ser observado nas metodologias Gaia (Zambonelli et al., 2003), Tropos (Bresciani et al., 2004) e Prometeus (Padgham & Winikoff, 2003). O mesmo se pode afirmar da metodologia i*, frequentemente citada na literatura de SMA (Yu, 1997).

O vazio de suporte à modelagem é suprido de forma *ad-hoc*, e cabe aos desenvolvedores de SMA realizarem a implementação das restrições temporais. A solução usual adotará implementação de mecanismos de execução periódica (*time–step*), nos quais se define um intervalo de tempo entre execução, ou de mecanismos nos quais um instante futuro para início da execução é indicado (*time-to-execution*).

A tecnologia Java oferece um suporte limitado para implementar esses agendadores de tempo (i.e *schedulers*). Em baixo nível de programação, os mecanismos se refletem no uso do tempo real corrente, obtido através de chamadas ao sistema operacional. Este tempo é comparado com o instante esperado para execução das tarefas agendadas.

O suporte usado na programação de agentes é o mesmo usado na programação *multi–thread*, ainda que o fato seja encoberto pelo uso de bibliotecas que ofereçam recursos de uso direto, como ocorre nas plataformas JADE e Cougaar (Helsinger et al., 2004). Estas plataformas oferecem mecanismos de controle de tempo para execução de ações dos agentes, que implementam o mecanismo descrito, facilitando a tarefa do programador.

A tecnologia Java não oferece garantias sobre a precisão de execução no tempo previsto quando são usados mecanismos de agentes: as execuções ocorrem a partir do instante agendado. Considerando este fato, um SMA poderia atingir estados de erro devido a atrasos individuais de seus agentes, atrasos estes que não podem ser previstos devido a complexidade em se determinar os estados futuros de um agente de software. Neste cenário, a ocorrência de *tardiness* pode representar risco a operação do sistema.

As implementações de simulações que utilizem sistemas multiagentes desenvolvidos em Java são afetadas também pelos atrasos em tempo de execução cujas causas foram apresentadas neste texto. Contudo é importante destacar que, em relação ao modelo de execução, os atrasos que afetam a corretude do sistema são atrasos em tempo de simulação. Uma vez que existe uma relação linear entre o tempo real e o tempo de simulação, o *tardiness* em uma medida se refletirá sobre a outra.

Um problema associado à avaliação de *tardiness* é o fato que ela somente pode ser medida em tempo de execução. Outra questão é a ausência de trabalhos que relacionem a ocorrência de atrasos com parâmetros relacionados ao paradigma de agentes de software e que possam ser tratados em tempo de execução. Esta

ausência de referencial teórico levou a execução de um experimento quantitativo e a consequente análise estatística de resultados, cujos resultados fundamentam as fases seguintes da pesquisa.

As próximas seções apresentam o experimento e seus resultados.

4.2 O estudo quantitativo

A intenção deste estudo é mostrar que algumas abstrações de SMA podem estar relacionadas com o nível de *tardiness* de um sistema. A ideia geral é que, de posse da informação sobre quais parâmetros afetam a *tardiness* em tempo real, seja possível desenvolver um mecanismo de controle para o *tardiness* em tempo de simulação.

Para prospectar estas informações, um experimento estatístico foi conduzido, utilizando um grande volume de dados a fim de reforçar a validade das conclusões obtidas.

4.2.1 Planejamento do Experimento

A questão geral a ser verificada pelo experimento era verificar se variações em propriedades associadas as abstrações de sistemas multiagentes, tais como agentes e comportamentos, poderiam influenciar a variação de *tardiness* do SMA.

Para planejar o experimento, a tarefa inicial foi escolher os fatores a serem analisados, ie. as abstrações de SMA candidatas a influenciar a *tardiness*. Considerando sistemas com características de tempo real, características associadas a carga do sistema e ao ambiente no qual o mesmo esta instalado são determinantes para existência de atrasos. Como o ambiente não pode ser controlado, no escopo desta pesquisa, resta o número de instruções por unidade de tempo que o processo deve executar. Quando este número cresce, reduz-se a possibilidade de cumprirem-se os tempos de agendamento.

Observando sistemas multiagentes, algumas abstrações dos mesmos possuem relação com o número de instruções que este componente executará por unidade de tempo, e, após alguns experimentos preliminares, apresentados em (Taranti et al., 2010a), foi decidido utilizar as seguintes abstrações classificadas como fatores no experimento:

- Número de Agentes em um SMA;
- Intervalo entre duas execuções consecutivas de comportamentos de agentes que possuem natureza repetitiva; e
- A carga dos comportamentos executados pelos agentes (medida em duração de tempo de execução).

Este conjunto não pretende ser completo, ie. admite-se outras abstrações de SMA podem ser utilizadas para se estabelecer a relação com o *tardiness*. Contudo estes fatores são elementos que podem ser tratados em tempo de projeto dos SMA através do uso de arquitetura distribuída, modificações nas restrições temporais em tempo de execução e uso de comportamentos que necessitem menor duração de tempo de execução.

O experimento considerou cada fator separadamente, uma vez que a influência cruzada entre os fatores poderia variar em diferentes SMA, o que inutilizaria os resultados. A análise foi executada utilizando a Metodologia de Análise de Variância (ANOVA), apresentada em (Montgomery, 2009).

Para o experimento, a Hipótese Inicial (H_0) para cada fator foi que sua variação não possuía influência sobre o *tardiness* do sistema, ou seja:

$H_0 : \mu_1 = \mu_2 = ... = \mu_a$

A Hipótese Alternativa (H_1) para cada fator foi que sua variação possuía influência sobre o *tardiness*, ou seja:

$H_1 : \mu_1 \neq \mu_2 \neq ... \neq \mu_a$

O *output* da análise dos dados é o Nível de Significância , i.e α, e o valor de β. O significado destas variáveis é:

$\alpha = P(reject\ H_0 \mid H_0\ is\ true)$; e

$\beta = P(acept\ H_0 \mid H_0\ is\ false)$,

onde P é a função de probabilidade.

A análise dos dados foi executada com o R, que é um ambiente para análise estatística oferecido como software livre (Kabacoff, 2009; R Development Core Team, 2011). Este software é uma ferramenta com capacidade de analisar grandes volumes de dados oriundos de bancos de dados relacionais ou arquivos formatados. Uma vez que o *plugin* do R para execução do teste ANOVA fornece o valor de *Power* em vez de β, o primeiro foi utilizado no experimento, e seu valor é calculado como abaixo:

$Power = 1 - \beta = P(reject\ H_0 \mid H_0\ is\ false)$

Os níveis de corte assumidos na execução do experimento para rejeitar H_0 e aceitar H_1 foram:

$\alpha < 0.0001$; e

$Power > 0.9999$.

A plataforma MASP, apresentada no capítulo 3, foi utilizada para instanciar um modelo de execução de simulação que serviu de base para produção da massa de dados da qual seria extraída a amostra.

Do modelo de execução inicial, foram derivados 120 sistemas. Os sistemas diferem entre si pela variação dos fatores a serem avaliados, de acordo com o planejamento do experimento.

