UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM AUTOMAÇÃO E SISTEMAS

Luis Pedro Arenhart Lampert

Tolerância a Faltas em Sistemas Multiagentes Multidimensionais

Florianópolis
2021

Luis Pedro Arenhart Lampert

Tolerância a Faltas em Sistemas Multiagentes Multidimensionais

Dissertação submetida ao Programa de Pós-Graduação em Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de Mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Jomi Fred Hübner, Dr.
Coorientador: Prof. Maicon Rafael Zatelli, Dr.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Lampert, Luis Pedro Arenhart
Tolerância a faltas em sistemas multiagentes multidimensionais / Luis Pedro Arenhart Lampert ; orientador, Jomi Fred Hübner, coorientador, Maicon Rafael Zatelli, 2021.
71 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Tolerância a faltas. 3. Sistemas Multiagentes. 4. Dependabilidade. 5. Programação Orientada a Sistemas Multiagentes. I. Hübner, Jomi Fred . II. Zatelli, Maicon Rafael. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Luis Pedro Arenhart Lampert

Tolerância a Faltas em
Sistemas Multiagentes Multidimensionais

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof.(a) Cesar Augusto Tacla, Dr(a).
Instituição UTFPR/Curitiba

Prof.(a) Marcelo Lopes de Lima, Dr(a).
Instituição CENPES/Petrobras

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 20/09/2021 15:45:28-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Jomi Fred Hubner
Data: 20/09/2021 10:40:07-0300
CPF: 596.334.769-68
Verifique as assinaturas em https://v.ufsc.br

Prof. Jomi Fred Hübner, Dr.
Orientador

Florianópolis, 2021.

AGRADECIMENTOS

Agradeço primeiramente a minha eterna companheira Daniela, que esteve presente desde o primeiro dia me ajudando de todas as maneiras possíveis. Aos pais Valdir e Lizete e aos sogros Ricardo e Marlene pela incentivo e torcida a distância.

Ao Prof. Jomi por sua exímia e amigável orientação neste trabalho e ao coorientador Prof. Maicon pela disposição constante e excelentes contribuições.

Aos amigos e colegas de mestrado Otávio, Alisson e Leandro pelas diversas horas gastas trocando ideias, experiências e apoio. Aos irmão de vida Marcos, Juliano e Raul pelo incentivo a distância e por sempre acreditar na minha capacidade.

Por último, um agradecimento especial Centro de Pesquisas e Desenvolvimento Leopoldo Américo Miguez de Mello (CENPES) pelo apoio financeiro durante o desenvolvimento do projeto de pesquisa *Sistemas e Algoritmos de Controle Multiagentes*, que deu origem ao tema de pesquisa dessa dissertação. Aos membros do CENPES participantes do projeto, um agradecimento pelas amplas discussões sobre os temas pertinentes a essa pesquisa.

“If in doubt, paddle out.”
(Nat Young)

RESUMO

Em um sistema multiagentes (SMA), diversos agentes autônomos interagem entre si e com o ambiente em que se encontram. Essa natureza do SMA possibilita a criação de sistemas bastante complexos e dinâmicos. No entanto, essas características também podem gerar cenários imprevisíveis, o que contribui para o surgimento de faltas no sistema e prejudica a confiabilidade dos SMA, reduzindo a sua utilização em aplicações que demandam soluções robustas. Para melhorar o nível da confiabilidade de um sistema, diversas técnicas podem ser empregadas. Entre elas, a tolerância a faltas (TF) visa garantir que o sistema entregue os serviços esperados, mesmo que faltas tenham ocorrido. Existem diversos trabalhos que abordam a TF nos SMA, entretanto, a maioria faz um tratamento não genérico. Além disso, os modelos de TF propostos nesses trabalhos se concentram na programação da dimensão dos agentes. Este trabalho elabora e implementa um modelo de TF adaptado aos SMA, explorando os conceitos existentes na programação de SMA multidimensionais (agente e ambiente). O modelo utiliza analogias de alto nível presentes nas abstrações da dimensão dos agentes e do ambiente para fornecer serviços de detecção e recuperação de erros nos agentes, além de outras funcionalidades para o desenvolvedor do SMA elevar o nível de confiabilidade do seu sistema. Utilizando diversos cenários dinâmicos e com alta ocorrência de faltas, o modelo de TF é testado e o comportamento do SMA é avaliado, provando a capacidade do modelo em detectar erros e recuperar os agentes do SMA, permitindo que o sistema tolere as faltas ocorridas.

Palavras-chave: Tolerância a Faltas. Sistemas Multiagentes. Dependabilidade. Programação Orientada a Sistemas Multiagentes.

ABSTRACT

In a multi-agent system (MAS), several autonomous agents interact with each other and with the environment where they are placed. This nature of MAS makes it possible to create very complex and dynamic systems. However, these characteristics can also generate unpredictable scenarios, which contributes to the appearance of faults in the system and impairs the reliability of the MAS, reducing its use in applications that demand robust solutions. To improve the level of reliability of a system, several techniques can be employed. Among them, fault tolerance (FT) aims to ensure that the system delivers the expected services, even if faults have occurred. There are several studies that address FT in MAS, however, most do a non-generic treatment. Furthermore, the FT models proposed in these works focus on agent dimension programming. This study elaborates and implements a FT model adapted to MAS, exploring the existing concepts in multidimensional MAS programming (agent and environment). The model uses high-level analogies present in the agent and environment dimension abstractions to provide agent error detection and recovery services, in addition to other functionalities for the MAS developer to increase the reliability level of their system. Using several dynamic scenarios with high occurrence of faults, the FT model is tested and the MAS behavior is evaluated, proving the model's ability to detect errors and recover the MAS agents, allowing the system to tolerate the faults that have occurred.

Keywords: Fault Tolerance. Multi-agent System. Dependability. Multi-agent Oriented Programming

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A presença massiva de sistemas computacionais nos setores econômicos e no cotidiano das pessoas é uma realidade que se consolida cada vez mais. O funcionamento básico da economia global é altamente dependente do uso e da confiança que se deposita em sistemas baseados em computadores. Pode-se afirmar que, cada vez mais, softwares com operações seguras e confiáveis são requerimentos significativos para a maioria dos sistemas existentes. O custo e as consequências do não tratamento de faltas nesses sistemas podem variar de suáveis incomodações até catástrofes com prejuízos sérios, como violações de segurança, perda de negócios e, nos casos mais extremos, perdas de vidas humanas. Nesse cenário, a *tolerância a faltas* (TF) surge como um meio para que o sistema consiga entregar os serviços que originalmente foram projetados, apesar da ocorrência de faltas durante sua execução (AVIZIENIS *et al.*, 2004; LAPRIE, 1992; PULLUM, 2001).

Paralelo a isso, outro aspecto observado nos sistemas computacionais é o aumento drástico que esses sistemas vêm recebendo em complexidade, o que estimula o emprego de paradigmas de computação que tenham foco em processos distribuídos e concorrentes. Essa tendência da computação distribuída, por sua vez, impulsiona o desenvolvimento de linguagens de programação que utilizam mecanismos e abstrações adequadas para implementação desse tipo de software. Entre as linguagens de programação, existe um interesse crescente por aquelas que abordam o desenvolvimento dos *sistemas multiagentes* (SMA). Em suma, os SMA são sistemas cuja arquitetura é composta por um série de entidades autônomas, denominadas agentes, que possuem capacidades computacionais e que, na maioria das vezes, interagem entre si (DÍAZ, 2018).

A TF é um tópico estudado há bastante tempo, tendo uma base comum de conhecimento que é amplamente compartilhada, o que facilita as especificações de sistemas, implementações de técnicas e comparações entre diferentes abordagens. Entretanto, quando se fala em TF aplicada aos SMA, percebe-se uma falta de generalidade nas abordagens, pois muitas delas focam em aspectos específicos do problema, tratando apenas algumas faltas que ocorrem em condições já conhecidas e tem maior impacto no sistema. Além disso, verifica-se que a natureza autônoma e proativa dos agentes podem ser fontes de faltas para o sistema. Isso se deve ao fato de que, em um SMA, vários agentes podem operar em ambientes compartilhados, a natureza autônoma e proativa dos agentes acaba tornando esses ambientes bastante dinâmicos e imprevisíveis, estabelecendo um cenário propenso a ocorrência de faltas, o que é indesejável em qualquer tipo de sistema. Somado a isso, a interação entre os agentes pode fazer com que um pequeno erro possa se propagar por todo o sistema, causando uma falha generalizada (FEDORUK; DETERS, 2002; POTIRON; SEGHROUCHNI;

TAILLIBERT, 2013; STANKOVIĆ; ŠTULA; MARAS, 2017).

Devido a essa predisposição dos SMA para ocorrência de faltas e a carência de abordagens de TF mais genéricas, o estudo de técnicas e conceitos de TF que sejam mais amplos e adaptados aos SMA mostra-se importante para o aprimoramento da confiabilidade desses sistemas, permitindo que eles, cada vez mais, sejam utilizados em aplicações que necessitam de uma maior robustez.

A fim de contribuir para o aumento da confiabilidade dos SMA, o presente trabalho apresenta um modelo de TF adaptado aos SMA que utilizam as abstrações das multidimensões, apresentadas no trabalho de Boissier *et al.* (2013). Neste tipo de abordagem considera-se a existência de três dimensões que servem como ferramenta para programação de SMA complexos: a *dimensão dos agentes*, onde os agentes interagem; a *dimensão do ambiente*, onde são desenvolvidos os recursos comuns aos agentes e conexões com o mundo real; e a *dimensão da organização*, que estrutura e regulamenta as relações dos agentes entre si e com o ambiente compartilhado. Até onde se investigou, não foi encontrado nenhum trabalho onde tenha sido proposto um modelo de TF em SMA que considerasse outras dimensões além da dos agentes.

Esse trabalho apresenta os mecanismos tradicionais de TF, explorando de que maneira os conceitos e abstrações de alto nível, fornecidas pela abordagem multidimensional, podem colaborar para implementação de um modelo de TF que contribua para o aumento da confiabilidade do SMA como um todo.

O modelo de TF elaborado neste trabalho provêm três funcionalidades principais ao SMA: (i) fornecimento do serviço de TF utilizando mecanismos de detecção e recuperação de agentes em estado de erro; (ii) criação da relação de monitoramento por interesse entre agentes, que possibilita que agentes do SMA monitorem outros agentes com os quais eles tenham algum tipo de vínculo de interesse; e (iii) fornecimento de um serviço de monitoramento contínuo da saúde do agente, que possibilita ao desenvolvedor do sistema acompanhar em tempo real a evolução de indicadores de saúde do agente monitorado, além de possibilitar a criação de planos personalizados ativados por eventos de interesse.

1.1 OBJETIVOS

O objetivo dessa dissertação é a elaboração e implementação de um modelo de TF aplicado aos SMA utilizando as abstrações das multidimensões. Esse modelo deve fornecer serviços de TF e favorecer o aumento da confiabilidade do SMA. Os objetivos específicos do trabalhos são:

- Desenvolver um modelo de TF em SMA que utilize abstrações além da dimensão dos agentes;
- Implementar o modelo proposto em uma plataforma de SMA adaptada a

programação multidimensional;

- Avaliar o modelo conforme métricas estabelecidas.

1.2 ESTRUTURA DO DOCUMENTO

A estrutura do trabalho é dividida em fases de pesquisa, implementação e experimentação. No Capítulo 2 são expostos as revisões da literatura pertinentes ao trabalho. A Seção 2.1 apresenta um breve referencial sobre os SMA, em especial os conceitos pertinentes aos SMA multidimensionais. Os conceitos mais genéricos sobre dependabilidade de sistemas computacionais são apresentados na Seção 2.2, que depois são aprofundados para as técnicas de TF na Seção 2.3. Na Seção 2.4 a TF é contextualizada na área dos SMA e, por fim, na Seção 2.5 são apresentados e comentados alguns trabalhos que serviram de inspiração para a elaboração do modelo de TF.

No Capítulo 3 o modelo de TF é apresentado, suas principais entidades e funcionalidades são detalhadas, passando por aspectos gerais, exemplos básicos e também maiores detalhes da implementação do modelo na linguagem JaCaMo. No Capítulo 4 são expostos diversos cenários de testes pelos quais os mecanismos de TF que o modelo implementa são avaliados. Para cada cenário de teste são apresentados e discutidos os resultados obtidos. Por fim, no Capítulo 5 são apresentadas as discussões finais sobre os resultados da dissertação, bem como são feitas algumas propostas para sequência do trabalho.

2 REVISÃO DA LITERATURA

Esta dissertação apresenta dois temas como principais pontos para desenvolvimento do trabalho: os *Sistemas Multiagentes* e a *Tolerância a Faltas*. Ao longo deste capítulo serão apresentados um referencial teórico sobre esses assuntos. Inicialmente o conceito de sistemas multiagentes será apresentado, identificando algumas ferramentas de abstrações utilizadas por esse tipo de sistema. Após isso será apresentado uma noção sobre a dependabilidade em sistemas de software e quais os principais atributos e termos que essa área engloba. A TF será apresentada de forma genérica e aplicada aos SMA. Por fim, alguns trabalhos relacionados a esta dissertação serão apresentados e comentados.

2.1 SISTEMAS MULTIAGENTES

Os SMA são uma subárea da inteligência artificial que oferece um grande repertório de ferramentas, técnicas e metáforas que auxiliam a análise, o projeto e a implementação de sistemas de softwares complexos. A utilização deste tipo de sistema é adequada para problemas onde múltiplas entidades utilizam múltiplos métodos para resolução de tarefas. A natureza distribuída e concorrente dos SMA é uma das tradicionais vantagens desse tipo de sistema (JENNINGS; SYCARA; WOOLDRIDGE, 1998).

Um dos conceitos utilizados como suporte para a programação de SMA são as abstrações de dimensões. Nesta proposta, considera-se a existência de três dimensões interligadas: a *dimensão dos agentes*; a *dimensão do ambiente*; e a *dimensão da organização*. Cada uma das dimensões será melhor aprofundada nas seções subsequentes.

2.1.1 Dimensão dos agentes

Essa dimensão reúne as abstrações que são usadas para programar agentes que, segundo (WOOLDRIDGE; JENNINGS, 1995), são entidades de software que possuem propriedades específicas, como: *autonomia*, que permite que a entidade opere sem intervenção humana, tendo um certo controle sobre suas ações e estados internos; *habilidade social*, que é a capacidade de interagir e de se comunicar; *reatividade*, que possibilita que os agentes percebam o ambiente em que estão inseridos e reajam às suas mudanças; e *proatividade*, que faz com que as ações possam ser tomadas por iniciativa própria, exibindo uma direção orientada à objetivos.

Além disso, alguns modelos de agente ainda adicionam conceitos usualmente utilizados para descrever o comportamento humano, como as noções mentais. Neste trabalho adota-se o modelo *Belifs-Desires-Intentions* (BDI), inspirado nos modelos de

comportamento humano da filosofia que estuda o raciocínio prático, que é o processo de decidir, momento a momento, quais ações devem ser realizadas para atingir os objetivos (WOOLDRIDGE, 1999). Esse conceito insere a ideia de que programas de computador podem ter um estado mental. Segundo Bordini, Hübner e Wooldridge (2007), a distinção entre essas noções mentais se dá da seguinte maneira:

- Crenças (*Beliefs*): são as informações que o agente possui sobre o mundo, sobre ele mesmo e sobre os demais agentes.
- Desejos (*Desires*): são os estados que o agente pode desejar alcançar. Ter um desejo, contudo, não necessariamente significa que o agente tomará ações para atingir esse desejo. É perfeitamente possível que agentes tenham dois desejos conflitantes. Os desejos podem comumente ser interpretados como sendo "opções" para um agente.
- Intenções (*Intentions*): são os estados que o agente decide trabalhar para atingir. As intenções podem ser delegadas ao agente, ou podem ser resultado de uma escolha feita por ele, após analise de suas opções.

Os agentes constantemente percebem e agem sobre o ambiente em que estão inseridos, interagem com outros agentes e assumem papéis em entidades organizacionais. Para exibir a propriedade da autonomia, os agentes são permitidos a refletir sobre quais são os objetivos que desejam atingir e como devem fazer isso, através de seus *planos*, dados as suas crenças atuais sobre o mundo. Os planos definem uma sequencia de ações e manipulação de crenças. Essas alterações nas crenças geram novos eventos e são esses eventos que ocasionam a execução de outros planos, que novamente dependem das crenças, do ambiente e de outras intenções. São essas abstrações que permitem aos agentes serem proativos, pois eles agem para atingir objetivos, e também reativos, pois eles reagem a alterações no ambiente, organização e outros agentes (BOISSIER *et al.*, 2019).

2.1.2 Dimensão do ambiente

Por sua vez, a dimensão do ambiente reúne uma série de abstrações que representam o ambiente compartilhado pelos agentes. A principal delas é a noção de um *artefato*, que é uma entidade básica que encapsula recursos e ferramentas computacionais que os agentes podem instanciar, compartilhar e usar de forma dinâmica. Um artefato pode ser pensado como sendo uma ferramenta computacional, que existe no ambiente dos agentes e fornece algum tipo de função ou serviço. Essas funções ou serviços podem ser utilizados pelos agentes a fim de atingir os seus objetivos individuais ou coletivos (RICCI; PIUNTI; VIROLI, 2011; RICCI; VIROLI; OMICINI, 2006)

Diferentemente dos agentes, os artefatos são entidades não autônomas, e podem fazer referência a elementos físicos, como prateleiras, portas, e telefones, ou

cognitivos, como procedimentos operacionais e roteiros. Para serem utilizados, os artefatos oferecem aos agentes as suas *operações*. Essas operações alteram o estado do artefato, refletindo em alterações no ambiente. Além das operações, os artefatos também oferecem uma visão parcial do seu estado atual através das *propriedades observáveis* e de *sinais*, estes são percebidos pelos agentes que estiverem focados no artefato (OMICINI; RICCI; VIROLI, 2008; BOISSIER *et al.*, 2019).

Para ilustrar esses conceitos, pode-se imaginar um exemplo simples, em que o agente do SMA é um robô que a intenção de preparar o café (`!preparar_cafe`). Neste exemplo, a máquina de café é modelada como sendo um artefato `maquina_de_cafe`, com as operações `ligar` e `desligar` e uma propriedade observável `estado`, que pode assumir os valores `estado(ligado)` ou `estado(desligado)`. O agente `robo` pode realizar a operação de `ligar` no artefato, alterando a propriedade observável para `estado(ligado)` e, após isso, ficar aguardando receber do artefato um sinal `cafe_pronto` em sua base de crenças. Quando isso acontece, o agente pode ir até o artefato e executar a operação `desligar`.