Descrição do Modelo de Execução usado no Experimento

O modelo de execução utilizado é formado por múltiplas instâncias de um agente que representa um submarino, as quais executam sobre a infraestrutura oferecida pelo MASP. Os submarinos representados pelos agentes possuem comportamentos que incluem o movimento no ambiente virtual, a capacidade de detectar outro submarino na proximidade (ie. sonar) e um comportamento de lançar torpedo.

A simulação não inclui a eliminação de agentes durante a execução, pois isto poderia afetar o valor de *tardiness* em função da redução das *thread* concorrentes. Ressalta-se que esta decisão não invalidaria os resultados para sistemas nos quais a eliminação ou criação de agentes ocorresse, pois são avaliados 20 níveis diferentes para o fator numero de agentes.

A figura 4.1 apresenta um diagrama de caso de uso simples para o submarino. A metodologia usada para modelar a simulação foi a descrita em (Nikraz et al., 2006).

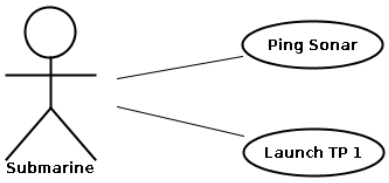

Figura 4.1: Diagrama de caso de uso

Os comportamentos foram implementados com execução periódica, o que corresponde aos sistemas reais, uma vez que o sonar emite pings em intervalos definidos e torpedos são lançados com intervalo correspondente ao ciclo de carregamento do tubo. Foi considerada a existência de um tubo por submarino.

A Tabela de Responsabilidades prevista pela metodologia de modelagem utilizada é apresentada na tabela 4.1.

Tabela 4.1: Tabela de Responsabilidades

submarineAgent	verificar se existem outros submarinos na proximidade utilizando o sonar a cada período
	lançar um torpedo contra um alvo adquirido (no máximo um torpedo por período)

Como pode ser observado, as restrições temporais foram registradas nos comportamentos e implementadas posteriormente utilizando agendadores. O MASP reimplementa mecanismos de agendamento de comportamentos disponíveis na Plataforma JADE.

O diagrama de classes da implementação utilizada é apresentada na figura 4.2. O período de execução dos comportamentos dos agentes é ajustado através do métodos de acesso a variável *SimulatedTimePeriod*, que é um atributo da classe *MaspSimulationBehavior*.

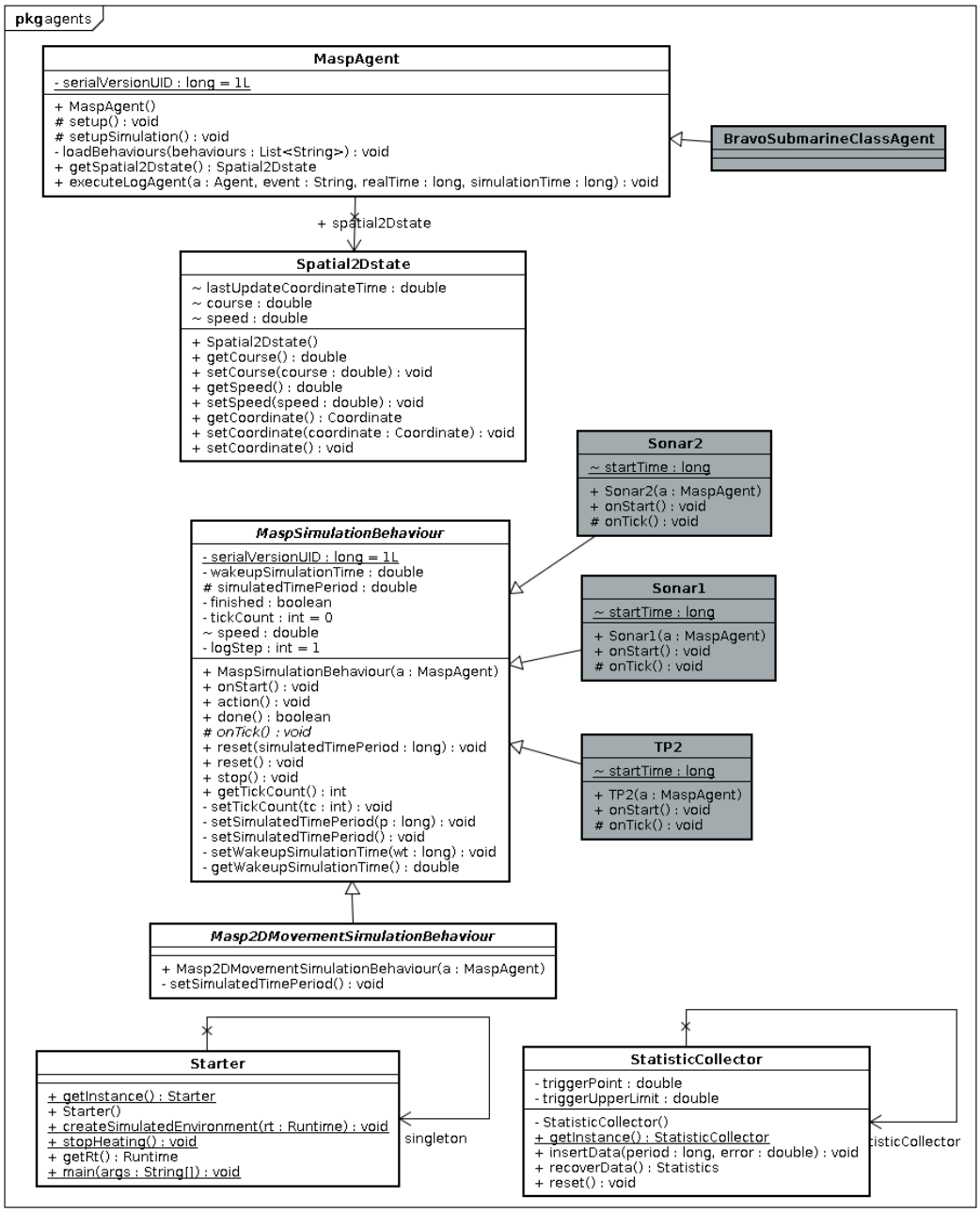

Figura 4.2: Diagrama de classe do experimento

A implementação para o combate entre os submarinos simulados foi modificado para atender as necessidades do experimento, passando os comportamentos a conter somente o código necessário para realizar o registro da informação sobre *tardiness* em cada execução. Também para evitar interferências nas medidas, foi decidido não permitir interação entre agentes, isto é, os agentes não trocaram

mensagens. Ressalta-se que as mensagens trariam mais complexidade e ruido às medições, o que dificultaria a análise isolada dos fatores. Outrossim, a ação de enviar ou receber uma mensagem nada mais é que a execução de um comportamento em nível de implementação.

As 120 instâncias do modelo de execução foram utilizadas na geração da massa de dados, sendo que a variação dos fatores considerados foi registrada em níveis discretos.

4.2.2 Metodologia

O objetivo do experimento é mostrar a existência de uma relação entre o *tardiness* do sistema em execução e os parâmetros apresentados, definidos como fatores. Para avaliar a relação, os fatores foram verificados para um conjunto de níveis, de forma individual. Abaixo se descreve a metodologia:

O primeiro passo do projeto do experimento foi definir como se mediria o *tardiness* no sistema. O *tardiness* possui uma definição genérica que é uma medida entre o tempo previsto e o executado. Após alguns testes, decidiu-se pelo uso de uma medida adimensional, calculada através da expressão:

$$Tardines = \frac{T_{executed}}{T_{programed}} - 1$$

O valor obtido indica o quanto a duração da execução é maior que o esperado. Por exemplo, se um ciclo com 2000 milissegundos de passo de execução é executado em 2400 milissegundos, então o *tardiness* será igual a 0.2, ou 20%.

Após ser definido o valor a ser medido, o passo seguinte foi formalizar os fatores para o experimento, que foram os já citados:

- número de agentes na simulação;
- intervalo entre execuções de comportamentos periódicos; e
- tempo de execução (carga) dos comportamentos.

Para gerar a massa de dados, todos os 120 modelos de execução foram executados por um período de 25 minutos, durante o qual os dados a serem analisados eram registrados para cada execução dos comportamentos individualizados dos agentes do SMA.

Para evitar ruídos não previstos, a computação dos comportamentos foi substituída por uma carga controlada de forma programática. Esta carga variava linearmente durante a execução do experimento com valor entre 0 e 200 milissegundos. A carga foi inserida conforme o código apresentado abaixo (método):

```
public void load(long startTime, long load) {
long now = System.currentTimeMillis();
long millis = load * (now - startTime) /
          ((PropertiesLoaderImpl.EXPERIMENT_DURATION_MIN) * 60000);
long a = 0;
while(System.currentTimeMillis() < (now + millis)){ a++;}
}
```

A análise do experimento foi executada para cada fator isoladamente, usando recurso de blocagem dos demais. Isto porque a influência cruzada entre fatores é dependente do ambiente de execução e implementação, não sendo viável extrapolação à outras situações. A distribuição dos sistemas instanciados pelos fatores e seus níveis é apresentada no cubo da figura 4.3.

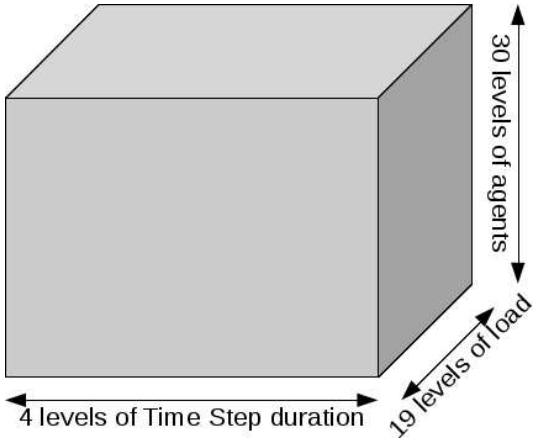

Figura 4.3: Visão gráfica da distribuição dos fatores por níveis

4.2.3 Coleta de dados

Todas as execuções dos modelos de execução ocorreram no mesmo ambiente, que consistiu de um computador com processador Intel Core i5-2400 CPU @ 3.10GHz com 8GB de memória RAM. O sistema operacional utilizado foi o Opensuse12.3 (64 bits), com JRE OpenJDK 7. Os modelos de execução foram compilados utilizando esta mesma versão do Java. As execuções foram disparadas sequencialmente usando o serviço Cron de agendamento dos sistemas operacionais Unix-like, com um intervalo de 05 (cinco) minutos entre o término de uma execução e o início da próxima. A máquina utilizada para o experimento ficou dedicada a tarefa, sem outro uso no período e estando desabilitadas as conexões de rede para evitar interferências.

Os níveis de *tardiness* observados nos primeiros segundos de cada execução se mostraram muito altos e com grandes variações, sendo que após curto período o *tardiness* reduzia e se mantinha em nível mais estável. Este comportamento geral foi

atribuído a forma como o MASP carrega o sistema: a inferência sobre a ontologia, seguida da instanciação dos agentes através de mecanismos de reflexão e injeção de dependências utiliza computação intensiva e ativa o Garbage Collector. O MASP retém o início do avanço do tempo de simulação para após a carga completa do sistema, ie. $T_s = 0$ ocorre somente após todos agentes estarem instanciados e prontos para executar.

Apesar desta variação de *tardiness* ser coerente com a carga do sistema, o objetivo da pesquisa é sobre o tempo de simulação, que somente é contado após a instanciação, sem este ruído inicial. Decidiu-se portanto desconsiderar os primeiros 80.000 milissegundos de execução na coleta de dados para todos os sistemas, após se verificar que depois deste instante o tempo de simulação já iniciava seu avanço e, consequentemente, todos agentes já haviam sido instanciados em objetos.

A massa de dados gerada foi composta de arquivos no padrão *comma-separated values* (csv), que eram alimentados em tempo de execução pelos comportamentos dos agentes. Cada SMA gerou sua própria massa de dados. Estes arquivos foram usados como fonte para criação de um banco de dados com toda informação produzida, o que foi realizado com uso do SGBD relacional PostgreSQL 9.2.

Para avaliar a possível influência dos fatores sobres a *tardiness* registrada pelos MAS, a análise dos dados foi executada com o R. O pacote Analyzes of Variance (ANOVA) deste software foi utilizado para verificar a rejeição de H_0 e a significância do teste, i.e α. O pacote *pwr* foi utilizado para se obter o valor de Power.

Uma vez que número total de medições registradas ultrapassou 83 milhões, mesmo o R utilizado em conexão com o banco de dados não foi capaz de processar todo o volume de dados disponível. Para executar a análise, então, extraiu-se da massa de dados uma amostra representativa, composta pelos dados gerados por 10 agentes representados em todos os MAS, em um total de 451.532 medidas.

O número de medidas da amostra supera em muito o necessário para assegurar-se um valor de Power superior a 99,99% em cada experimento. Esta avaliação foi executada através da função *power.anova.test* do R.

O histograma da amostra original é apresentado na figura 4.4. Cada unidade no eixo das coordenadas, que representa o nível medido de *tardiness*, representa mais 100% de atraso na execução em relação ao período agendado originalmente. Observa-se então que muitos eventos foram executados em períodos 10 vezes maior que o previsto, o que oferece uma visão do impacto que o *tardiness* pode causar em um SMA.