2.1.3 Dimensão da organização

A dimensão da organização agrupa abstrações que dão estrutura e guiam como as atividades e interações dos agentes entre si e com o ambiente devem ser coordenadas (BOISSIER *et al.*, 2019).

Em um SMA que considera essa dimensão admite-se que a organização existe *a priori* e os agentes devem segui-lá. Hübner, Sichman e Boissier (2002) classifica os modelos de organização em dois grupos: a organização *funcional* e *estrutural*. No primeiro grupo podem ser feitos as especificações dos planos globais, alocações de tarefas para os agentes, e coordenação para execução os planos. O principal objetivo ao estruturar essa organização é atingir os planos globais de uma maneira melhor, mais eficiente. O segundo grupo representa uma estrutura mais estática da organização, focando nos *papeis* da sociedade dos agentes, nas relações entre eles, obrigações e permissões. Neste tipo de estruturação, os objetivos globais devem ser atingidos enquanto os agentes seguirem suas obrigações e permissões adquiridas, de acordo com o papel assumido na estrutura organizacional.

Retomando o exemplo do robô e da máquina de café, imagina-se que agora existam dois robôs compartilhando a mesma máquina de café na empresa e que, além de fazer, os robôs devem servir o café em diversos setores. Nesse novo formato, justifica-se a adição da organização ao sistema, que será responsável por coordenar as ações dos robôs de forma com que eles trabalhem organizadamente e sem conflitos de tarefas, mantendo todos os setores abastecidos de forma eficiente. Um ponto interessante em utilizar organização nessa abordagem é que os programadores dos robôs não precisam se preocupar em adicionar linhas de código responsáveis pela

coordenação de tarefas e, dessa maneira, conseguem focar melhor na programação das tarefas fundamentais para as quais o robô foi designado.

2.1.4 Relação entre dimensões

Para relacionar as dimensões dos agentes, ambiente e organização, este trabalho toma como base a proposta do JaCaMo, que foi a primeira plataforma de desenvolvimento de SMA a investigar explicitamente uma integração de todas as três dimensões (BOISSIER *et al.*, 2013). O JaCaMo é uma plataforma que integra os três ambientes de desenvolvimento de SMA: Jason; CArtAgO; e Moise. O primeiro é utilizado para criação de agentes BDI, o segundo é um framework para programação do ambiente e o terceiro implementa a programação de modelo organizacional. A proposta do JaCaMo é oferecer conexões semânticas entre os conceitos de cada dimensão, proporcionando uma programação de SMA complexos de forma uniforme e consistente.

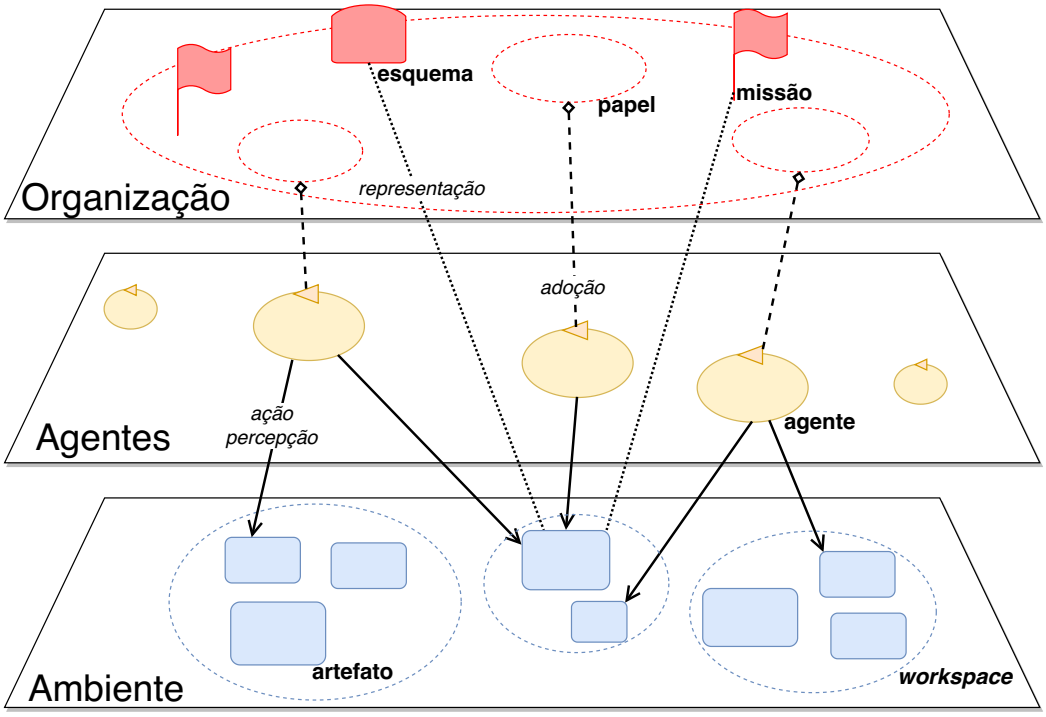
De forma simplificada, pode-se definir as conexões entre dimensões conforme a Figura 1. A relação entre os agentes e o ambiente se da por dois meios: (i) pela transformação de ações dos agentes em operações nos artefatos do ambiente e (ii) pela transformação das propriedades observáveis e sinais dos artefatos em crenças e eventos nos agentes. A conexão entre a organização e o ambiente é feita através da projeção da estrutura organizacional nos moldes de um artefato organizacional, dessa forma a arquitetura aproveita a dinâmica existente entre os agentes e o ambiente, permitindo reestruturações organizacionais em tempo de execução. Por fim, a relação entre as dimensões dos agentes e da organização se faz pela utilização dos artefatos organizacionais. Através deles, os agentes podem escolher adotar papeis e se comprometer com a execução de missões dentro da organização. Essas missões são compostas de sub-objetivos que foram atribuídos conforme os esquemas sociais. Os objetivos globais, definidos na dimensão da organização, são mapeados em objetivos individuais dos agentes que decidiram adotá-los. Essa delegação dos objetivos é expressada por obrigações.

2.2 DEPENDABILIDADE

Em sistemas computacionais o termo dependabilidade aparece com frequência, trata-se de uma tradução literal do termo em inglês *dependability*, e indica a qualidade do serviço oferecido pelo sistema (WEBER, 2002). Os conceitos relacionados a dependabilidade em softwares podem ser ilustrados em forma de árvore, como a Figura 2 apresenta. Essa árvore apresenta os *atributos*, os *meios* e as *ameaças* relacionadas a dependabilidade.

Os atributos são um conjunto de propriedades que servem como um caminho

Figura 1 – Representação das dimensões de um SMA com suas principais abstrações. As setas contínuas representam a relação de ação e percepção entre os agentes e os artefatos do ambiente. A seta tracejada indica a adoção de papeis e comprometimento com as missões da organização por parte dos agentes. Setas pontilhadas indicam a relação de representação da organização em artefatos organizacionais.

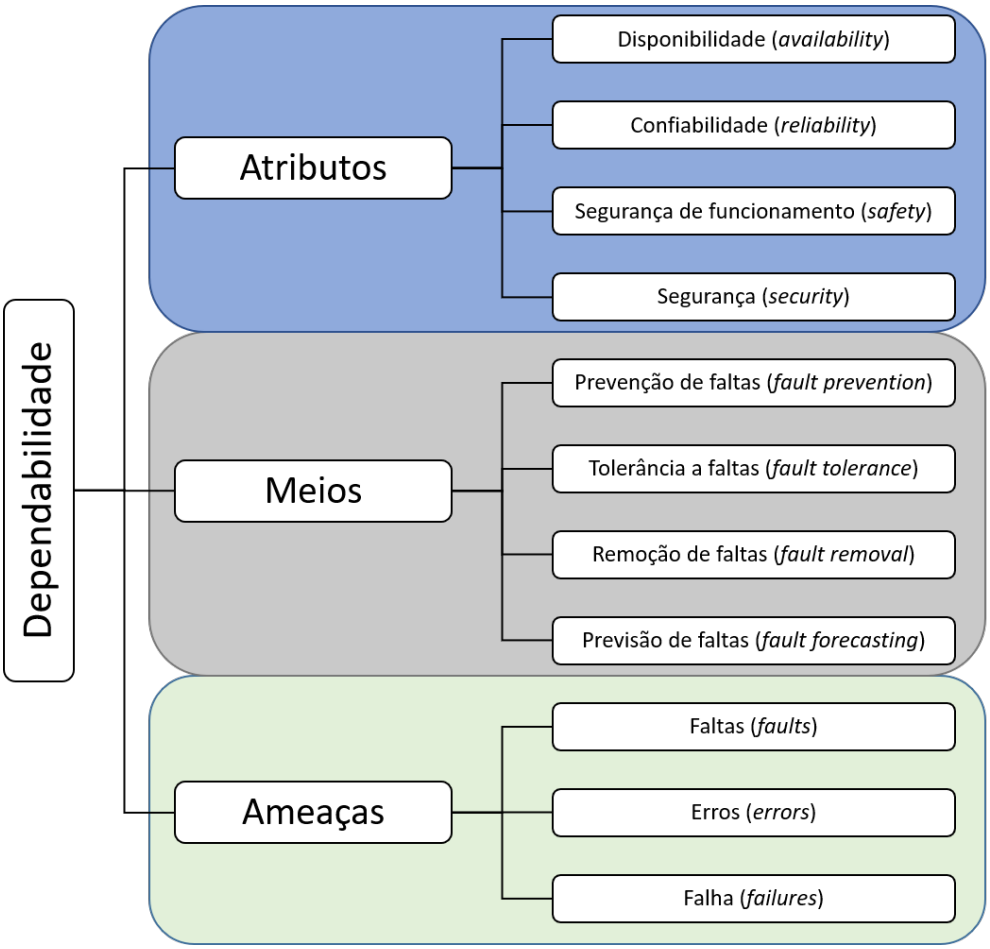


Fonte – Adaptado de Boissier *et al.* (2013).

para avaliar o nível da dependabilidade total do sistema. Os principais atributos são: disponibilidade, confiabilidade, segurança de funcionamento (*safety*) e segurança computacional (*security*). Entretanto, atributos adicionais podem derivar desses listados, dependendo da fonte utilizada (PULLUM, 2001).

Os fenômenos que ameaçam ou prejudicam a dependabilidade são três: a *falha* (*failure*), o *erro* (*error*) e a *falta* (*fault*). Por definição, uma falha é definida como um desvio da especificação, ou seja, ela ocorre quando o sistema não produz a mesma entrega para o qual este foi especificado inicialmente. O erro é uma parte dos estados do sistema que podem levar a uma falha subsequente. O surgimento de um erro é uma indicação de que uma falta ocorreu no sistema (LAPRIE, 1992). As faltas são processos inevitáveis, associadas ao universo físico e que podem ocorrer por uma série de motivos. Em contrapartida, uma falha pode ser evitada utilizando, por exemplo, técnicas de tolerância a faltas (WEBER, 2002). Dependendo do autor, as palavras *fault* e *failure* podem sofrer alterações em sua tradução, alguns autores traduzem *fault* com falha e *failure* como defeito. Nesse caso, por coerência, deve-se adotar o termo

Figura 2 – Árvore da dependabilidade.



Fonte – Adaptado de Laprie (1992).

tolerância a falhas.

Por fim, os meios para se atingir dependabilidade são divididos em quatro grupos: a *prevenção de faltas*, a *tolerância a faltas*, a *remoção de faltas* e a *previsão de faltas*. Resumidamente, pode-se definir cada um desses meios da seguinte maneira (PULLUM, 2001):

- **Prevenção de Faltas**: evitar ou prevenir a introdução e ocorrência de faltas;
- **Tolerância a Faltas**: prover serviços de acordo com a especificação, apesar das faltas;
- **Remoção de Faltas**: detectar a existência das faltas, localiza-las e eliminá-las;
- **Previsão de Faltas**: estimar a presença de faltas e suas consequências.

Na sequência do capitulo serão explorados quesitos pertinentes a tolerância a faltas, que é um dos tópicos principais do trabalho.

2.3 TOLERÂNCIA A FALTAS

O termo tolerância a faltas foi apresentado no trabalho de Avižienis (1967). Desde então ele tem sido utilizado pela comunidade acadêmica para tratar do comportamento de sistemas computacionais que estão sujeitos a ocorrência de faltas. Em alguns setores, nomes alternativos foram ganhando mais aceitação, como *sistemas redundantes* para o setor da indústria, ou *alta disponibilidade* para a comercialização de sistemas computacionais. O próprio termo tolerância a faltas não deve ser entendido como sendo uma propriedade absoluta, numa visão em que um sistema tolerante a faltas seria capaz de tolerar toda e qualquer falta em qualquer situação, o que de fato é uma promessa irrealizável. O objetivo da TF é ser um caminho para alcançar uma melhor dependabilidade, evitando que as faltas conduzam o sistema para uma falha.

Segundo Weber (2002), as técnicas de TF podem pertencer a duas classes: (i) mascaramento e (ii) detecção, localização e reconfiguração. As técnicas da primeira classe são mais indicadas para casos de sistemas de tempo real crítico, pois emprega bastante redundância e não apresenta os gastos de tempo para os processos que as técnicas de detecção, localização e reconfiguração exigem. As técnicas pertencentes a classe de mascaramento não serão abordadas neste trabalho.

No caso das técnicas de TF pertencentes a segunda classe, o autor Lee e Anderson (1990) descreve um fluxo de desenvolvimento em quatro fases, são elas: (i) detecção de erros, (ii) confinamento e avaliação de danos, (iii) recuperação dos erros e (iv) tratamento da falta e continuação dos serviços. Para se obter um sistema tolerante a faltas, no entanto, não é obrigatório que todas as fases sejam implementadas, cabendo ao projetista definir quais seriam as etapas imprescindíveis. Um resumo sobre cada fase é apresentado a seguir:

- **Detecção de erros**: A detecção dos erros usualmente é o ponto de partida para a implementação sistemas tolerantes a faltas. Nesta fase inicial, implementam-se medidas para que o sistema consiga detectar um erro e fazer surgir um sinal de exceção. Na prática raramente é possível garantir que um sistema seja capaz de criar um identificador ideal que realize a detecção de todos os erros do sistema. O que adota-se na maioria dos sistemas é um termo conhecido como “aceitabilidade”, onde se busca garantir que uma grande parte dos casos seja detectado. Como exemplo pode-se citar a verificação em um sistema onde existem muitos processos concorrentes, nesse caso é possível optar por realizar uma verificação baseada em cada processo enviando mensagens de verificações periódicas (e.g., “*I’m alive*”), ao invés de uma verificação completa no sistema.
- **Confinamento e avaliação dos danos**: Devido a existência de um intervalo de tempo entre a manifestação de uma falta e a detecção de um erro, é ra-

zoável presumir que informações inválidas podem se espalhar pelo sistema, gerando erros futuros. Dessa maneira, nessa fase de pode-se adotar estratégias para avaliação de danos, tentando estabelecer com uma precisão maior qual foi a dimensão do dano causado pela falta. Todavia, na prática temos que a avaliação de danos é um aspecto bastante incerto da TF. Na maioria dos sistemas é mais habitual tentar evitar uma propagação dos danos do que determinar quais foram os danos que um fluxo de informação incorreta pode ter causado. A medida que os sistemas de software se tornam mais distribuídos e descentralizados, a importância de se evitar propagações de danos será cada vez maior.

- **Recuperação dos erros**: Diferentemente das fases anteriores, a fase de recuperação de erros tem características ativas. Seu objetivo é levar o estado de erro atual para um estado livre de erro e bem definido, no qual o sistema pode seguir operando. Dependendo do tipo de dano que o sistema sofre, pode-se optar pela manipulação de parte dos estados de erro ou então pela troca por completo desses estados para que o sistema volte a sua operação normal.
- **Tratamento da falta e continuação do serviço**: As primeiras três fases se preocupam com os erros no sistema. A ultima fase, por sua vez, se preocupa com a origem do erro, que é a falta. A justificativa para realização dessa investigação é que a mesma falta pode ocorrer novamente caso ela não seja identificada e corrigida. Em sistemas de software, no entanto, estratégias para tratamento automático de faltas é tão complexo que raramente é investigada e, em vez disso, ignora-se a origem da falta na esperança que ela não ocorra novamente, como no caso das faltas transitórias, que ocorrem uma vez, mas não necessariamente irão ocorrer no futuro.

Para cada fase existem mecanismos que podem ser implementados, os mais comuns estão listados no Quadro 1. Os itens destacados apresentam relação com esse trabalho e serão explorados no restante da sessão.

2.3.1 Checagens de tempo

As checagens de tempo são uma ferramenta extremamente versátil e efetiva para detectar erros em processos. A ideia desta ferramenta é bastante simples: em uma situação hipotética tem-se um processo `p`, performando uma operação periódica crítica que, ao final da execução, libera os bloqueios de acessos para outros processos concorrentes. Nesse cenário, pode-se perceber que a ocorrência de faltas no processo crítico poderá resultar no bloqueio de diversos outros processos. Ao implementar uma checagem de tempo, cria-se um processo adicional `w`, que tem a função de monitorar

Quadro 1 – Exemplos de mecanismos para cada fase de um sistema TF.

Fases	**Principais Mecanismos**
Detecção de erros	replicação (*replications checks*) checagens de tempo (*timing checks*) testes de reversão (*reversal checks*) codificação (*coding checks*) testes de razoabilidade (*reasonableness checks*) testes estruturais (*structural checks*) testes de diagnósticos (*diagnostic checks*)
Confinamento e avaliação dos danos	processos atômicos esquema baseado em listas ou tickets operações encapsuladas cache de recuperação trilhas de auditoria (*audit trails*)
Recuperação dos erros	recuperação de erro por retorno (backward error recovery) recuperação de erro por avanço (forward error recovery)
Tratamento da falta e continuação do serviço	localização da falta por diagnostico reparo do sistema por reconfiguração

Fonte – Adaptação de Lee e Anderson (1990).

a execução de `p`, requerendo que o segundo envie periodicamente um “sinal de vida” (*heartbeat*) para o primeiro.