Para condução da análise foi necessário definir um modelo mátematico para a amostra. Após testes exploratórios, que envolveram modelos de poisson, exponencias e outros, decidiu-se pelo uso de um Modelo Linear Generalizado (GLM)

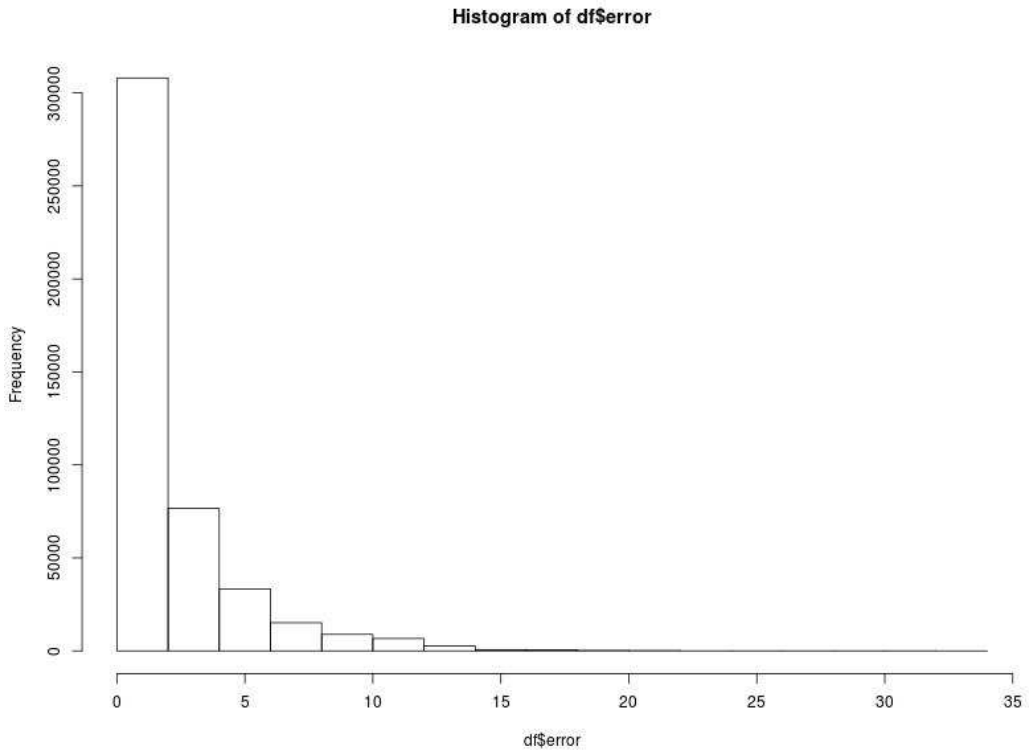

Figura 4.4: Histograma do nível de tardiness da amostra

com o modelo *quasi* fornecido pelo R. Foram utilizadas transformações para permitir a aproximação dos erros residuais da distribuição normal, que é assumida pela ANOVA em sua computação.

O Modelo final utilizado é:

$Tardiness = Y$

$Y = f(A, B, C)|A, B, C$ são os níveis dos fatores

$G = f(Y)|G = log(y) + y$

Um sumário do resultado da transformação é apresentado na tabela

O histograma do Modelo Generalizado Linear é apresentado na figura 4.5.

A análise residual foi executada utilizando o teste de Kolmogorov-Smirnov, com resultado para o valor $D = 0.0574$. Os valores dos desvios residuais são apresentados na tabela 4.2 e um gráfico com os resíduos e densidades estudentizados é apresentado na figura 4.6.

Tabela 4.2: Desvios residuais do GLM

Min	1Q	Median	3Q	Max
-15.4317	-1.3476	0.1856	1.5406	29.2115

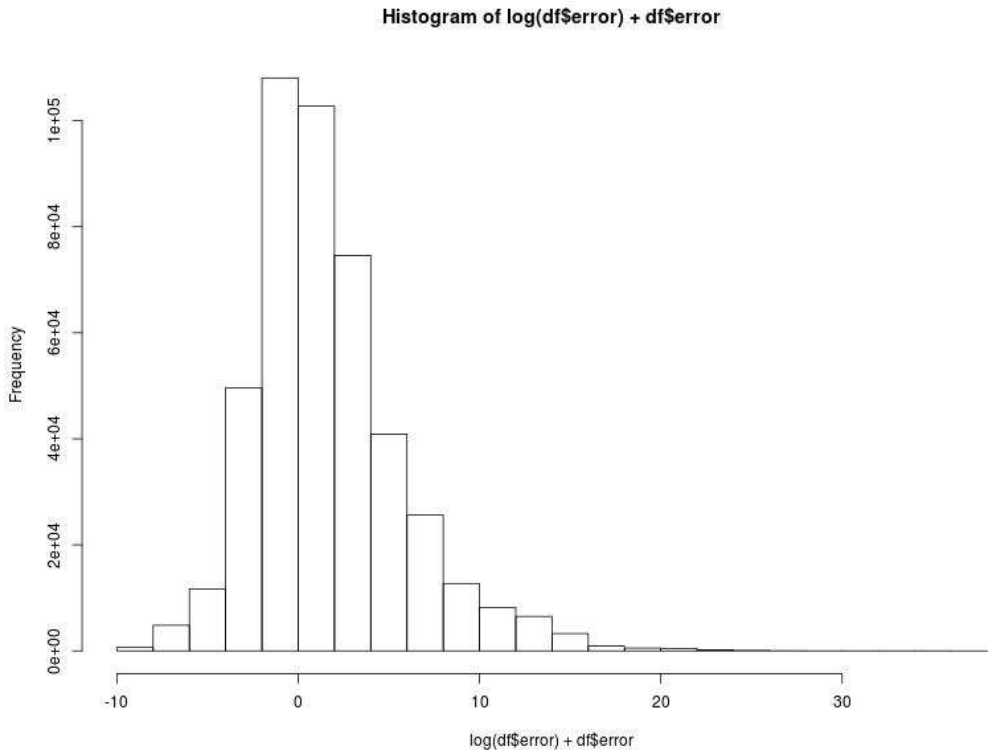

Figura 4.5: Histograma do modelo transformado

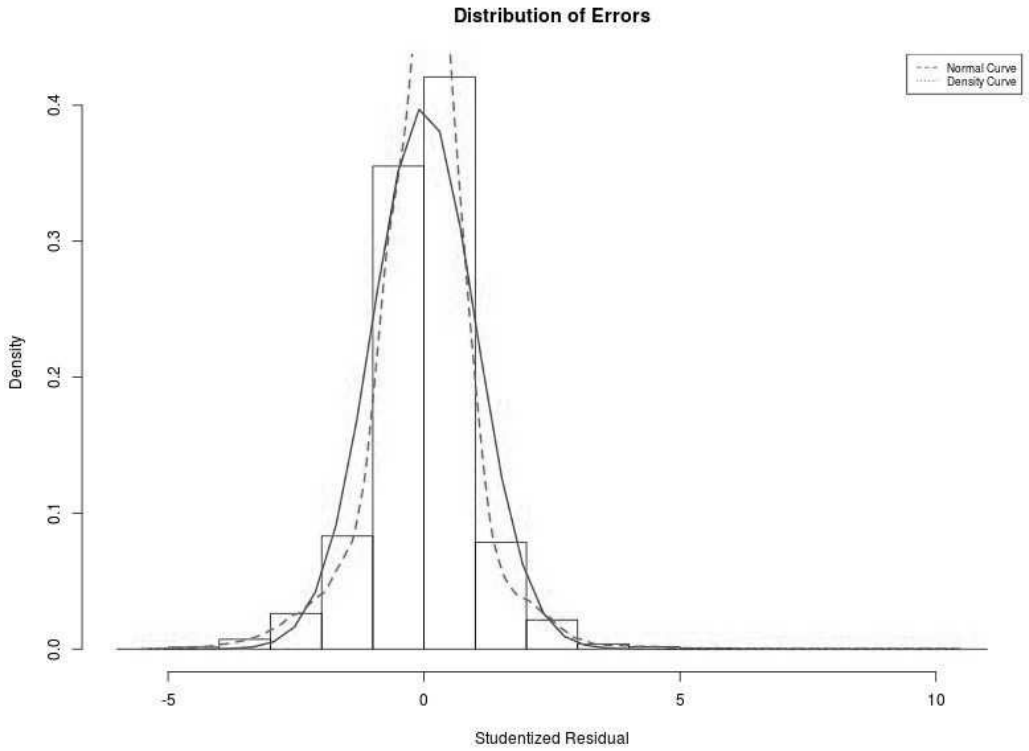

Figura 4.6: Análise gráfica dos resíduos do modelo

Um sumário do modelo transformado é apresentado na tabela 4.3. Para cada nível de fator, a tabela apresenta o valor *t-value*, que é o resultado do teste *t-test* executado sobre o nível corrente considerando o nível de referência. O t-test verifica a diferença entre os dois grupos e, para o experimento, valores absolutos maiores que 1 foram considerados como sendo uma indicação forte que os grupos eram similares em comportamento. O *p-value* é o nível de significância para cada nível testado. A indicação do *t-test* de que grupos seriam diferentes somente foi aceita para p-values (Pr) menores que 0,05.

Uma vez que o modelo foi definido, o passo seguinte foi conduzir os testes do método ANOVA. Os resultados dos testes são apresentados na tabela 4.4, e consistem principalmente dos resultados do *F-test* (F) e sua significância (Pr) .

O F-test indica que os grupos possuem a mesma média (H_0), e esta hipótese é rejeitada por valores altos de F. O experimento adotou uma abordagem conservativa e somente valores de F superiores a 4,0 poderiam ser rejeitados. O valor Pr indica o valor de α, ou seja, a significância.

Finalmente, para verificar o valor de Power (i.e, 1-β), foi usado o pacote pwr.anova. Uma vez que este pacote não permite analisar diversos fatores simultaneamente, o teste foi conduzido de forma isolada por fator. A tabela 4.5 apresenta o valor de Power para cada fator. Na tabela, K representa o número de níveis de cada fator, n é o tamanho de cada nível, F é o fator inserido pelo usuário. No experimento foram testados 4 diferentes valores, produzindo resultado similar. Enquanto a ANOVA oferece p-value, este teste oferece o valor de Power. O conjunto dos resultados indica uma probabilidade muito alta de rejeitar H_0, se H_0 é falsa.

Em seguida são apresentados resultados e comentários específicos para cada fator.

4.2.4 Análise do Primeiro Fator: número de agentes em um SMA

A hipótese (H_0) para esta análise era que o número de agentes executando em um SMA não afetaria o nível de *tardiness* do sistema. Para executar o teste, um total de 30 níveis diferentes foram usados neste fator. A amostra de dados de cada nível foi composta por 15,050 medidas. A figura 4.7 apresenta o gráfico *boxplot* com a distribuição da amostra por quartis em cada nível, que permite observar o comportamento geral da amostra.

A análise realizada com ANOVA indicou a rejeição da hipótese inicial, H_0 e, consequentemente, a aceitação da hipótese alternativa, H_1 , que afirma existir relação entre o número de agentes em um SMA e o nível de *tardiness*.

Como resultado da análise da amostra, foi possível confirmar que o número de agentes executando possem relação direta com o nível de *tardiness* no sistema.

Tabela 4.3: Summary of the transformed GLM, $G = ln(Y) + Y|Y = f(A, B, C)$

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.4057	0.0289	-14.03	0.0000
step-time (sec.)2	-3.4096	0.0114	-298.50	0.0000
step-time (sec.)3	-4.6444	0.0114	-406.65	0.0000
step-time (sec.)4	-5.3727	0.0114	-470.39	0.0000
load (x10 millisec.) 2	0.1463	0.0248	5.90	0.0000
load (x10 millisec.) 3	0.0902	0.0248	3.64	0.0003
load (x10 millisec.) 4	0.0916	0.0248	3.69	0.0002
load (x10 millisec.) 5	0.0278	0.0248	1.12	**0.2631**
load (x10 millisec.) 6	0.0990	0.0248	3.99	0.0001
load (x10 millisec.) 7	0.1632	0.0248	6.58	0.0000
load (x10 millisec.) 8	0.2577	0.0248	10.39	0.0000
load (x10 millisec.) 9	0.3378	0.0248	13.62	0.0000
load (x10 millisec.) 10	0.3167	0.0248	12.77	0.0000
load (x10 millisec.) 11	0.3872	0.0248	15.61	0.0000
load (x10 millisec.) 12	0.3811	0.0248	15.36	0.0000
load (x10 millisec.) 13	0.4456	0.0248	17.96	0.0000
load (x10 millisec.) 14	0.4518	0.0248	18.21	0.0000
load (x10 millisec.) 15	0.4265	0.0248	17.19	0.0000
load (x10 millisec.) 16	0.5239	0.0248	21.11	0.0000
load (x10 millisec.) 17	0.5210	0.0248	20.99	0.0000
load (x10 millisec.) 18	0.6026	0.0248	24.28	0.0000
load (x10 millisec.) 19	0.5034	0.0256	19.63	0.0000
number of agents 200	1.2111	0.0313	38.65	0.0000
number of agents 300	1.8436	0.0313	58.88	0.0000
number of agents 400	2.2674	0.0313	72.43	0.0000
number of agents 500	2.4633	0.0313	78.69	0.0000
number of agents 600	2.8312	0.0313	90.48	0.0000
number of agents 700	3.0603	0.0313	97.78	0.0000
number of agents 800	3.3598	0.0313	107.35	0.0000
number of agents 900	3.6506	0.0313	116.68	0.0000
number of agents 1000	3.9269	0.0313	125.49	0.0000
number of agents 1100	4.1811	0.0313	133.61	0.0000
number of agents 1200	4.3614	0.0313	139.36	0.0000
number of agents 1300	4.6065	0.0313	147.17	0.0000
number of agents 1400	4.8833	0.0313	156.00	0.0000
number of agents 1500	5.1804	0.0313	165.52	0.0000
number of agents 1600	5.4503	0.0313	174.13	0.0000
number of agents 1700	5.7342	0.0313	183.13	0.0000
number of agents 1800	5.8919	0.0313	188.10	0.0000
number of agents 1900	6.2057	0.0313	198.01	0.0000
number of agents 2000	6.4932	0.0313	207.15	0.0000
number of agents 2100	6.5890	0.0314	210.14	0.0000
number of agents 2200	6.8726	0.0314	218.95	0.0000
number of agents 2300	7.0362	0.0314	224.03	0.0000
number of agents 2400	7.3054	0.0314	232.72	0.0000
number of agents 2500	7.4694	0.0314	237.72	0.0000
number of agents 2600	7.7057	0.0314	245.15	0.0000
number of agents 2700	8.0475	0.0315	255.86	0.0000
number of agents 2800	7.8130	0.0315	248.40	0.0000
number of agents 2900	8.3570	0.0315	265.60	0.0000
number of agents 3000	8.4097	0.0316	265.72	0.0000