Através dessa verificação, o processo monitor assegura que o processo monitorado está ao menos capacitado de enviar sinais de vida durante o último período em que a checagem de tempo ocorreu. Caso o processo `w` não receba o sinal esperado no tempo limite, um sinal de exceção é gerado pelo sistema para seja tomado a devida ação de reparo. No que se refere ao projeto de um sistema de checagem de tempo, existem alguns pontos que o projetista deve levar em conta como, por exemplo, as frequências que o processo `p` deve enviar o sinal de vida e que `w` deve verificar as mensagens recebidas, fatores que tem impacto no desempenho do sistema e na latência do sistema de detecção do erro. Em sistemas de software, um processo bastante conhecido de checagem de tempo são os processos “cães de guarda” (*watchdog timer*) (DE FLORIO, 2009).

2.3.2 Recuperação de erro por retorno

A recuperação de erro por retorno (*backward error recovery*) é um mecanismo de tolerância a faltas largamente utilizado. A ideia desse mecanismo é que alguma entidade (e.g., um usuário, o próprio sistema, o programador) deve periodicamente armazenar o estado “instantâneo” (*snapshot*) do sistema e, caso ocorra alguma falta no futuro, o sistema consiga recarregar esse estado salvo anteriormente, permitindo

que o sistema retorne a sua operação correta. Esse tipo de procedimento visa corrigir faltas transientes, que são as faltas que ocorrem e possivelmente não devem ocorrer novamente na retomada do sistema (DE FLORIO, 2009).

Os estados instantâneos que são salvos em tempo de execução são um dos pontos chaves desse mecanismo, conhecidos como "pontos de restauração" do sistema (*checkpoints*), eles podem ser criados em períodos fixos de tempo (*checkpoints* baseados em tempo), criados a partir de um sinal gerado por um evento no sistema (*checkpoints* baseado em eventos) ou ainda criados em partes especificas do algoritmo (*checkpoints* baseados em algoritmo), como no inicio de um laço de repetição.

Para processos concorrentes que operam e trocam informações entre si, como é o caso dos SMA, tem-se algumas peculiaridades quando implementa-se uma recuperação de erro por retorno. Destaca-se o *efeito dominó*, que é quando ocorre um propagação de recuperações entre processos que compartilham um fluxo de informação. Pode-se imaginar a situação em que um processo `p` faz recuperação (devido a ocorrência de uma falta) após ter transmitido informações, essas informações podem estar corrompidas e, dessa maneira, o processo `r`, que é o destinatário da mensagem, deveria executar uma recuperação para garantir integridade do sinal. Esse ciclo pode-se estender por todos os processos concorrentes do sistema que receberam informações duvidosas, gerando um grande retrocesso em toda a operação. Quando implementa-se esse tipo de mecanismo em sistemas concorrentes, é recomendado que sejam impostas algumas restrições ou que penalidades sejam toleradas. Como exemplo, pode-se limitar a capacidade de propagação de pontos de recuperação ou então aceitar um nível de perda em termos de rendimento do sistema, devido ao tempo gasto para realizar cadeias de recuperação mais longas (LEE; ANDERSON, 1990).

2.4 TOLERÂNCIA A FALTAS EM SISTEMAS MULTIAGENTES

A questão de como as faltas devem ser tratada em SMA foi inicialmente discutida em Hägg (1997), que afirma que existem duas opiniões opostas sobre a vulnerabilidade desse tipo de sistema perante as faltas. Um ponto de vista diz que um SMA é inerentemente tolerante a faltas, já o outro lado alega que os SMA é inerentemente inseguro. Essa divergência de opiniões se da devido ao fato do paradigma de agentes possuir uma arquitetura modular, que é uma estrutura básica de sistemas tolerantes a faltas, pois ela contribui para evitar a dispersão dos efeitos de uma falta no restante do sistema. Em contrapartida, os SMA possuem outra característica que penaliza a dependabilidade, que é a questão do não-determinismo, ou seja, um SMA pode apresentar diferentes comportamentos ao longo do tempo, mesmo quando expostos as mesmas entradas.

Atualmente existem diversos trabalhos desenvolvidos na área de tolerância a faltas em sistemas multiagentes, esses trabalhos apresentam diferentes abordagens

para implementar mecanismos de TF visando fornecer uma execução confiável do SMA, especialmente em termos de detecção e recuperação de erros. Nos trabalhos de Stanković, Štula e Maras (2017) e Isong e Bekele (2013) foram analisados diversos estudos desenvolvidos na área de TF aplicadas a SMA. Nessas revisões da literatura foram levantados quais os principais modos de falha (*failure modes*) que os SMA estão suscetíveis e quais as principais características que as abordagens de TF apresentam. Um modo de falha é a maneira como identifica-se que o sistema falhou (LAPRIE, 1992). Os principais modos de falha dos SMA são:

- **Falha de quebra (*Crash failure*):** ocorre quando o processo para de responder por completo, mantendo o seu estado inalterado.
- **Falha de omissão (*Omission failure*):** ocorre quando o processo deixa de enviar e/ou receber mensagens, continuando com o restante dos serviços,
- **Falha de tempo (*Timing failure*):** ocorre quando o processo não consegue finalizar dentro de um período específico, desde que algum período de tempo tenha sido definido.
- **Falha Arbitraria/Bizantina (*Arbitrary/Byzantine failure*):** ocorre quando o processo, de maneira imprevisível, desvia do seu comportamento predefinido. O processo pode parar, começar a perder mensagens, performar cálculos com erros ou enviar mensagens corrompidas.

Stanković, Štula e Maras (2017) aponta três características nas abordagens de TF em SMA que podem ser usados para classificar e comparar as abordagens entre si. São elas: o (i) tipo de detecção de faltas, o (ii) protocolo de tolerância a faltas e (iii) nível de manipulação de falta. As duas primeiras características condizem exatamente com as fases para desenvolvimento de sistemas TF, apresentadas na Seção 2.3. O tipo de detecção de faltas refere-se a Fase I, que Lee e Anderson (1990) chama de detecção de erros. O protocolo de tolerância a faltas está relacionado a Fase III, chamada de recuperação dos erros.

A novidade apresentada por Stanković, Štula e Maras (2017) é a terceira característica das abordagens, chamada de *nível de manipulação de falta*. Essa característica indica em qual nível deve ocorrer o gerenciamento das exceções e a recuperação das faltas: (i) no nível dos agentes ou (ii) no nível do SMA. Esse tipo de classificação também é abordada no trabalho de Dellarocas e Klein (2000), que utiliza as denominações abordagem *"sobrevivente"* e *"cidadã"*.

Na abordagem sobrevivente, adota-se a ideia de que cada agente deve ser responsável por manter a sua própria existência, ou seja, a programação dos mecanismos de TF devem ser adicionadas ao código de cada agente. Esse tipo de abordagem é bastante tradicional, porém apresenta uma série de desvantagens. Uma delas é a adição de responsabilidades sobre os desenvolvedores do SMA, que devem coordenar o

comportamento dos agentes na ocorrência de faltas que são, muitas vezes, difíceis de prever. Essa adição de código defensivo nos agentes resulta em uma maior dificuldade para manutenção de código, pode prejudicar o desempenho do sistema e aumentar as chances de ocorrências de novas faltas inseridas pelos próprios desenvolvedores.

A abordagem cidadã, por sua vez, estabelece que os processos de TF devem ser gerenciados por uma entidade especializada e externa ao agente, habitualmente chamada de *sentinela*. Ao adicionar esse tipo de entidade, incorpora-se ao sistema a noção de expertise, separando o domínio "normal" do SMA com o domínio de tratamento de faltas. Nessa configuração, os agentes "comuns" do sistema ficam focados nos seus objetivos mais básicos. A abordagem cidadã traz analogias com a sociedade humana, onde os cidadães (i.e., os agentes do sistema) reconhecem que existem instituições pré estabelecidas fornecedoras de serviços especializados. Como exemplo, pode-se pensar na polícia, que oferece serviços de segurança aos cidadãos e, em troca, é reconhecida por eles como uma autoridade que tem permissão para atuar no estado de cada um deles, podendo, por exemplo, prender indivíduos que estejam causando problemas aos demais (DELLAROCAS; KLEIN, 2000).

2.5 TRABALHOS RELACIONADOS

Entre os diversos trabalhos que abordam a TF aplicadas aos SMA, alguns serviram de inspiração para a elaboração do presente trabalho. Em Marin, Bertier e Sens (2003) foi apresentado o *framework* DARX (*Dynamic Agent Replication eXtension*), que é uma arquitetura que permite a criação de replicas para os agentes do sistema. A ideia básica de uma réplica é que ela deve possuir a capacidade de executar as mesmas tarefas que o agente original e estar apta a assumir o papel desse agente caso fosse necessário. O *framework* proposto por Marin tem a capacidade de criar as réplicas de forma adaptativa e em tempo de execução, podendo alterar algumas características das réplicas,além de controlar o número de replicas presentes no sistema de acordo com a criticidade do agente. Para detecção de faltas, o *framework* DARX utiliza controle de mensagens do tipo *heartbeats*.

Em seu trabalho, Hägg (1997) introduziu o termo *sentinela*, uma classe especial de agente que tem a função de proteger algumas funcionalidades do sistema. Esse tipo de agente não pertence ao domínio da solução de problemas, ficando responsável por monitorar os agentes comuns do sistema e, se for detectado a ocorrência de alguma falta, o sentinela pode tomar ações corretivas, como escolher planos alternativos, excluir agentes, alterar parâmetros do sistema e reportar aos operadores humanos. O sentinela pode monitorar a comunicação dos agentes e interagir com eles, além de usar *timers* para detectar agentes mortos ou faltas em links de comunicação.

Klein, Rodriguez-Aguilar e Dellarocas (2003) é outro trabalho que utiliza um serviço de tratamento de faltas baseado em agentes sentinelas inseridos no sistema

para monitorar outros agentes utilizando trocas de mensagens do tipo *heartbeat*. Além disso, os sentinelas comunicam entre si para enviar notificações caso algum agente tenha entrado em estado de erro. No modelo de Klein, todo agente tolerante a faltas acessa o SMA portando uma "assinatura" com configurações sobre o seu protocolo de *heartbeat* e como devem ser tratados as exceções caso ele entre em estado de erro. O desenvolvedor do SMA pode optar por criar agentes com uma assinatura padrão, assinatura personalizada ou agente comuns, que não possuem nenhum tratamento contra faltas.

Em sua tese, Díaz (2018) apresentou o eJason, um interpretador da linguagem de programação Jason em Erlang. Sua principal contribuição foi o projeto e implementação de um ambiente de desenvolvimento para SMA que combina as sintaxes e semânticas do Jason, que é uma linguagem de programação para SMA com alto nível de abstração para programação de agentes BDI, com as ferramentas de TF do Erlang, uma linguagem de programação com características que favorecem o desenvolvimento de aplicações concorrentes, distribuídas e com alta robustez. Entre as ferramentas de TF fornecidas pelo eJason, destacam-se dois tipos de relações que podem ser estabelecidas entre os agentes do sistema: a relação de *monitor* e de *supervisor*. A primeira relação permite que agentes monitores recebam informações sobre acontecimentos nos agentes monitorados. Já a segunda relação permite que, além de receber informações, os agentes supervisores possam realizar atuações especiais de controle sobre os seus supervisionados.

O modelo de TF proposto nesse trabalho procurou incorporar pontos positivos dos trabalhos citados. O trabalho desenvolvido por Marin vem servindo como base para diversos outros estudos. Um dos pontos chaves desse *framework* está na ideia de criar um serviço de tratamento de faltas dinâmico e externo ao SMA. O presente trabalho procura explorar esse mesmo conceito, permitindo que sejam feitas configurações de parâmetros iniciais e fornecendo informações sobre o sistema em tempo real, para que esses parâmetros possam ser adaptados conforme evolução do SMA.

Os trabalhos de Hägg e Klein também servem como referência na área de TF em SMA, seus modelos conceituais deixam claro a separação entre domínio "natural" do sistema e domínio para tratamento das faltas. O presente trabalho procura combinar esse tipo de solução com conceitos da programação multidimensional, com o intuito de que as abstrações e as conexões semânticas das multidimensões possam contribuir com os mecanismos de TF.

A tese de Díaz também trouxe referências para o presente trabalho. As relações de monitor e supervisor do eJason foram incorporadas ao modelo deste trabalho, que utiliza essas relações associadas as abstrações da linguagem JaCaMo. O eJason também é uma inspiração pelo fato utilizar o AgentSpeak, que é uma linguagem de programação orientada a agentes baseada na arquitetura BDI. A mesma linguagem e

arquitetura são utilizadas no modelo deste trabalho.

3 MODELO DE TOLERÂNCIA A FALTAS PARA SISTEMAS MULTIAGENTES MULTIDIMENSIONAIS

Com base nos estudos apresentados na revisão da literatura, esse trabalho concebeu um modelo de TF para SMA baseado em detecção de erros por checagens de tempo via monitor de batimentos cardíacos (*heartbeat*), combinado com um protocolo de recuperação baseado em recuperação de erro por retorno, que registra *checkpoints* periodicamente para realizar restaurações em casos de falta. O modelo gerencia o tratamento de faltas no nível do sistema (abordagem "cidadã") e utiliza abstrações das dimensões dos agentes e do ambiente para implementar os mecanismos de TF e as funcionalidades da relações de monitoramento por interesse e de monitoramento contínuo. O Quadro 2 resume as principais características do modelo.

No restante do capítulo serão apresentados uma visão geral do modelo, detalhamentos a respeito de cada entidade e funcionalidades que o modelo fornece. Alguns detalhes à nível de implementação do modelo na linguagem JaCaMo também são apresentados.

Quadro 2 – Resumo das características do modelo de TF.

Funcionalidade fornecidas pelo modelo	detecção e recuperação de erros monitoramento por interesse monitoramento contínuo
Dimensões utilizadas pelo modelo	agentes ambiente
Principais entidades do modelo	agente de saúde artefato monitor de saúde
Mecanismo de detecção	monitor de *heartbeat*
Mecanismo de recuperação	recuperação de erro por retorno
Nível de manipulação das faltas	nível do SMA (abordagem cidadã)

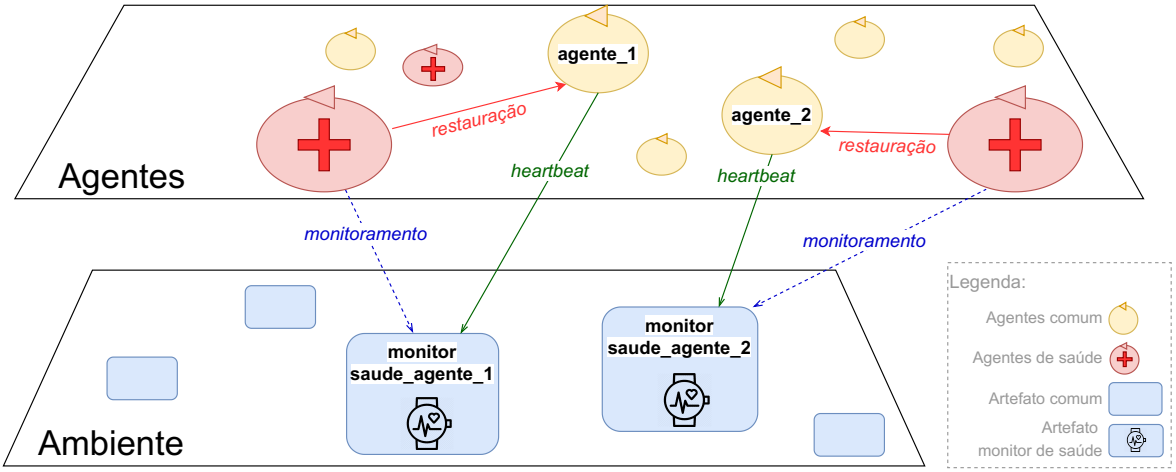
3.1 VISÃO GERAL

O modelo de TF proposto neste trabalho faz uma analogia com um sistema de monitoramento de saúde remoto, onde os indivíduos são acompanhados por equipamentos móveis, como os relógios inteligentes (*smartwatches*) ou as pulseiras monitoras de frequência cardíaca, que tem a capacidade de monitorar aspectos da saúde do indivíduo e enviar informações para equipes médicas, que podem interpretar as informações dos equipamentos e entrar em ação nos casos de emergência.

Neste cenário, o modelo fornece mecanismos de TF para SMA que contam com, ao menos, duas dimensões (agentes e ambiente). A Figura 3 ilustra um sistema deste tipo atuando juntamente com o modelo de TF. Nessa representação, o SMA recebe a adição de duas entidades básicas do serviço de TF: o artefato `monitor_de_saude`,

instrumentado na dimensão do ambiente, e os `agentes_de_saude`, que estão inseridos na dimensão dos agentes.

Figura 3 – Modelo de TF para SMA Multidimensional.



Nesta concepção, cada agente do sistema pode dispor de um artefato monitor de saúde atrelado a si. O monitor mantém uma relação exclusiva com o agente, verificando periodicamente o seu estado de saúde, que é indicado na forma de batimento cardíaco (*heartbeat*). Dessa maneira, a ocorrência de faltas no agente é interpretada pela ausência de sinais vitais, sinalizando a morte do agente.

A segunda entidade adicionada ao sistema, os agentes de saúde, são agentes especializados que possuem a capacidade de socorrer os agentes "comuns" do sistema. Os agentes de saúde possuem permissões para executar protocolos de recuperação específicos para restaurar um agente que esteja em estado de erro.

Além do serviço de TF fornecido pelo agente de saúde e pelo monitor de saúde, o modelo deste trabalho disponibiliza outras duas funcionalidades para o SMA: a relação de *monitoramento "por interesse"* entre agentes do sistema e a relação de *monitoramento contínuo* do agente. Na sequência deste capítulo serão detalhados o funcionamento do modelo, das entidades que participam do modelo e das relações de monitoramento por interesse e contínuo do agente. Para cada tópico, serão apresentados informações sobre como se deu a implementação do modelo na linguagem JaCaMo[1].

3.2 AS ENTIDADES DO MODELO

O modelo de TF é estruturado para funcionar em SMA com, pelo menos, duas dimensões. Nessas dimensões existem três entidades que se relacionam para fornecer

[1] Para detalhes sobre a implementação, pode-se consultar o repositório do trabalho em `https://gitlab.com/luis.lampert/fault-tolerant-jacamo`.