Tabela 4.4: Analise de Variância. Model: quasi, link: identity

	Df	Deviance	Resid. Df	Resid. Dev	F	Pr(>F)
NULL			451531	7524133.79		
step-time (sec.)	1	1692492.24	451530	5831641.55	212992.34	0.0000
load (x10 millisec.)	1	9601.14	451529	5822040.41	1208.26	0.0000
number of agents	1	2234082.05	451528	3587958.36	281148.92	0.0000

Tabela 4.5: Análise de *Power* para o teste ANOVA

	number of agents	load (x10 millisec.)	step-time (sec.)
K =	30	19	4
n =	15,050	23,764	112,880
F =	0.2, 0.4, 0.8, 1.0	0.2, 0.4, 0.8, 1.0	0.2, 0.4, 0.8, 1.0
p-value (α)	2.2e-16	2.2e-16	2.2e-16
Power (1-β)	1	1	1

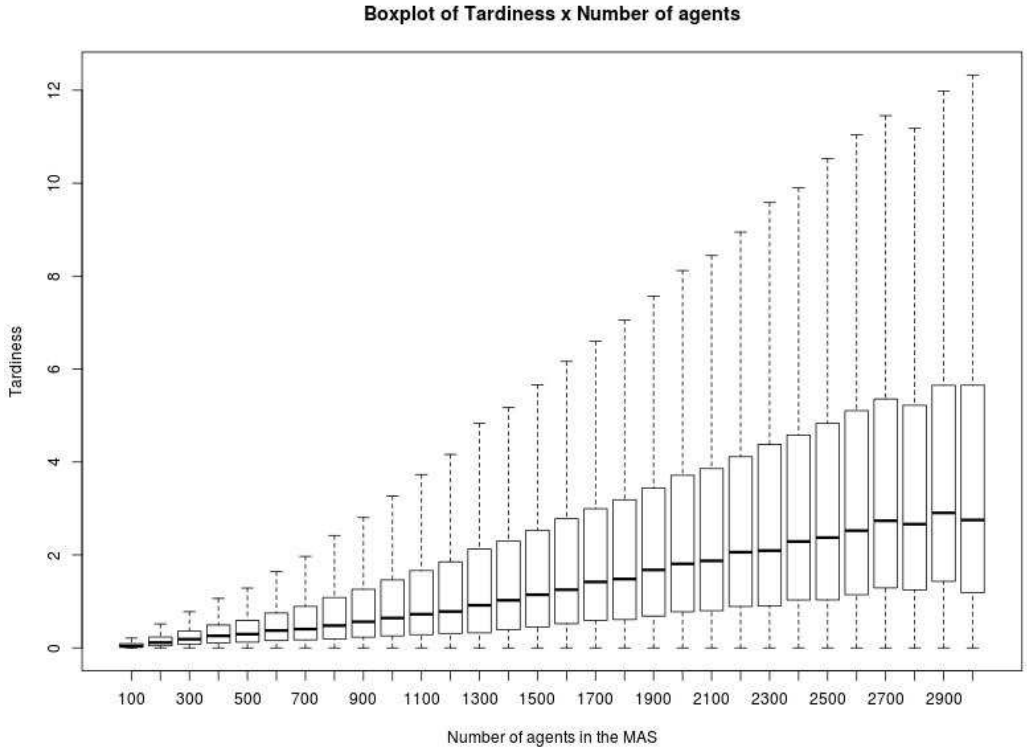

Figura 4.7: Primeiro fator – nível de tardiness x número de agentes (boxplot)

O gráfico *boxplot* permite inferir que a relação.

Contudo é importante destacar que o mesmo gráfico e os resultados do *t-test* indicam que a influência é reduzida para número de agentes nos níveis mais baixos. Este dado se confirmou em outros experimentos realizados.

Concluindo a análise deste primeiro fator, para pequenos números de agentes a influência do fator é pequena, contudo existe um valor crítico a partir do qual esta

influência torna-se maior. Porém, este ponto deve ser verificado individualmente para cada sistema em tempo de execução, considerando o ambiente no qual este estará instalado.

Este número, cujo cálculo não é possível em tempo de projeto por ser dependente de ambiente, poderia ser der verificado após a entrega do sistema para, por exemplo, ser considerado um valor limitante para o número de agentes em uma simulação, a fim de minimizar a variação de *tardiness* no sistema.

4.2.5 Análise do Segundo Fator: intervalo de tempo entre execuções

O segundo fator analisado possuía como hipótese (H_0) que o intervalo entre duas execuções periódicas de comportamentos de agentes em um SMA não deveriam afetar o *tardiness* do sistema. Para executar o teste, um total de 3 diferentes níveis com intervalos diferentes foram utilizados, com 112,880 medições para cada nível.

Os valores dos níveis foram 1.000, 2.000, 3.000 e 4.000 milissegundos como intervalo de tempo entre as execuções. O gráfico apresentado na figura 4.8 ilustra o comportamento da amostra.

A análise de variância foi executada através do teste ANOVA com o R. Como resultado desta análise, a hipótese inicial H_0 foi rejeitada e a hipótese alternativa H_1 foi aceita. Isto significa que o intervalo entre as execuções dos comportamentos possui influência sobre o nível de *tardines* do sistema.

Como resultado na análise de dados, é possível verificar que o intervalo de tempo entre execuções de comportamentos periódicos possui uma relação inversa com o aumento de *tardiness* do sistema. Esta relação pode ser verificada no *boxplot* apresentado.

Apesar do número de níveis selecionado ser baixo, os testes estatísticos indicam que os resultados são consistentes, sendo este o fator de maior influência no experimento, a considerar o valor do *F-test*.

Uma questão importante é que este fator permite realizar intervenção de modo menos complexo que os demais, pois se trata de um único parâmetro no código do comportamento.

4.2.6 Análise do terceiro fator: tempo necessário para execução de comportamentos (carga)

O terceiro fator analisado tinha como hipótese inicial (H_0) a inexistência de uma relação entre a carga, ou tempo de computação dos comportamentos e o nível de *tardiness* do sistema. Para executar o teste foram estabelecidos 19 níveis

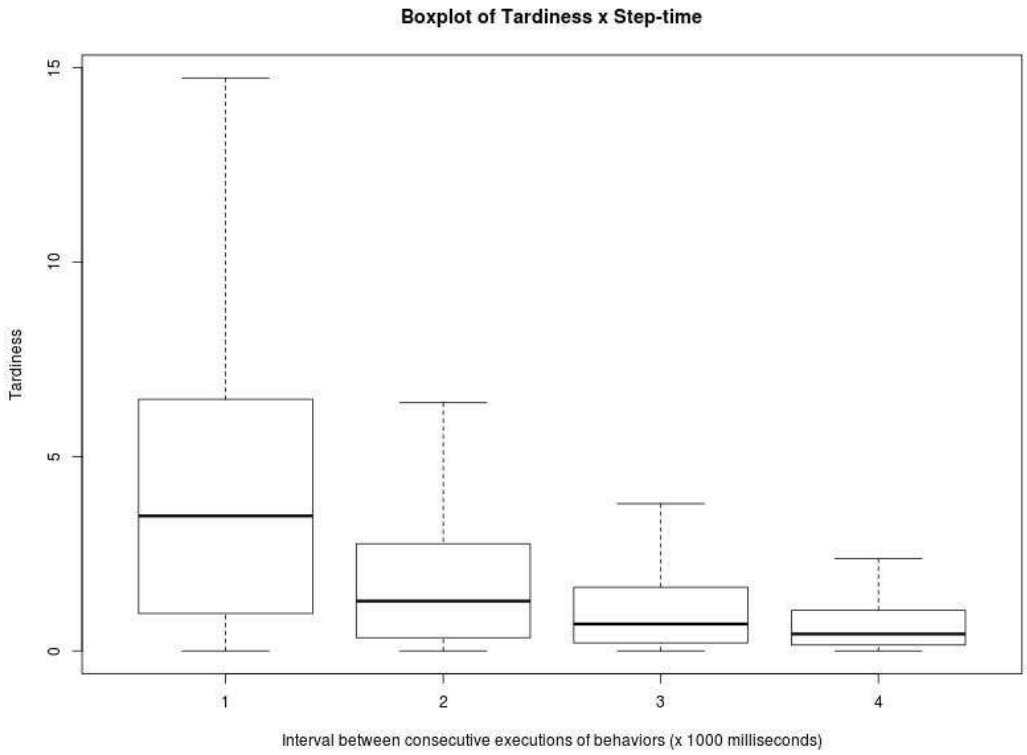

Figura 4.8: Segundo fator – nível de tardiness x intervalo de execução (boxplot)

diferentes de carga, com 23,764 medições em cada nível. O gráfico de *boxplot* apresentado na figura 4.9 ilustra o comportamento dos sistemas.

A análise gráfica não é conclusiva, pois o boxplot não apresenta uma clara influência do fator. Contudo, a análise de variância realizada com o teste ANOVA permitiu rejeitar a hipótese inicial (H_0). Como consequência, a hipótese alternativa (H_1) foi aceita, o que implica que a carga dos comportamentos individuais possui influência sobre o nível de *tardiness*.

É importante destacar que este fator foi o de menor influência sobre o *tardiness* para este experimento, conforme resultados do F-test. Contudo a significância do teste e o valor de *Power* são fortes e confirmam a avaliação. Uma possibilidade a ser considerada é que a carga de teste escolhida, variando de 0 a 200 milissegundos, tenha sido de valor baixo em relação ao sistema avaliado, e por isso não permitiu uma análise gráfica conclusiva. Em outros SMA, nos quais a computação dos comportamentos seja exija tempo maior de processador, deve-se esperar que a influência seja maior.

Outro ponto a se observar é a dificuldade em se intervir neste parâmetro em sistemas: considerando que o design do sistema seja bem feito, não é tarefa trivial tentar reduzir tempo de processamento sem modificar o ambiente de execução (ex. processador e memória).

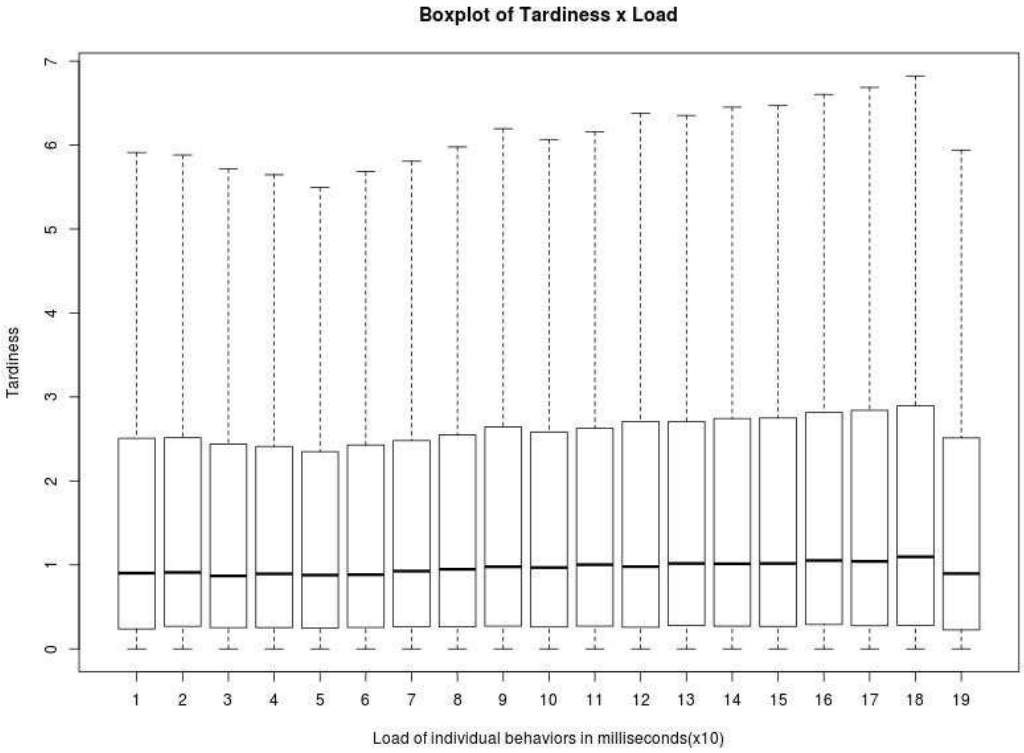

Figura 4.9: Terceiro fator – nível de tardiness x carga dos comportamentos

4.2.7 Observações gerais sobre os resultados

Considerando as relações apresentadas nas subseções anteriores, é possível concluir que o SMA poderia ser planejado com uma arquitetura que permitisse *tunning* em tempo de execução.

Por exemplo, é possível considerar uma arquitetura distribuída para permitir a migração de agentes em tempo de execução e reduzir o *tardiness* em um *host* específico. O intervalo de tempo entre execuções poderia ser controlado em tempo de execução, a fim de reduzir o *tardiness*. A carga de cada comportamento poderia ser estudada em tempo de projeto, para avaliar a pertinência do uso de técnicas de concorrência aplicadas aos comportamentos para evitar a construção de métodos que exijam muitos recursos em tempo de execução.

O fato do experimento considerar apenas uma implementação como base pode vir a ser questionado. O motivo para esta decisão foi evitar o aumento de complexidade das variáveis a um nível que não fosse possível tratar, pela dificuldade que existiria em isolar os ruídos. Os fatores selecionados foram cuidadosamente selecionados na fase preliminar, cujos resultados auxiliaram a compor as hipóteses iniciais.