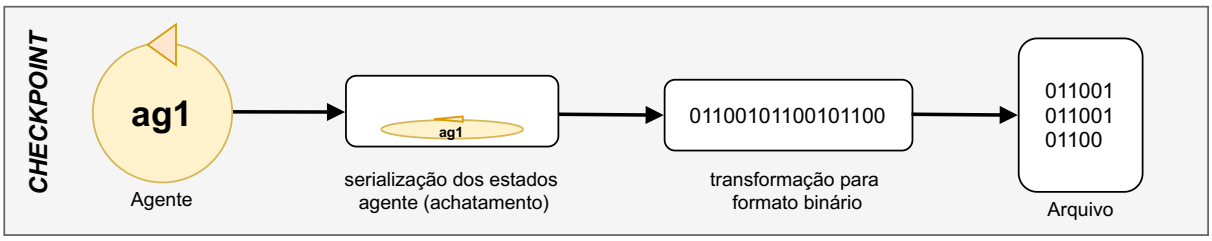
os mecanismos de TF ao SMA, são elas: o agente tolerante a faltas, o agente de saúde e o artefato monitor de saúde.

3.2.1 O agente tolerante a faltas

O agente do sistema que é configurado como tolerante a faltas possui as mesmas características de um agente comum do sistema, acrescidos de funções específicas que o modelo implementa. As principais funções extras são: (i) enviar periodicamente um sinal do tipo *heartbeat* para o artefato monitor de saúde e (ii) executar periodicamente o processo de gravação dos pontos de restauração dos seus estados (*checkpoint*).

O envio do sinal de *heartbeat* é realizado por meio de uma operação do agente tolerante a faltas no seu respectivo artefato monitor de saúde. O processo de criação de pontos de restauração é realizado através da técnica de "serialização" de objetos, que converte os objetos do sistema em sequências de bytes e os armazena em arquivos. Ao executar esse processo, o agente tolerante a faltas armazena os seus estados naquele instante de tempo, salvando as suas crenças, desejos, intenções e todas informações necessárias que permitam uma futura restauração do agente no SMA. O esquema da Figura 4 apresenta as etapas de um processo de *checkpoint*.

Figura 4 – Processo de criação de pontos de restauração (*checkpoint*) do agente.



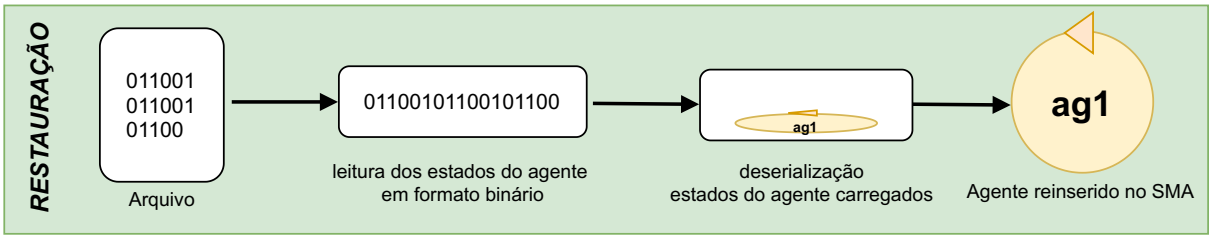
3.2.2 O agente de saúde

O agente de saúde é outra entidade do modelo que está presente na dimensão dos agentes. Trata-se de um agente BDI com tarefas específicas e permissões especiais dentro do SMA. Suas principais funções são: (i) criar artefatos monitores de saúde sempre que um agente tolerante a faltas for inserido no SMA, (ii) monitorar o estado de saúde do agente através das propriedades observáveis do artefato monitor de saúde e (iii) iniciar protocolo de recuperação ao detectar a presença de alguma falta no agente, reestabelecendo o agente de acordo com o último ponto de restauração válido.

O processo de restauração do agente utiliza a técnica de "deserialização", que é o caminho inverso apresentado anteriormente. Nesse processo o arquivo de bytes é lido e convertido novamente para um objeto do sistema, conforme indica o esquema da Figura 5. Ao fazer isso, o agente de saúde reestabelece o agente tolerante a faltas

no SMA com os mesmos estados (i.e., mesmas crenças, desejos e intenções) salvos no último ponto de restauração.

Figura 5 – Processo de restauração do arquivos de bytes em objetos.



3.2.3 O artefato monitor de saúde

O artefato monitor de saúde é a entidade do modelo presente na dimensão do ambiente. Ele desempenha a função de exibir as condições de saúde dos agentes monitorados e detectar a presença de erros. Para fazer isso, o artefato executa um monitor de *heartbeat*, que é uma *thread* responsável pela checagem de tempo dos sinais vitais do agente.

Assim como todos os artefatos do ambiente, o monitor de saúde disponibiliza como interface para as outras dimensões as suas propriedades observáveis e operações. Conforme indica a Figura 6, a principal operação disponível para os agentes é o processo de *heartbeat* e as principais propriedade observáveis são:

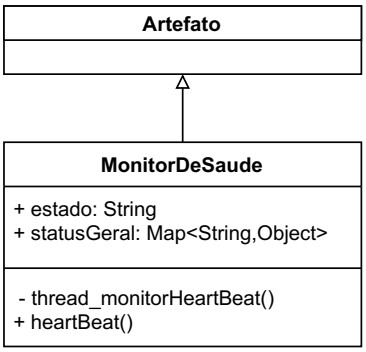
- **o estado** do agente, que pode assumir os valores: “vivo” (*live*), em operações normais, “atrasado” (*late*), caso o agente atrase os primeiros sinais de *heartbeat*, ou “morto” (*dead*), quando o agente não envia mais sinais vitais para o monitor de saúde
- **o status geral** do agente, que mostra informações do nível mental e nível de arquitetura do agente, que são informações que dizem respeito a características internas do agente e do ambiente que ele está inserido. Esses dois níveis serão aprofundados na Seção 3.4, que trata sobre o monitoramento contínuo.

3.2.4 Relação entre as entidades

Para facilitar o entendimento de como as entidades do modelo atuam e se relacionam durante os ciclos de operação do SMA, as Figuras 7 e 8 apresentam diagramas de sequência que ilustram casos de ciclos de operação normal e com ocorrência de falta, respectivamente.

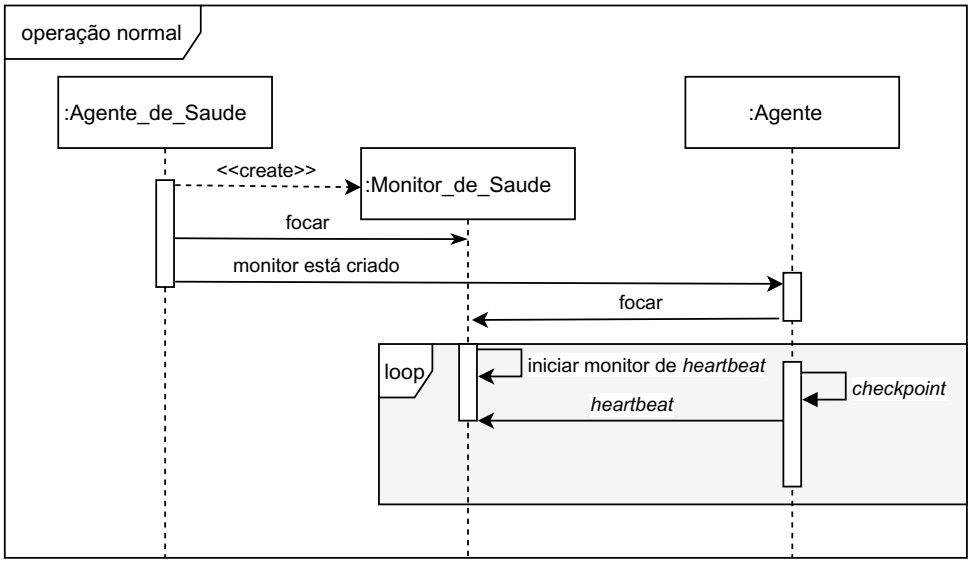
Em um ciclo normal de operação (Figura 7), onde não existe a ocorrência de nenhuma falta, o agente de saúde instancia um artefato monitor de saúde no momento

Figura 6 – Diagrama de classes do artefato monitor de saúde, indicando as principais propriedades observáveis e operações disponíveis.



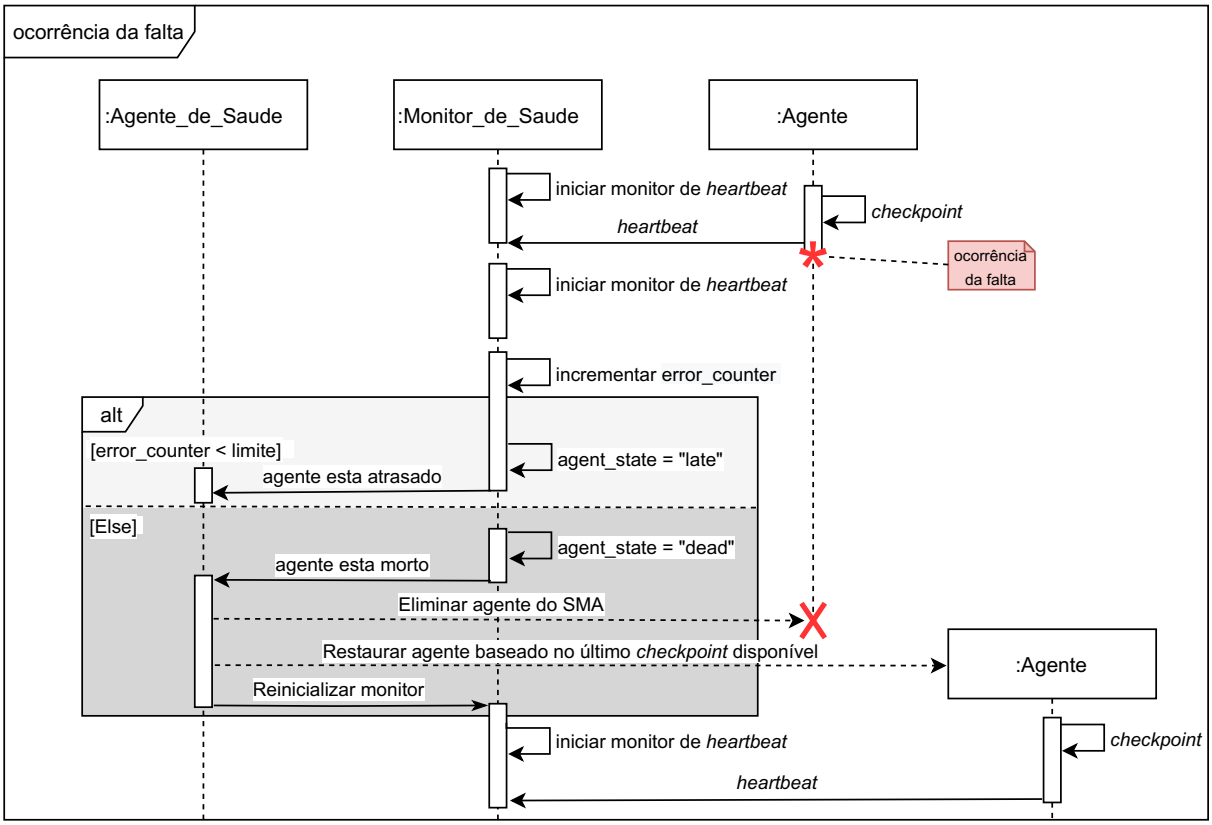
em que um agente tolerante a faltas é inserido no sistema. Após isso o agente de saúde informa o agente tolerante a faltas que o seu monitor está operando e, a partir dai, ambos focam suas atenções para o artefato. Posteriormente, o agente tolerante a faltas realiza o processo periódico de criar um ponto de restauração e enviar um sinal de *heartbeat* para o artefato monitor de saúde. Caso os sinais vitais do agente estejam sendo enviados dentro do período esperado pelo monitor de *heartbeat*, a operação é considerada normal e o estado do agente permanece como “vivo”.

Figura 7 – Diagrama de sequência para um ciclos livres de falta.



No caso em que existe a ocorrência de uma falta no agente (Figuras 8), o monitor de saúde acusa a falta do sinal de *heartbeat*, incrementando um contador e erros e alterando o estado do agente para “atrasado”. Caso o contador de erros seja superior a um limite pré estabelecido, o monitor de saúde altera o estado do agente para “morto”. Neste momento o agente de saúde inicializa o protocolo de recuperação, que consiste em eliminar o agente defeituoso e restaurar o agente com base no último ponto de restauração existente.

Figura 8 – Diagrama de sequência para um ciclos com ocorrência de falta.



3.2.5 Implementação das entidades do modelo no JaCaMo

Para que o modelo atue como um *framework* para a linguagem, ou seja, para que ele adicione funcionalidades de TF de maneira transparente para o desenvolvedor, optou-se por customizar a arquitetura padrão de agentes JaCaMo. Dessa maneira, quando se deseja adicionar as funcionalidades do modelo a um agente em específico, basta o usuário optar pela arquitetura customizada tolerante a faltas, como indica a Figura 9. Esse trecho de código exemplifica a criação de um SMA composto por dois agentes distintos, `agente_1` e `agente_2`, sendo que o primeiro é um agente comum da linguagem JaCaMo e o segundo está sendo criado com os mecanismos de TF que o modelo oferece. Quando essa configuração é feita, o SMA automaticamente adiciona planos específicos de TF para o `agente_2`, além de adicionar ao SMA um agente de saúde e um artefato monitor de saúde.

No que diz respeito aos planos que um agente tolerante a faltas recebe, os principais referem-se ao processo periódico de enviar sinais vitais ao monitor de saúde e ao processo periódico de salvar pontos de restauração próprios. Esses dois planos são identificados pelos marcadores `@ft1` e `@cp1`, respectivamente, e podem ser analisados na Figura 10. No trecho de código, o agente possui dois planos, no primeiro o agente envia um sinal vital para o artefato monitor de saúde e aguarda o período pré configurado até que o plano possa ser executado novamente. No segundo plano, o

Figura 9 – Segmento de código para adição da arquitetura de TF.

```
1       mas sma_example {
2         agent agente_1{
3           instances:1
4           }
5
6         agent agente_2{
7           instances:1
8                ag-arch: customArch.FTArch % arquitetura customizada para TF
9
10        }
11      }
```

agente cria um ponto de restauração em um diretório local padrão e aguarda o período necessário para executar o plano novamente.

A ação destacada no código na linha 9 é referente ao processo de criação de pontos de recuperação (*checkpointing*). Essa ação foi implementada no JaCaMo na forma de ações internas (*internal actions*), que são ações executadas pelo agente que não alteram o ambiente em que ele esta inserido (diferentemente das operações executadas em artefatos). Com isso, o agente pode realizar a gravação de pontos de recuperação dos seus estados durante a execução de um plano, utilizando a performativa `customIA.checkpoint()`.

Figura 10 – Plano de *heartbeat* e *checkpoint* do agente.

```
1     @ft1
2     +!heartbeat_signal : hb_period(D) & .my_name(N)
3      <-  ft::heartBeat(N,system.time); % enviar heartbeat para o monitor de saúde
4          .wait(D); % aguardar o período de heartbeat configurado
5          !!heartbeat_signal. % colocar novamente o plano nas intenções do agente
6
7     @cp1
8     +!checkpoint : cp_delay(D) & .my_name(N)
9      <- customIA.checkpoint(My_name); % criar novo ponto de restauração do agente
10       .wait(D); % aguardar o período entre checkpoints configurado
11       !!checkpoint. % colocar novamente o plano nas intenções do agente
```

O agente de saúde, por sua vez, possui planos específicos e permissões especiais para garantir o funcionamento dos mecanismos de TF. Sua implementação em JaCaMo é feita como qualquer agente BDI, sendo que os principais planos implementados estão identificados na Figura 11. No trecho de código, o plano identificado por `@e1` é um evento gatilho para iniciar o protocolo de recuperação (plano `@p1`). Esse evento é acionado sempre que o artefato monitor de saúde detecta que um agente está em estado de erro. No plano `@p1` o agente de saúde ativa outros dois planos, o primeiro para reiniciar o artefato monitor de saúde (plano `@p2`), limpando todo o histórico de erros do agente, e o segundo para restaurar o agente no sistema (plano `@p3`) com base

nos estados salvos anteriormente. Esse plano de restauração utiliza, na linha 17, outra *internal action* criada para o JaCaMo.

Figura 11 – Planos padrões de um o agente de saúde.

```
1     @e1
2      +N::dead_agent[artifact_id(AId)] : monitoring(N,Art_name,AId,_)
3      <- !recovery_protocol(N,Art_name).
4
5     @p1
6     +!recovery_protocol(N,Art_name)
7     <- !clear_artifact(Art_name); % chamar plano para reiniciar artefato
8        !revive_ft_agent(N). % chamar plano para reviver agente morto
9
10    @p2
11    +!clear_artifact(Art_name) : monitoring(N,Art_name,AId,_)
12    <- N::clear_monitor[artifact_id(AId)]. % reinicializar o monitor de saúde
13
14    @p3
15    +!revive_ft_agent(N) : monitoring(N,Art_name,_,WsN)
16    <- .print("Resuming agent ",N," in the System");
17       customIA.resume_agent(N). % internal action para restaurar o agente
```

Os processos de *checkpoint* e restauração, conforme explicado anteriormente, são construídos a partir da técnica de *serialização* e *desserialização* de objetos, que converte os objetos do sistema em sequências de bytes e armazena em arquivos para que possam ser salvos, transferidos, lidos e convertidos novamente em objetos. As implementações dessas técnicas foram realizadas através da interface *Serializable*, da linguagem Java, que está presente no pacote *java.io*. [2].

3.3 MONITORAMENTO POR INTERESSE

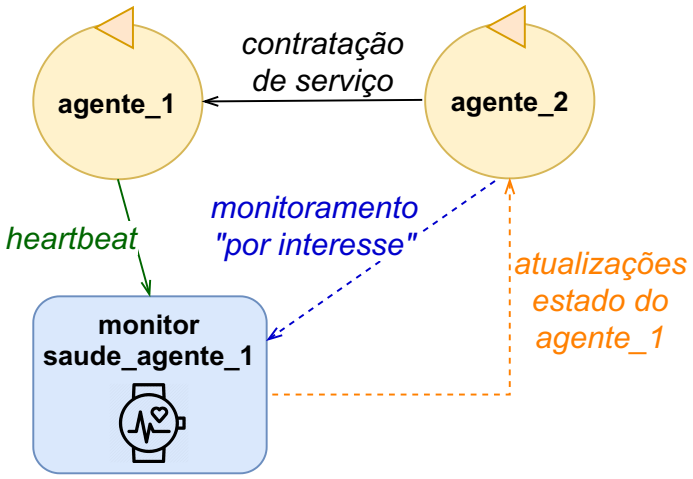
O modelo de TF deste trabalho foi projetado de maneira que possa ser estabelecido uma relação de monitoramento “por interesse” entre agentes do sistema. De maneira análoga as pulseiras monitoras de frequência cardíaca, o artefato monitor de saúde serve como um “visor”, que outros agentes do sistema podem observar e se atualizar sobre o estado de saúde do agente monitorado. A abstração dos artefatos do ambiente permite essa funcionalidade por padrão, bastando apenas que um agente opte por monitorar/focar as propriedades observáveis de um artefato monitor de saúde.

Essa característica possibilita a criação de relações de monitoramento por interesse, onde agentes do sistema podem acompanhar o estado de saúde de outros agentes. Esse tipo de relação torna-se bastante útil em casos que existe uma certa dependência entre agentes do sistema. Para exemplificar, tem-se a situação hipotética

[2]Maiores detalhes sobre a implementação desses mecanismos na linguagem Java podem ser vistos na referência (SIERRA; BATES, 2003), ou então a documentação da Oracle, no url: `https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html`

exposta na Figura 12, onde os agentes `agente_1` e `agente_2` tem acordado um contrato de prestação de serviços, onde o primeiro se compromete a entregar um serviço para a segundo. Nessa implementação considera-se que o agente `agente_1` é um agente que utiliza a arquitetura de TF (i.e., ele possui um artefato monitor de saúde associado) e que o `agente_2`, por ser diretamente interessado nos serviços prestados pelo seu contratado, opta por acompanhar o monitor de saúde atrelado ao `agente_1`. Caso ocorram faltas no agente contratado, um sinal de atualização de estado chegará na base de crenças de `agente_2`, gerando um evento ativador. Esse evento pode ser utilizado pelo programador como um gatilho para acionar planos de ações alternativos do agente `agente_2` como, por exemplo, refazer o contrato com um outro agente do sistema.

Figura 12 – Monitoramento por interesse entre agentes do sistema.



A relação de monitoramento por interesse é uma alternativa para que programadores possam criar planos com estratégias defensivas visando evitar que os agentes estejam totalmente expostos as faltas ocorridas em terceiros. Essa relação, no entanto, não fornece os mesmos privilégios de uma relação entre o agente monitorado e o agente de saúde. Neste segundo tipo de relação, somente a entidade de saúde tem a capacidade de executar protocolos para recuperar um agente que está em estado de erro.

3.3.1 Implementação do monitoramento por interesse no JaCaMo

A implementação dessa funcionalidade no JaCaMo se da de forma natural pois, devido as abstrações das propriedades observáveis que os agentes têm com os artefatos do ambiente, basta o agente interessado dar o comando de focar no artefato de saúde do agente tolerante a faltas. Um trecho de código em JaCaMo exemplificando essa relação pode ser visto na Figura 13. Nesse trecho, o agente inicia seu ciclo em `@g1`,

gerando o objetivo de executar o plano `@p1`, que inicialmente contrata uma prestação de serviço com o `agente_1` e depois estabelece a relação de monitoramento por interesse, focando no artefato monitor de saúde do contratado. Em `@e1` está descrito um plano que é acionado quando o agente recebe um evento ativador que o estado do `agente_1` passou para o valor "atrasado". Nesse momento, o agente coloca em pratica o plano alternativo `@p2`, que cancela o contrato com o `agente_1` e realiza um novo contrato com outro agente do sistema.

Figura 13 – Trecho de código para um agente que utiliza a relação de monitoramento por interesse.

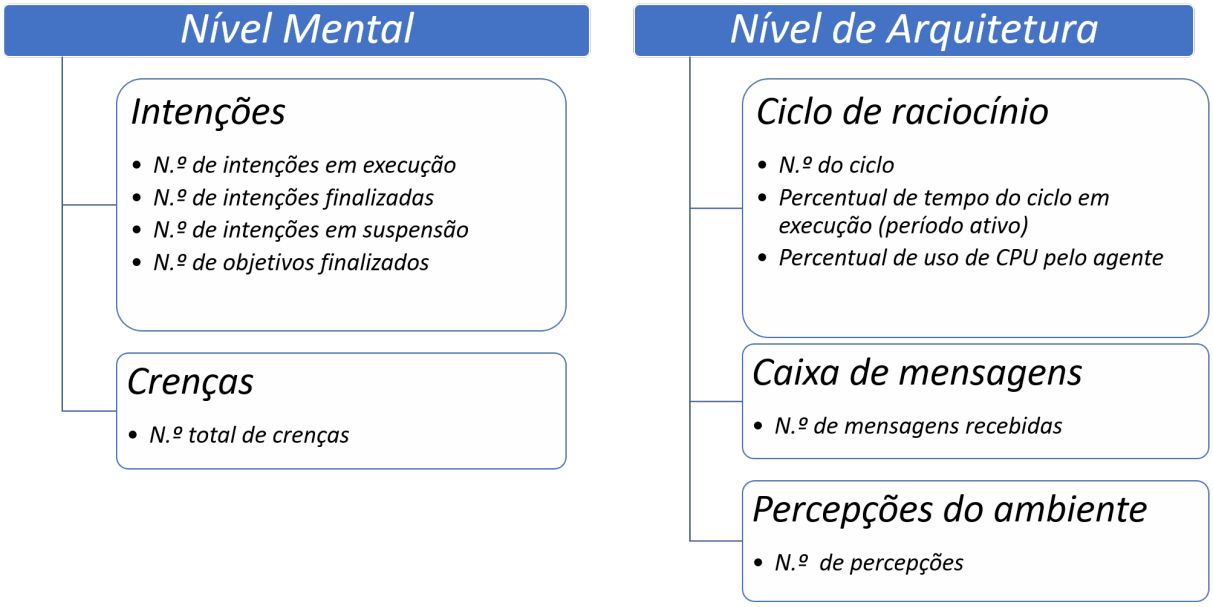
```
1       @g1
2       !iniciar.
3
4       @e1
5       +estado("late")[artifact_id(AId)] : contrato(N,AId)
6       <- !refazer_contrato.
7
8       @p1
9       +!iniciar
10      <- !contratar_servico("agente_1");
11         lookupArtifact("monitor_agente_1",AId);
12         focus(AId).
13
14      @p2
15      +!refazer_contrato
16      <- !cancelar_contrato("agente_1");
17         !contratar_serviço("agente_3");
18         lookupArtifact("monitor_agente_3",AId);
19         focus(AId).
```

3.4 MONITORAMENTO CONTÍNUO DO AGENTE

O monitoramento contínuo do agente é outra funcionalidade do modelo que visa fornecer informações adicionais a respeito do SMA. O propósito é fornecer, em tempo de execução, informações que dizem respeito as características internas do agente e de como ele interage com o ambiente. Neste modelo convencionou-se chamar de informações a *nível mental* e a *nível de arquitetura* do agente.

No nível mental estão as informações que dizem respeito a mente do agente e, como neste modelo adota-se agentes BDI, o nível mental se traduz em informações sobre as intenções do agente e as suas crenças. No nível de arquitetura estão as informações relacionadas a interação com o ambiente em que o agente esta inserido (i.e., sobre as mensagens recebidas e as percepções que ele tem sobre o mundo) e sobre o ciclo de raciocínio do agente. Os dados coletados para os dois níveis estão indicados na Figura 14.

Figura 14 – Informações no nível mental e de arquitetura.



Com o intuito de sintetizar todas as variáveis do nível mental e de arquitetura em um único ponto de análise, modelou-se um indicador chamado de *indicador de estresse*, que é calculado a partir da equação de média ponderada apresentada na Equação (1).

$$IE_t = \frac{Int_t \times IntW + BB_t \times BB_W + Evn_t \times Evn_W}{IntW + BB_W + Evn_W}, \tag{1}$$

onde Int_t é a número total de intensões em execução pelo agente no tempo t, BB_t é o número total de crenças que o agente possui e Evn_t representa o número total de eventos ocorridos entre $t-1$ e t, onde Evn_t considera-se o somatório do número de mensagens que o agente recebe e o número de percepções que o agente faz sobre o ambiente no período em questão. As variáveis de $IntW$, BB_W e Evn_W representam os pesos que podem ser atribuídos para as intensões, para as crenças e para os eventos, respectivamente. Os valores para esses pesos podem ser empiricamente atribuídos pelo programador, de acordo com a dinâmica que ele deseja dar ao seu indicador de estresse. Por exemplo, caso o programador deseje caracterizar o nível de estresse do seu agente de acordo com o ambiente em que ele vai ser exposto, pode-se atribuir um peso para os eventos relativamente maior em relação aos outros dois parâmetros.

Neste modelo, as informações do monitoramento contínuo também ficam disponíveis no artefato monitor de saúde do agente, na forma de propriedade observável. Entretanto, para não deixar essas informações mais “íntimas” do agente disponíveis para todos os agentes do sistema que estão monitorando o artefato, o modelo permite acesso desses dados apenas pelos agentes de saúde, que já são entidades com privi-

légios dentro do SMA. Para usar essas informações em seu benefício, o programador do SMA pode optar por personalizar o agente de saúde do sistema, usando informações do monitoramento contínuo para caracterizar situações em que o agente possa estar "sobrecarregado".

Como exemplo, pode-se imaginar que o programador, por já conhecer o ambiente em que o agente será inserido, deseja se proteger contra um sobrecarregamento de mensagens sendo enviadas para o agente. Para fazer isso, ele pode criar um plano de contenção quando for detectado que a fila de mensagens na caixa de mensagens do agente está constantemente crescendo. Nesse plano de contenção, o agente pode optar por apagar todas as mensagens ou filtrar mensagens de remetentes específicos.

Em outra situação hipotética, o agente está participando de um SMA que roda continuamente por períodos longos. Em um determinado momento, o agente começa a apresentar sinais de perda de desempenho nas tarefas que executa, chegando ao ponto que algumas tarefas não estão sendo entregues dentro do prazo. Alguns parâmetros do monitoramento contínuo podem indicar esse sintoma, como um taxa elevada de período ativo em seu ciclo de funcionamento, ou um valor excessivo para uso de CPU pelo agente, ou inclusive o próprio indicador de estresse, que apresenta valores cada vez mais altos. Nesses casos pode-se adotar medidas para "desestressar" o agente, como limpar ou reduzir a sua base de crenças e eliminar intenções suspensas.

Essa versatilidade fornece inúmeras alternativas ao desenvolvedor do SMA para tornar o sistema mais robusto, além de abrir possibilidades futuras, como a de implementar outras técnicas, como a distribuição tarefas para outros agentes do sistema, ou então realizar um processo de clonagem, como apresentado no trabalho de (SHEHORY *et al.*, 1998), que faz uma cópia de um agente para subdividir tarefas mais exigentes.

3.4.1 Implementação do monitoramento contínuo no JaCaMo

A implementação do monitoramento contínuo no JaCaMo se deu de duas maneiras: (i) transportando os indicadores do nível mental e de arquitetura para o artefato monitor de saúde e (ii) implementando uma interface gráfica que apresenta a evolução temporal das variáveis do monitoramento contínuo.

No artefato monitor de saúde, os indicadores do monitoramento contínuo ficam disponível na forma de uma lista chamada de `status_geral`, que pode ser acessada pelo agente de saúde como uma propriedade observável. Já a interface gráfica é uma ferramenta que o desenvolvedor pode optar por habilitar ou não. Ela possibilita uma análise da evolução do agente no sistema. A interface gráfica foi desenvolvida em JavaFX[3], que é uma plataforma para desenvolvimento de aplicações construídas

[3]Para mais informações sobre essa plataforma, consultar documentação em `https://openjfx.io/openjfx-docs`

em Java, que rodam em desktop, mobile e sistemas embarcados. Algumas telas da interface podem ser vistas nas Figuras 15, 16 e 17. As telas servem como monitores em tempo real dos parâmetros do nível mental, do nível de arquitetura e do indicador de estresse do agente, além de possibilitar um acompanhamento em gráficos para análise da evolução temporal desses parâmetros.

Figura 15 – Interface gráfica para monitoramento contínuo - Tela principal.

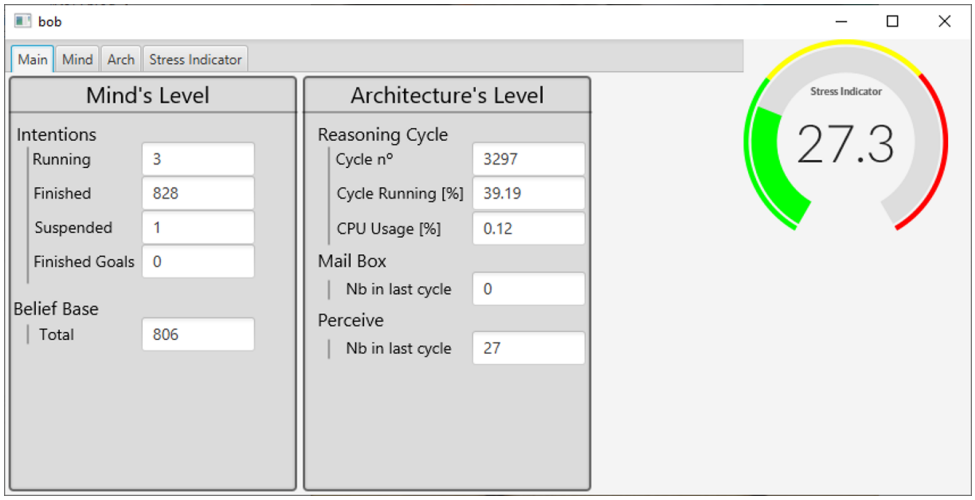
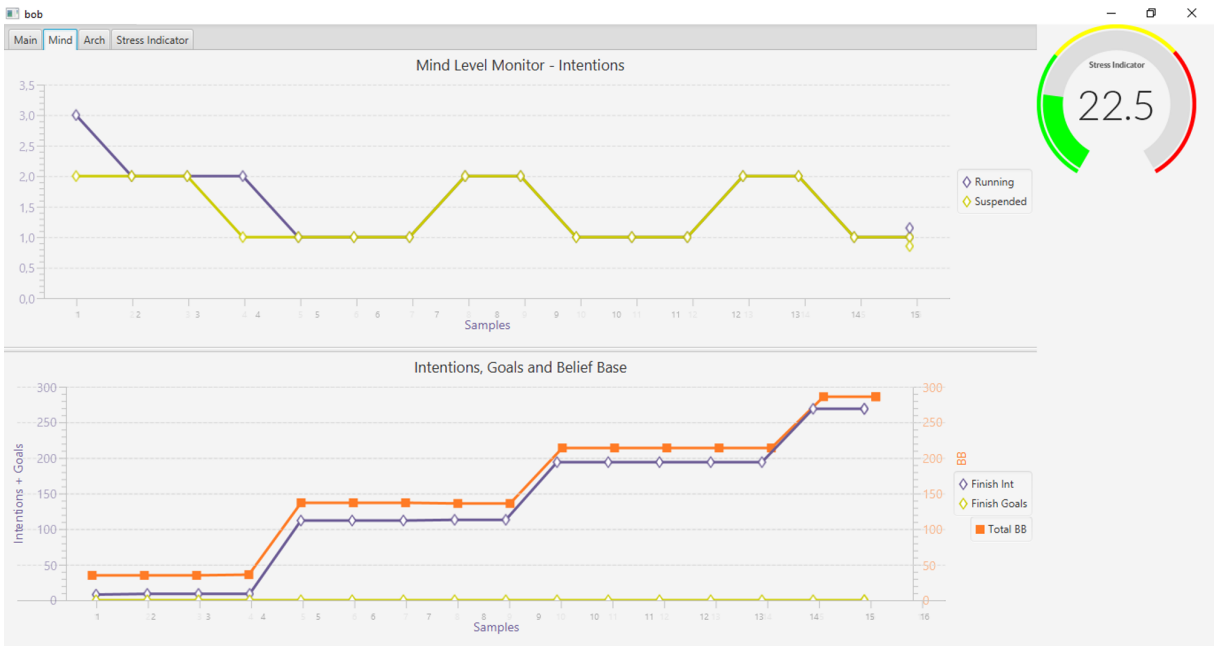


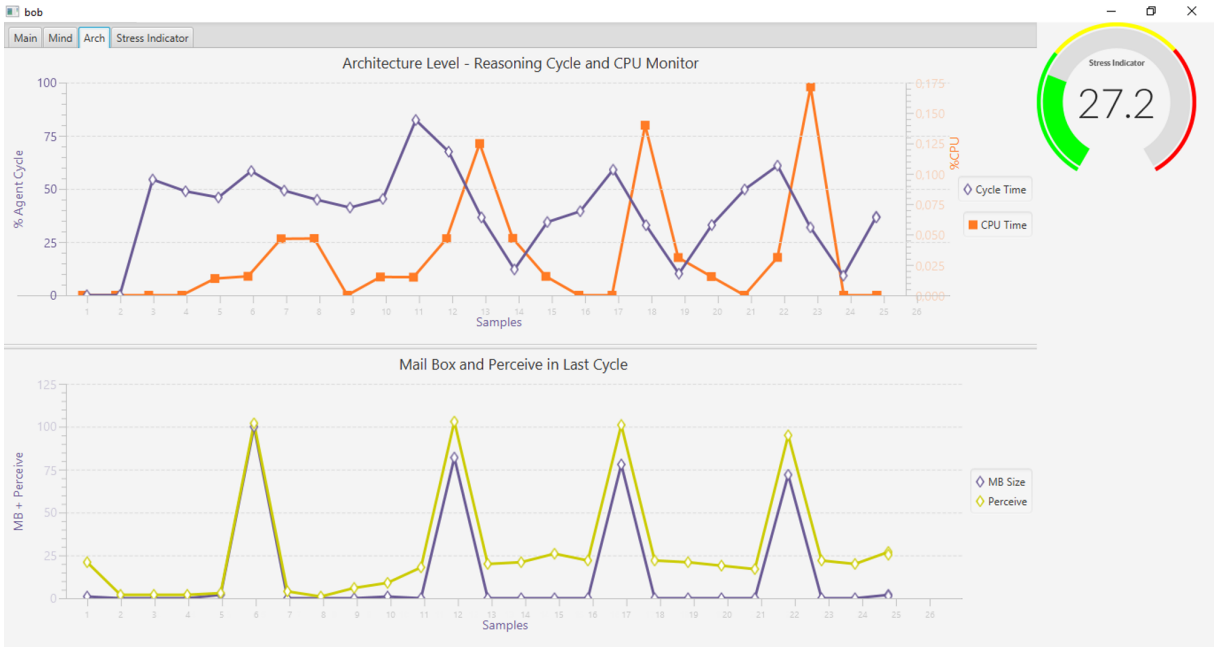
Figura 16 – Interface gráfica para monitoramento contínuo - Variáveis nível mental.