Outra implementação de SMA, se usada como base para o experimento, possivelmente apresentaria valores diferentes de *tardiness* para os mesmos níveis

dos fatores, contudo a influência verificadas para os fatores sobre *tardiness* podem ser generalizadas para todas as implementações de SMA baseadas em Java, pois estas partilham o mecanismo de agendamento de *threads*.

4.3 Consequências do tardiness e mitigações

A observação dos gráficos *boxplot* e a analise numérica não permitem afirmar a existência de uma relação linear ente o *tardiness* e os fatores escolhidos, conquanto seja verificada a influência destes. O experimento não considerou as influências cruzadas entre fatores, que provavelmente existem. Os resultados não permitem prever o valor de *tardiness*, ao contrário.

Como já descrito, todas as medições do experimento foram realizadas no mesmo ambiente, de forma controlada, para evitar ruídos externos que afetassem as medidas. Isto porque o nível de *tardiness* pode ser afetado por fatores ambientais, tais como conexões de rede, hardware, sistema operacional, implementação de máquina virtual e memória. Estes fatores têm influência sobre a latência associada ao hardware do computador, infraestrutura de rede e sistema operacional. Adicionalmente, a tamanho da *heap* de memória possui influência sobre o comportamento do GC.

Para verificar a influência do ambiente, outros experimentos foram executados em hardware diferente, porém com o mesma versão do sistema operacional e JRE. Estas experiências apresentaram níveis diferentes de *tardiness*, indicando a existência da relação com o ambiente. Com isto, qualquer solução para controle de *tardiness* deve ser sensível ao ambiente também.

Sistemas de tempo real classificados como *firm real–time* ou *soft real–time* admitem níveis de *tardiness* em relação as tarefas agendadas, porém atrasos superiores a estes níveis podem comprometer a corretude dos sistemas.

Por exemplo, o atraso de um comportamento que deva apagar registros antigos possivelmente não será um erro crítico. Contudo, atrasos no fechamento da contabilidade de um sistema financeiro provavelmente será um erro crítico. O mesmo pode ser dito de atrasos no avanço do tempo de simulação em uma *engine* de simulação para SMA.

Então o nível de *tardiness* necessita ser monitorada em tempo de execução para garantir que a execução do sistema gera uma saída correta. O que sugere que abordagens que monitoração e controle de *tardiness* são necessárias.

Outra questão importante é que o nível de *tardiness* crítico ao sistema deveria ser estabelecido em tempo de projeto, quando a avaliação de riscos é normalmente executada, seguindo-se algum modelo de desenvolvimento. Porém as atuais metodologias de MAS não consideram a atividade de avaliação de risco.

A avaliação de risco do SMA sendo projetado deveria incluir informação sobre agentes que possuem restrições temporais, nível crítico de *tardiness* e quais ações o projeto deveria considerar para mitigar o risco.

Um exemplo de ação de mitigação que é apoiada pelo experimento apresentado neste capítulo é o projeto de intervalos de tempo entre execuções que possuam flexibilidade para serem alterados em tempo de execução, caso a corretude do software permita esta ação.

Contudo, para monitorar o nível de *tardiness* em um sistema é necessário realizar testes não funcionais em tempo de execução, que permitam detectar o aumento do *tardiness*. Estes testes devem ser realizados de forma automática e frequente. Mecanismos de coleta e avaliação de dados são necessários, além da capacidade de intervir no sistema, evitando sua degradação.

Estes testes são uma atividade *ad-hoc*, uma vez que não se verificam ferramentas ou teoria que os apoiem. Os trabalhos verificados que possuíam objetivo em testes de SMA abordavam aspectos funcionais destes sistemas (Nguyen et al., 2009; Coelho et al., 2006; Rodrigues et al., 2005). Não se localizou na literatura um trabalhos que discutissem testes não–funcionais para SMA dedicados especificamente para restrições temporais.

Uma abordagem possível, utilizada na etapa seguinte deste trabalho, é a inclusão de assertivas no código dos agentes e dos comportamentos. Estas assertivas, por sua vez, podem iniciar rotinas que tenham por fim estabilizar o sistema e reduzir o *tardiness* do mesmo.

A instrumentação do código pode ser realizada de forma manual ou semiautomática. A JML (Leavens et al., 2009) é um exemplo de trabalho que disponibiliza um suporte consistente para instrumentação de código através de assertivas. Contudo o uso da JML exige o uso de um framework próprio e suporte a ambiente de execução, o que pode limitar o uso futuro em caso de descontinuidade do projeto que a desenvolve. Outra solução é o uso do suporte nativo de Java a assertivas (Mahmoud, 2005), ou mesmo a implementação destas de forma ad-hoc.

A seguir, apresenta-se um exemplo de assertiva para detecção de *tardiness* superior a 0,05, ou seja, de atraso superior a 5% em relação ao tempo esperado:

```
Public class Verify Send extends Misbehavior {
.....................
 private void onTick() {

 if(nowTime-lastTime > period && nowTime-lastTime < period*1.05){
 System.out.println("agent "+ a.getLocalName()+
 " is executing with delay greater than 5%");
 System.out.println("The systen is being shuting down -" +
```

```
 " an inconsistent state was achieved");
 System.exit(1);
 }
 }
 ..................
}
```

4.4 Conclusão do Capítulo

Este capítulo apresentou segunda fase da pesquisa, que buscava criar uma base de conhecimento consistente para os passos seguintes, cobrindo a atual ausência de trabalhos sobre o problema da relação entre atrasos em tempo de execução e as abstrações de agentes. O experimento quantitativo estabeleceu relação entre a variação dos três fatores e o *tardiness* do SMA.

A análise dos resultados permitiu inferir a necessidade de testes não funcionais em tempo de execução, bem como atuadores que sejam controlados pela informação gerada por estes testes (ex. gatilhos). Também se verificou a necessidade de estabelecer níveis críticos de *tardiness* para os sistemas em tempo de projeto, pois estes níveis devem ser considerados na construção dos testes não funcionais.

As ações de mitigação apresentadas em 4.3 não pretendem atender as necessidades de suporte a questão de controle de *tardiness* em SMA, pois esta parte da pesquisa foi encerrada quando o suporte à fase seguinte foi avaliada como suficiente. Abordagens mais robustas, metodologias e ferramental são necessários para tratar o problema, que deve ser considerado um tema de pesquisa em aberto. A necessidade de melhor suporte as questões de tempo real para desenvolvimento de SMA foi também discutida em (Dignum, 2011), e deve ser considerada um problema em aberto para pesquisas futuras.

Uma outra possibilidade para a análise estatística da questão, e que se mostra promissora no escopo, é a modelagem utilizando Cópulas (Lai & Balakrishnan, 2009). A abordagem não foi explorada por limitações de recursos.

O próximo capítulo apresenta a solução da principal questão da pesquisa e cujos resultados foram viabilizados pelos trabalhos apresentados no capítulo 3 e neste.