3.5 EXEMPLO ILUSTRATIVO

Para ilustrar o funcionamento dos recursos de TF, do monitoramento por interesse e do monitoramento contínuo, propõem-se um cenário bastante simples de um SMA com quatro agentes: o agente `pesquisador_1`, `pesquisador_2`, `pesquisador_3`

Figura 17 – Interface gráfica para monitoramento contínuo - Variáveis nível de arquitetura.



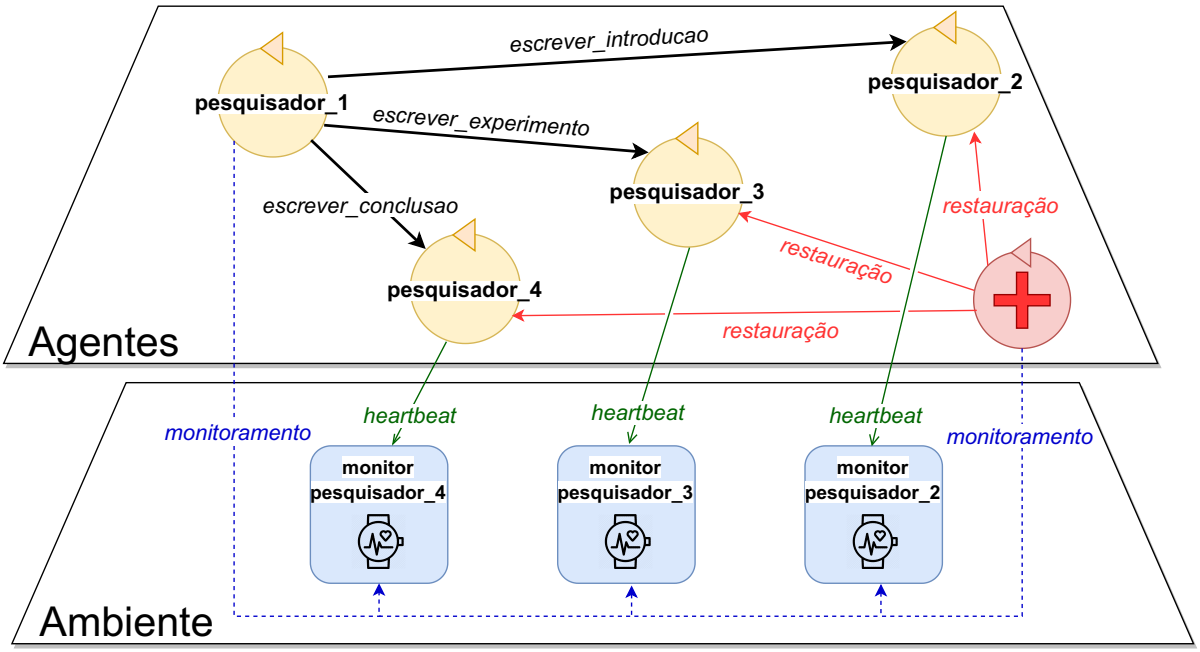
e `pesquisador_4`. O agente `pesquisador_1` é o líder de pesquisa em um projeto, ele tem como objetivo escrever um artigo sobre o projeto. Para concretizar esse objetivo ele põe prática o plano de dividir tarefas entre os outros pesquisadores, enviando mensagens para solicitar a execução de três tarefas: `escrever_introdução` para o agente `pesquisador_2`, `escrever_experimento` para o agente `pesquisador_3` e `escrever_conclusao` para o agente `pesquisador_4`.

Supõe-se que os agentes que estão realizando as tarefas foram configurados com a arquitetura que implementa o modelo de TF. Ao fazer isso, o modelo automaticamente cria a infraestrutura de TF, que adiciona ao SMA um agente de saúde e os artefato monitores de saúde vinculados aos agentes tolerantes a faltas do sistema. Supõe-se também que o agente `pesquisador_1`, pelo fato de ser diretamente interessado na saúde dos demais, decide por criar a relação de monitoramento por interesse e, dessa maneira, também passa a monitorar os artefatos monitores de saúde. Esse cenário ilustrativo com todas a relações existentes é representado na Figura 18.

Durante a execução das tarefas, imagina-se que o agente `pesquisador_4` sofra um falta que o impossibilita de entregar a tarefa. Em um cenário convencional, onde não existe nenhuma tratativa para tolerância a faltas, esse agente iria falhar no seu objetivo de escrever a conclusão. Somado a isso, o `pesquisador_1` também falharia no objetivo de entregar o relatório final completo e não saberia informar o que de fato ocorreu.

Ao incluir o modelo de TF no sistema o cenário se modifica. Após a ocorrência da falta, o monitor de saúde do `pesquisador_4` percebe a ausência dos sinais de *heartbeat* do agente e emite um sinal de emergência para o agente de saúde. Esse,

Figura 18 – Cenário ilustrativo do SMA, onde três agentes são configurados para serem tolerantes a faltas, cada um esta atrelado a um monitor de saúde. O agente de saúde e o agente pesquisador_1 monitoram os artefatos monitores de saúde. O primeiro possui essa relação por padrão e o segundo cria a relação por interesse.



por sua vez, inicializa o protocolo de restauração, que reestabelece o `pesquisador_4` conforme os estados salvos no último processo de *checkpoint*, possibilitando que ele finalize a execução da tarefa interrompida e permitindo que o sistema entregue o documento que foi especificado inicialmente, mesmo com a ocorrência da falta durante a execução.

Pelo lado do agente `pesquisador_1`, devido a relação de monitoramento por interesse existente com os outros agentes, ele receberá sinais informativos indicando que o agente `pesquisador_4` entrou em estado de erro e, posteriormente, ele também receberá a informação que o agente foi restabelecido. Esses sinais percebidos pelo `pesquisador_1` podem ser utilizados para programar eventos ativadores de planos de contingência. Como exemplo, pode-se citar os seguintes planos: (i) caso um agente entrar em estado de erro, repassar para outro agente a tarefa que estava alocada com ele. Ou então (ii) caso um agente entregue o documento, porém esse agente apresentou faltas durante a execução da tarefa, criar um plano revisão para esse documento.

Outro recurso do modelo que pode ser explorado nesse exemplo é o monitoramento contínuo. Neste caso o programador pode optar por customizar os planos do agente de saúde no SMA para que ele passe a reagir conforme as mudanças referentes nos parâmetros do nível mental e da arquitetura do agente, criando assim

estratégias de contingência personalizados. Abaixo estão listados algumas sugestões de planos possíveis:

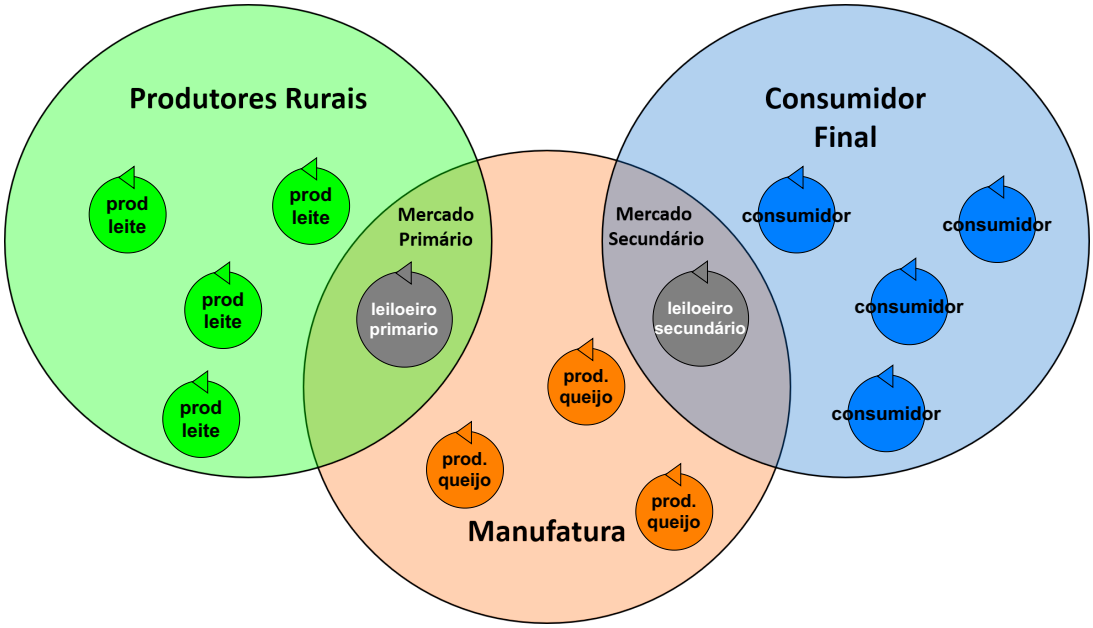
- Caso o nível de estresse do agente monitorado atingir um certo valor, restaurar o agente para o estado de um *checkpoint* mais antigo;
- Caso o número de intenções do agente esteja crescendo a uma taxa elevada (e.g., o agente está recebendo mais tarefas do que consegue concluir), manter apenas as intenções que foram enviadas por um agente específico e eliminar o restante (e.g., manter apenas o que foi solicitado pelo pesquisador líder).
- Caso o agente esteja com um valor muito elevado na base de crenças, eliminar as crenças mais antigas e manter apenas as mais novas (i.e., esquecer as informações muito antigas);
- Caso o agente monitorado esteja com um número muito elevado de mensagens na sua caixa de e-mails, filtrar as mensagens de remetentes importantes e deletar o restante.

A implementação de algumas dessas sugestões pode favorecer a dependabilidade do SMA, pois alguns comportamentos indesejados pelo desenvolvedor podem ser detectados e tratados com antecedência, podendo inclusive evitar que ocorram faltas futuras. O monitoramento contínuo é uma ferramenta que pode trazer inúmeros benefícios ao SMA, ficando sob a responsabilidade do desenvolvedor utiliza-lá da maneira que mais lhe parece positiva.

4 AVALIAÇÃO DO MODELO

Para avaliar o modelo de TF proposto, elaborou-se um SMA onde dois mercados virtuais operam negociações em formato de leilão duplo. Nesse cenário, ilustrado pela Figura 19, os agentes do sistema são divididos em três grupos: os produtores rurais, os agentes da indústria de manufatura e os consumidores finais. Cada agente desses grupos pode operar nos mercados de compra e venda de produtos primários ou secundários, dependendo de seu interesse. Cada mercado é gerenciado por um agente leiloeiro, responsável pelo controle do leilão dos produtos ofertados.

Figura 19 – Sistema Multiagente para operação dos mercados virtuais.



4.1 PROCESSO DE LEILÃO DUPLO

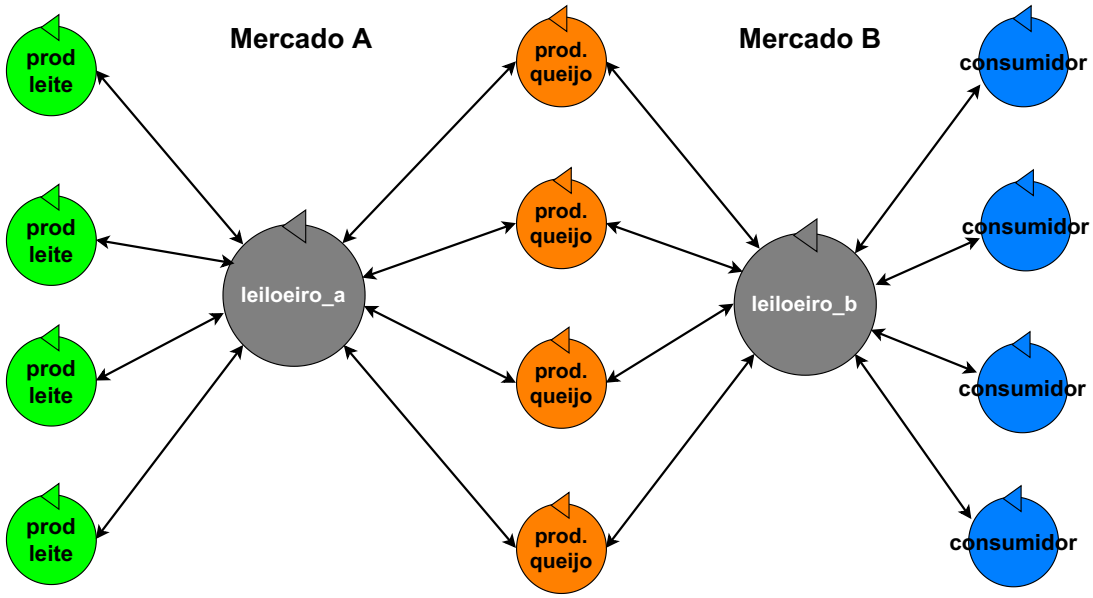
O leilão duplo é um processo de câmbio bastante utilizado em diversos mercados, como ações, commodities e financeiros (OUYANG; WANG; YANG, 2004). A natureza deste tipo de leilão se difere dos leilões de oferta, onde vendedores ofertam um bem que um único comprador deseja comprar, e dos leilões de demanda, onde diversos compradores fazem lances por um bem de um único comprador. Nos leilões duplos, vendedores e compradores podem submeter suas propostas por um produto de forma simultânea.

Um leilão também pode ser classificado de acordo com a maneira que os lances pelo produto ocorrem. Existem duas formas básicas de se fazer os lances: aberta ou fechada. Na forma aberta, o preço do item é determinado dinamicamente e todos participantes têm acesso ao valor do bem conforme o leilão evolui. Na forma fechada

os lances são entregues ao leiloeiro, tradicionalmente por meio de envelopes fechados, e vencem os participantes que fizerem as melhores propostas.

Nesse cenário de testes, implementou-se leilões duplos e fechados, que ocorrem simultaneamente. No mercado A ocorrem as negociações de produtos primários, onde o agente *leiloeiro_a* é responsável por gerenciar os lances de produtores rurais de leite e de produtores de queijo. No mercado B operam as negociações de produtos secundários, onde o agente *leiloeiro_b* administra as negociações entre os produtores de queijo e os consumidores finais. A Figura 20 ilustra a dinâmica do sistema.

Figura 20 – Leilão duplo implementado.



4.2 O MERCADO VIRTUAL

A dinâmica do SMA implementado segue um protocolo orquestrado pelos agentes leiloeiros. O funcionamento do leilão e as especificidades de cada agente serão aprofundadas a seguir.

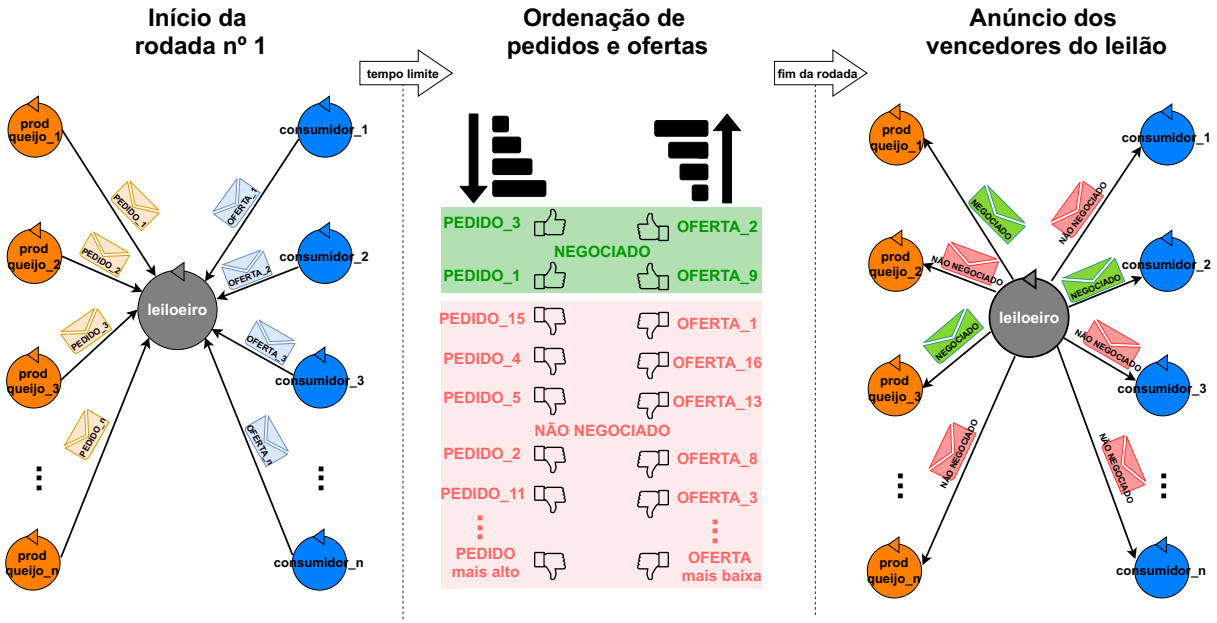
4.2.1 Protocolo do mercado

Os leilões duplos desse SMA operam por um determinado número de rodadas. Cada rodada se inicia com a abertura do leilão, que fica nesse estado por um tempo estipulado, recebendo propostas de vendedores e compradores. Ao término desta etapa, o responsável pelo leilão ordena os pedidos dos vendedores em ordem crescente de valor (do agente que pediu menos até o que pediu mais pelo produto) e as ofertas dos compradores em ordem decrescente de valor (do agente que ofertou mais até o que ofertou menos pelo produto). Desta forma, os vendedores que estão dispostos a

vender mais barato os seus produtos e os compradores que ofereceram mais dinheiro terão mais chance de realizar seus negócios nessa rodada.

Para avaliar quais propostas podem se concretizar em negócios, as ofertas ordenadas são comparadas com os pedidos, caso o preço da oferta seja igual ou superior ao pedido, o negócio é concretizado. Ao final da rodada, todos os participantes são informados se tiveram sucesso ou não na sua proposta, os vencedores são retirados do leilão e não podem mais participar dele nas próximas rodadas. Os perdedores, por sua vez, são liberados para enviar uma nova proposta, caso tenham interesse. Esse processo se repete até que o limite de rodadas seja atingido ou até o momento que não existam mais compradores ou vendedores. A Figura 21 exemplifica o funcionamento desse protocolo durante uma rodada.

Figura 21 – Protocolo do leilão duplo.



4.2.2 Agente leiloeiro

O agente leiloeiro é o responsável pelo controle dos leilões duplos. Entre suas principais funções estão: iniciar o leilão, avisar todos os agentes do sistema que o leilão iniciou, receber e registrar as propostas recebidas dentro do tempo limite, verificar quais propostas são aptas para se concretizar em negócios e avisar os participantes quais propostas foram vencedoras.

Para auxiliar nessas tarefas, o agente leiloeiro utiliza um artefato chamado de *mercado*. Esse artefato funciona como uma ferramenta de auxilio para o agente, realizando os processos de inicialização de novos leilões, de novas rodadas, cadastro de ofertas e pedidos e ordenação das propostas. As principais ações do agente leiloeiro no artefato mercado são exibidas no diagrama de caso de uso exibido na Figura 22.

Para interação com o usuário, elaborou-se uma tela onde é possível acompanhar a evolução das propostas de cada rodada do leilão. Um exemplo dessa tela é apresentada na Figura 23.

Figura 22 – Diagrama de caso de uso para o artefato mercado.

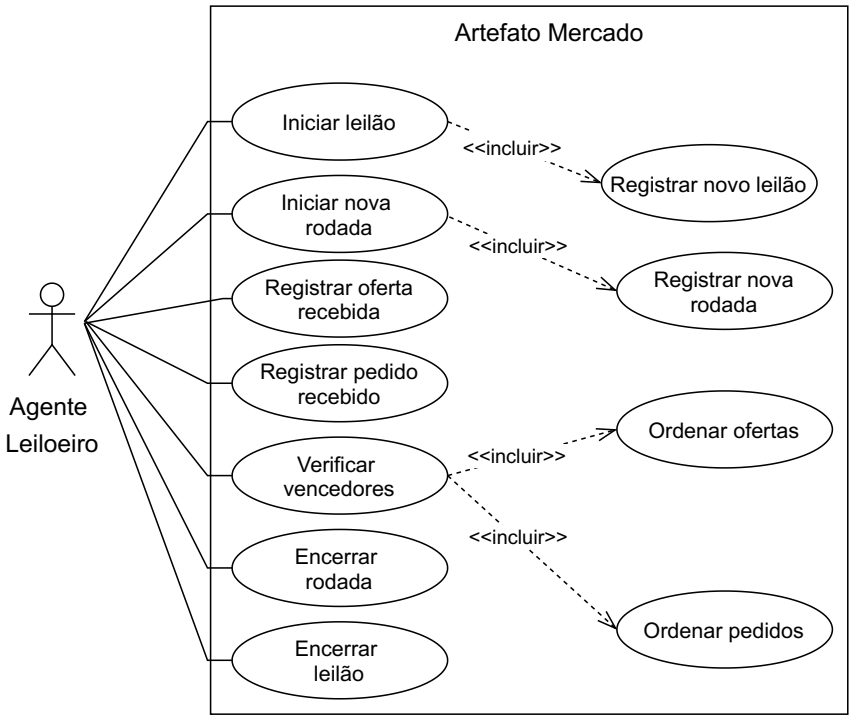
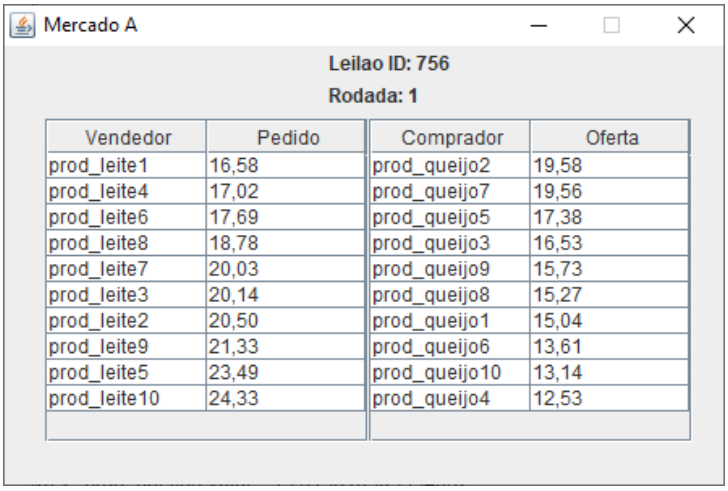


Figura 23 – Tela de interface para acompanhamento dos leilões.



Vendedor	Pedido	Comprador	Oferta
prod_leite1	16,58	prod_queijo2	19,58
prod_leite4	17,02	prod_queijo7	19,56
prod_leite6	17,69	prod_queijo5	17,38
prod_leite8	18,78	prod_queijo3	16,53
prod_leite7	20,03	prod_queijo9	15,73
prod_leite3	20,14	prod_queijo8	15,27
prod_leite2	20,50	prod_queijo1	15,04
prod_leite9	21,33	prod_queijo6	13,61
prod_leite5	23,49	prod_queijo10	13,14
prod_leite10	24,33	prod_queijo4	12,53

4.2.3 Agente produtor de leite

O agente produtor de leite é um agente que periodicamente produz uma unidade de leite e, a partir desse momento, fica aguardando o início de algum leilão pelo produto. Durante a rodada do leilão, o agente produtor envia o valor que ele deseja receber

pelo seu produto. Esse valor é escolhido dentro de uma faixa de preço de mercado estipulado.

Ao final da rodada do leilão, o agente produtor será informado se algum comprador ofereceu um valor igual ou superior a sua solicitação. Caso nenhum negócio tenha sido concretizado nessa rodada, o agente produtor toma a estratégia de baixar o seu preço de venda e aguardar o início da rodada seguinte desse leilão. Essa estratégia é adotada repetidamente até que exista algum comprador para o seu produto ou o leiloeiro encerre o leilão. Todos os preços escolhidos pelos agentes do sistema são obtidos de forma aleatória.

4.2.4 Agente consumidor final

O agente consumidor se encontra na outra ponta do sistema. Periodicamente esse agente tem a intenção de comprar uma certa quantidade de queijo para seu consumo próprio. Nesse instante ele aguarda iniciar um leilão deste produto, quando isso ocorre, ele oferece um valor dentro de uma faixa de mercado. Caso a sua oferta não seja alta o suficiente para adquirir o produto, ele adota a estratégia de acrescentar um valor na sua proposta para a próxima rodada do leilão.

4.2.5 Agente produtor de queijo

O agente produtor de queijo faz o meio de campo entre o produtor de leite e o consumidor de queijo. Esse agente possui características tanto de comprador como de vendedor. A sua atuação como comprador se da quando ele negocia com o produtor de leite, ofertando um valor dentro de uma faixa de preço de mercado. Já a atuação como vendedor se da quando o agente oferece seu produto para o agente consumidor, pedindo por ele um preço dentro de uma faixa de mercado. Esse agente funciona sob demanda, produzindo queijo a medida que consegue adquirir o leite e comprando mais matéria prima conforme consegue vender o queijo produzido.

4.3 GERADOR DE FALTAS

Para que seja possível avaliar o comportamento do SMA em conjunto com o modelo de TF elaborado, é necessário que ocorram faltas durante a execução do sistema. Dessa forma, insere-se no SMA um agente com a função de submeter o sistema a faltas periódicas. O *agente gerador de faltas* opera de maneira que sejam geradas faltas nos outros agentes do sistema, essas faltas são do tipo faltas de quebra (*crash*), ou seja, elas fazem com que o agente pare de operar por completo.

Nas simulações realizadas neste trabalho, foi considerado que o agente gerador de faltas é capaz de eliminar os agentes participantes do leilão (i.e. produtores de leite, produtores de queijo e consumidores), deixando livre de faltas os agentes leiloeiros.

Essa escolha foi feita devido a dois fatores principais: (i) os agentes participantes dos leilões são mais numerosos dentro do SMA e (ii) para que as simulações possam ser comparáveis entre si, devido ao impacto da ocorrência de faltas nos agentes participantes serem muito semelhantes ou equivalentes.

Os agentes leiloeiros são agentes com comportamento muito específico dentro do SMA. Esse fato faz com que a ocorrência de faltas nessa classe de agente tenha um impacto diferente no sistema do que as faltas ocorridas nos agentes participantes dos leilões. Como as faltas são distribuídas aleatoriamente entre os agentes, para tornar mais justa as comparações entre simulações, opta-se por distribuir as faltas entre os agentes que tenham impacto semelhantes entre si. Para ilustrar essa situação, tem-se um cenário hipotético onde as faltas são distribuídas aleatoriamente entre todos os participantes do SMA (incluindo os agentes leiloeiros). Nesse cenário, tem-se uma simulação `A`, em que todas as faltas ocorreram em agentes participantes do leilão (i.e. nenhuma falta ocorreu em agentes leiloeiros, apenas em produtores ou consumidores) e uma simulação `B`, onde diversas faltas foram distribuídas para os agentes leiloeiros. Nesse cenário, a comparação de desempenho entre as simulações `A` e `B` será comprometida pelo fator de impacto diferente entre agentes do sistema.

O protocolo de operação do agente gerador de faltas consiste em, de forma periódica, pegar a lista de todos os agentes que estão participando de um leilão naquele instante e escolher aleatoriamente um desses agentes para que este seja eliminado do sistema. Ao fazer isso, todas as ofertas de compra, pedidos de venda e negociações que o agente tenha feito não serão efetuadas, impondo assim uma série de dificuldades ao sistema (e.g. erros em trocas de mensagens, não conclusão em negociações do leilão, entre outros). Esse processo se repete periodicamente dentro de um período fixo de tempo.

4.4 CENÁRIOS DE TESTES

Como forma de analisar o comportamento do SMA atuando em conjunto com o modelo de TF, implementou-se diferentes cenários, cada um com diferentes parâmetros que configuram a dinâmica do leilão, do modelo de TF e da geração de faltas. O Quadro 3 identifica os principais parâmetros utilizados nas simulações.

Os parâmetros do leilão modelam a dinâmica do sistema como um todo, definindo qual o número de agentes participantes, qual o número de rodadas que um mesmo leilão pode ter e durante quanto tempo os leilões podem ocorrer.

Os parâmetros do modelo de TF definem como se da a dinâmica do modelo em si. Os principais parâmetros que se pode alterar são o intervalo de vigilância do monitor de *heartbeat* e qual o intervalo de tempo entre criação de pontos de restauração de cada agente tolerante a faltas.

Por fim, para as simulações também é possível alterar a dinâmica de como

Quadro 3 – Quadro de parâmetros para configuração das simulações.

Leilão	Número de participantes Número de rodadas Duração do leilão
TF	Intervalo para monitor de heartbeat Intervalo entre criação de checkpoints
Gerador de Faltas	Intervalo entre ocorrência de faltas Duração do teste com gerador de faltas ativo

ocorrem as faltas no SMA. Para fazer isso, altera-se os parâmetros do agente gerador de faltas, configurando qual o período do teste em que a geração de faltas está ativa e qual o intervalo fixo entre a ocorrência de uma falta e outra.

4.4.1 Cenário 1 - Simulação do SMA na presença do gerador de faltas

O primeiro cenário apresenta uma comparação de como o SMA se comporta em três condições distintas. A primeira simulação executa o sistema em condições ideais, onde não existem ocorrências de faltas durante toda a operação. Na segunda simulação, é introduzido no sistema o agente gerador de faltas, que passa a inserir periodicamente faltas nos agentes participantes do leilão. Na terceira simulação o cenário faltoso é mantido, mas adiciona-se o modelo de TF. Os parâmetros de teste são resumidos no Quadro 4.

Quadro 4 – Quadro de parâmetros para cenário 1.

Leilão	Número de participantes Número de rodadas Duração do leilão	2 leiloeiros 15 produtores 20 manufatureiros 15 consumidores 5 12min
TF	Intervalo para monitor de heartbeat Intervalo entre checkpoints	1s 1s
Gerador de Faltas	Intervalo entre faltas Período com gerador de faltas ativo	5s 10min

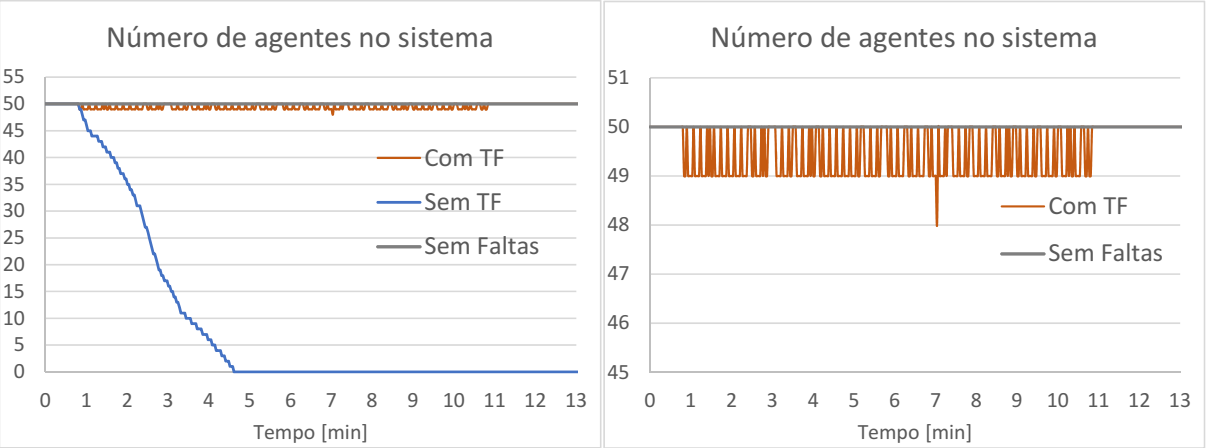
Como primeira métrica para avaliação deste cenário, analisa-se a quantidade de agentes presentes no sistema ao longo do tempo. A contabilização é feita apenas dos agentes que participam dos leilões, ou seja, ao iniciar o SMA existem cinquenta agentes aptos a comprar e vender produtos nos leilões. A Figura 24 apresenta o total de agentes participantes de leilões ao longo do tempo em uma das simulações.

Na Figura 24a observa-se que na condição ideal o número de agentes se mantém em cinquenta, conforme esperado. Na condição em que adiciona-se o gerador de faltas no sistema sem TF, o número de agentes participantes decresce conforme

Figura 24 – Avaliação do número de agentes participantes do leilão - Cenário 1.

(a) Número de agentes presentes no sistema.

(b) Número de agentes presentes no sistema, comparação cenário sem faltas e cenário com faltas e modelo de TF atuando.



as faltas vão ocorrendo, chegando na situação em que não existem mais agentes para participantes dos leilões. Quando adiciona-se o modelo de TF, o sistema inicia os protocolos para tentar manter os agentes em funcionamento. A Figura 24b da enfase na comparação entre o sistema sem faltas e o sistema com TF, observa-se que o modelo de TF consegue manter o SMA operando com o número de agentes entre 48 e 50, o que demonstra que o modelo consegue restaurar os agentes com sucesso. A diminuição na quantidade de agentes operando ocorre devido ao tempo que o modelo de TF leva para detectar a ocorrência da falta, iniciar o protocolo de recuperação e restaurar o agente no SMA.

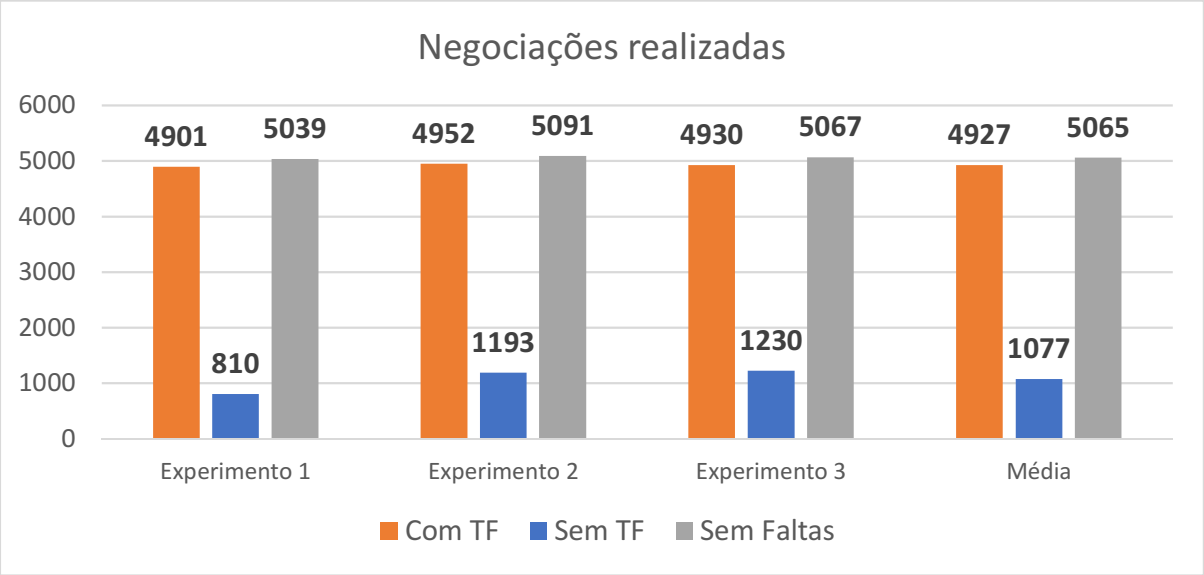
Como consequência para a redução dos agentes que participam dos leilões, espera-se uma diminuição no número de negociações realizadas durante a execução do SMA. Considera-se como negociação toda oferta de compra e venda que é feita por algum agente do sistema, não importando se essa oferta se concretiza em uma venda ou não. A Figura 25 mostra o número total de negociações realizadas para três repetições do experimento, onde cada repetição é executada utilizando os mesmos parâmetros (cada repetição está identificada no gráfico como “experimentos 1, 2 e 3”).

Pelas simulações realizadas, percebe-se uma proximidade entre o número de negociações do cenário ideal e do cenário faltoso com TF. Na média dos experimentos, o número de negociações do cenário com TF ficou em 97,3% das negociações de um cenário ideal.

4.4.2 Cenário 2 - Simulação com variação no intervalo entre faltas

O cenário 2 explora o impacto da frequência de ocorrência das faltas em um sistema com o modelo de TF. Desse modo, os parâmetros referentes a dinâmica do

Figura 25 – Total de negociações realizadas no cenário 1. Dados para três repetições e média dos resultados.



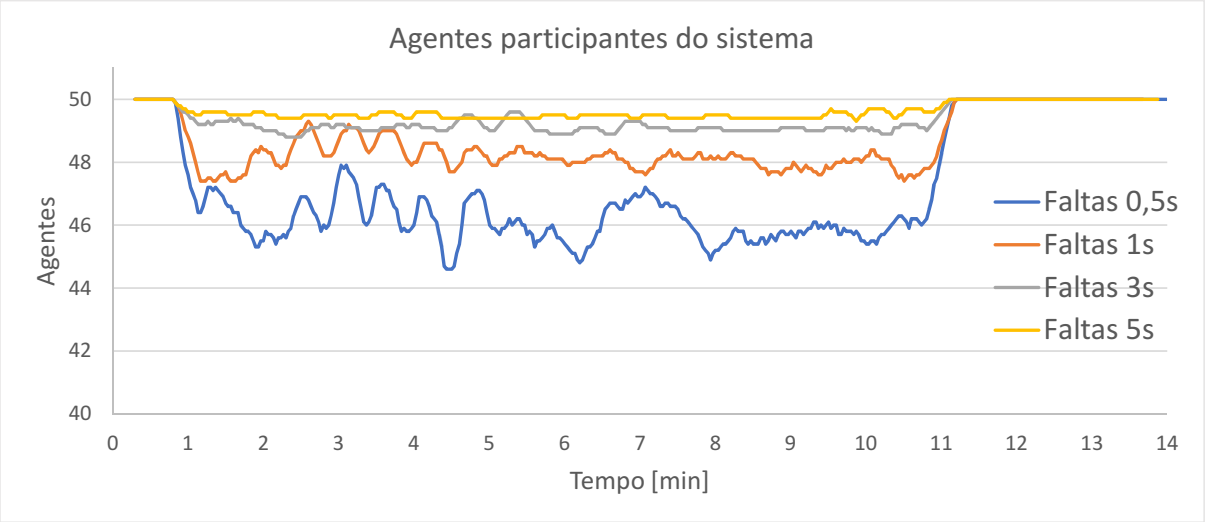
modelo de TF são fixados e varia-se o intervalo para ocorrência de faltas do gerador de faltas entre 0,5s e 5 segundos, conforme apresenta o Quadro 5.

Quadro 5 – Quadro de parâmetros para cenário 2.

Leilão	Número de participantes Número de rodadas Duração do leilão	2 leiloeiros 15 produtores 20 manufatureiros 15 consumidores 5 12min
TF	Intervalo para monitor de heartbeat Intervalo entre checkpoints	1s 1s
Gerador de Faltas	Intervalo entre faltas Período com gerador de faltas ativo	0,5s - 1s - 3s - 5s 10min

Novamente são analisados a evolução do número de agentes participando dos leilões ao longo das simulações e o número total de negociações neste período. Pela Figura 26 é possível verificar que, apesar da frequência mais elevada de faltas, o modelo mantém a capacidade de restaurar os agentes no sistema, como pode-se observar o número total de agentes retornando ao mesmo valor inicial no instante em que o gerador de faltas é suspenso (próximo aos 11 minutos de execução). A diferença no número de agentes operando nos cenários onde as faltas ocorrem com maior frequência se da devido a dinâmica da restauração dos agentes ser mais lenta do que a dinâmica da ocorrência das faltas. Essa disparidade faz com que o modelo de TF não consiga reestabelecer o agente faltoso antes da ocorrência de uma nova falta.

Figura 26 – Avaliação do número de agentes participantes do leilão no cenário 2. Para melhor leitura do gráfico, as curvas apresentadas foram geradas aplicando um filtro de média móvel com janela de 10 amostras.



Quando analisa-se o número total de negociações para as quatro condições, obtém-se números muito próximos, conforme indica a Figura 27. Os dados indicam uma tendência para um maior número de negociações nos cenários em que a dinâmica das faltas é mais próxima ou mais lenta que a dinâmica do modelo de TF. Entretanto, deve-se levar em conta que esses experimentos são feitos em sistemas que possuem a natureza autônoma dos agentes e, dessa maneira, a métrica do número total de negociações pode não ser a melhor maneira de comparar as condições. Para exemplificar, pode-se pensar em um cenário onde um leilão é iniciado, mas o agente opta por não enviar nenhuma proposta naquele momento por motivos próprios (e.g. o agente acabou de negociar uma quantidade de leite e não tem a intenção de comprar mais antes de vender o seu estoque de queijo).

Uma maneira mais apropriada para medir o quanto o sistema foi capaz de tolerar as faltas introduzidas é contar quantas falhas surgiram ao longo da operação. Para fazer isso, deve-se primeiro entender o que pode ser interpretado como uma falha para o experimento. Voltando para a definição da palavra, uma falha é um desvio da especificação original do sistema que pode ser observada no nível do usuário. Nesse experimento, uma falha é contabilizada de duas maneiras: (i) quando uma negociação é oficializada, mas o agente vendedor não confirma a entrega do produto ofertado e/ou o agente comprador adquire o produto, mas não confirma o recebimento deste. E (ii), quando os agentes participantes do leilão tem suas ofertas rejeitadas mas, por motivos de falta, não enviam novas propostas na próxima rodada do leilão (considerando que ainda existam rodadas deste leilão). A Figura 28 apresenta o fluxo pelo qual uma oferta de um agente passa, identificando quais são os dois pontos que podem ser contabilizados como uma falha para o sistema.

Figura 27 – Total de negociações realizadas no cenário 2. Dados para três repetições e média dos resultados.

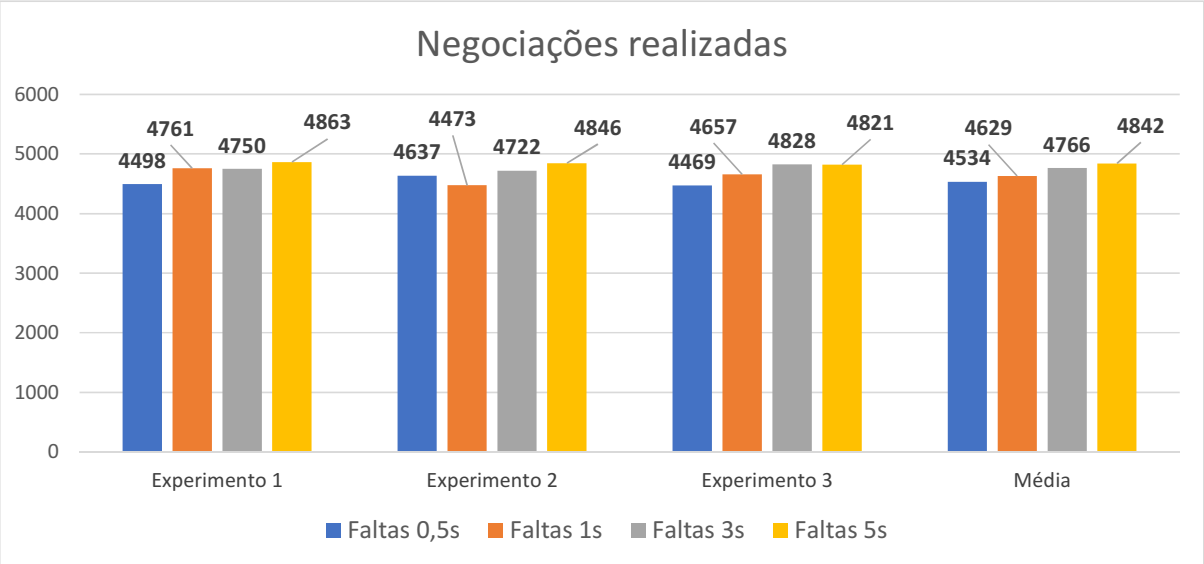
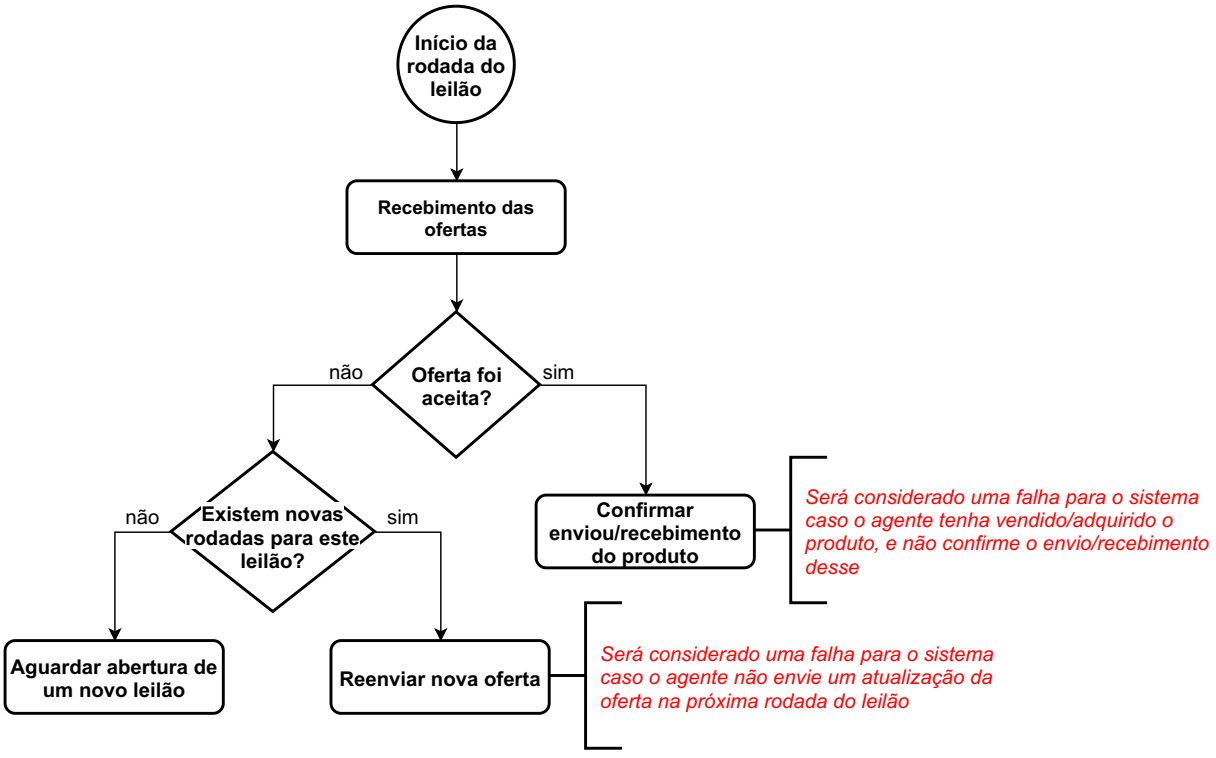


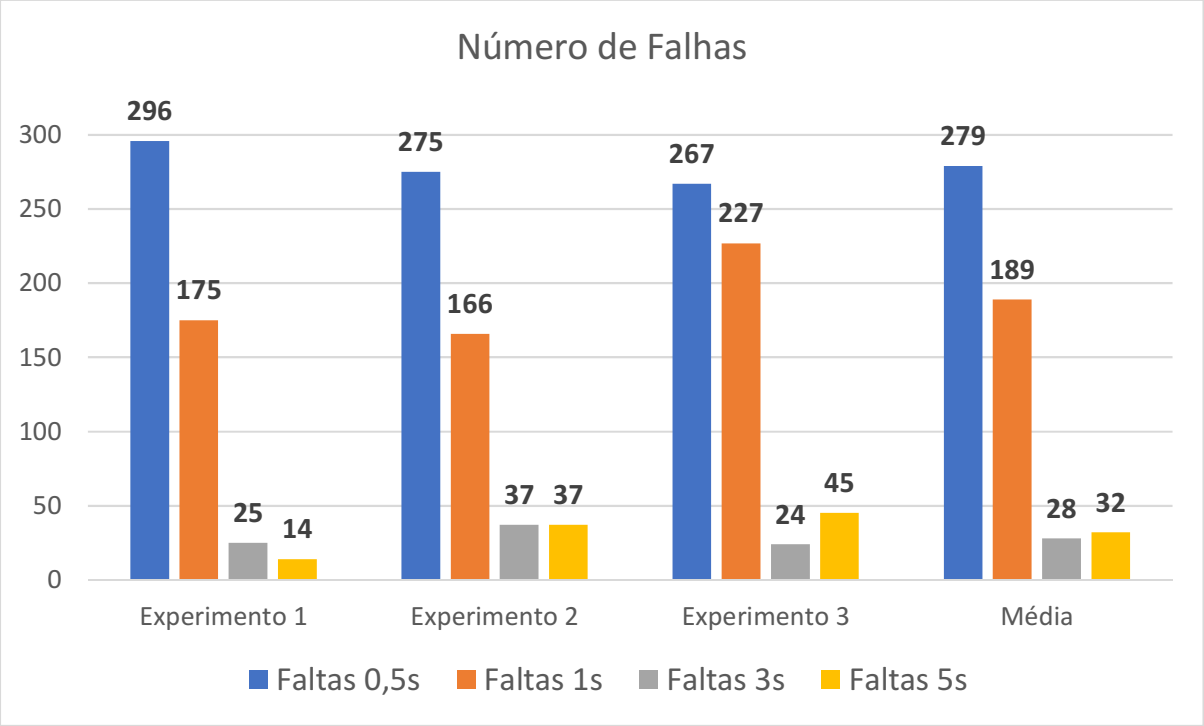
Figura 28 – Fluxo para uma oferta de um agente do SMA e apontamento de falhas no processo.



Dessa maneira, as falhas ocorridas durante os três experimentos e a média entre os experimentos são apresentadas na Figura 29. Nessa análise, pode-se observar que o sistema com frequências altas de faltas terão, como consequência, mais falhas perceptíveis. Na medida em que a dinâmica da ocorrência de faltas se aproxima da dinâmica do modelo de TF, o surgimento de falhas diminui consideravelmente, chegando

a um nível praticamente estável a partir de um certo ponto.

Figura 29 – Número de falhas identificadas durante a execução de três experimentos e média das falhas - Cenário 2.



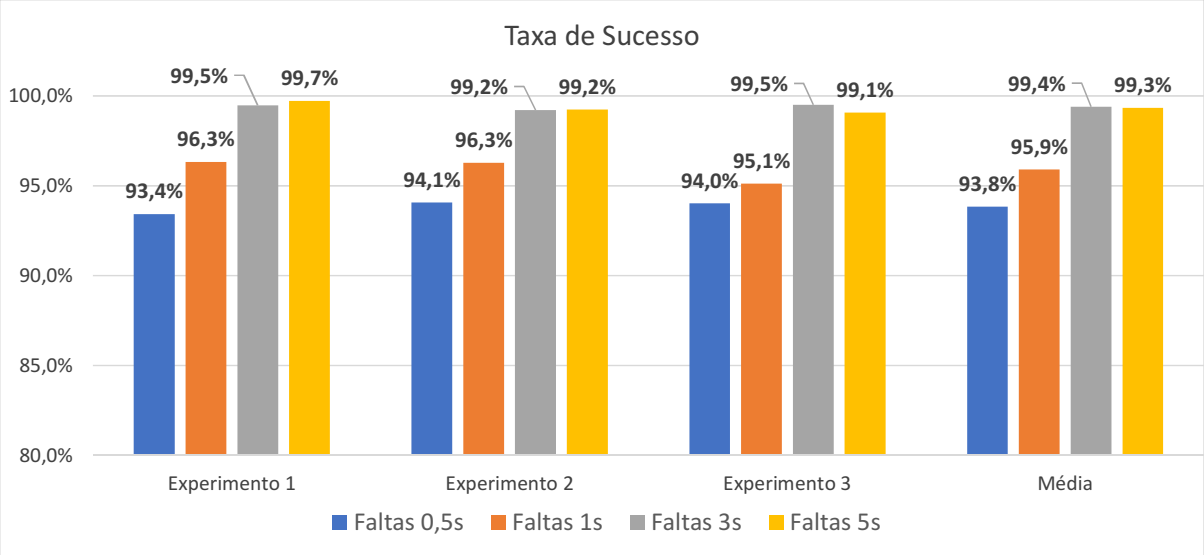
Essa análise do número de falhas também pode ser apresentada em números relativos, onde considera-se o número total das negociações e o número de negociações em que ocorreram falhas. Essa métrica é denominada *taxa de sucesso*, e é calculada divindo o número total de negociações com falhas pelo total de negociações do sistema. A Figura 30 apresenta as taxas de sucesso dos experimentos. Observa-se que as simulações com dinâmica de falta mais próximas da dinâmica do modelo de TF apresentam médias de sucesso em mais de 99% das negociações do SMA.

4.4.3 Cenário 3 - Simulação com variação no intervalo do monitor de *heartbeat*

O cenário 3 investiga como o sistema se comporta quando a dinâmica do modelo de TF é alterada. Para fazer isso, varia-se o parâmetro do intervalo do monitor de *heartbeat* entre 250ms e 1 segundo. Essa mudança tem como consequência uma variação do tempo que o modelo leva para detectar a ocorrência de uma falta no sistema, intervalos mais curtos permitem que o sistema identifique um agente em estado de erro antes mas, como consequência, demandam mais tempo de execução do agente, pois esse passa a enviar seus sinais de *heartbeat* com uma frequência mais elevada. O Quadro 6 apresenta os parâmetros utilizados neste cenário.

Novamente utiliza-se como métrica de comparação o número total de falhas e a taxa de sucesso das negociações do sistema. A Figura 31 apresenta uma diminui-

Figura 30 – Taxa de sucesso para três experimentos e média das taxas - Cenário 2.



Quadro 6 – Quadro de parâmetros para cenário 3.

Leilão	Número de participantes Número de rodadas Duração do leilão	2 leiloeiros 15 produtores 20 manufatureiros 15 consumidores 5 12min
TF	Intervalo para monitor de heartbeat Intervalo entre checkpoints	250ms - 500ms - 750ms - 1s 1s
Gerador de Faltas	Intervalo entre faltas Período com gerador de faltas ativo	1s 10min

ção no número de falhas conforme diminui-se o intervalo do monitor de *heartbeat*. A Figura 32 mostra que os modelos de TF que utilizaram intervalos de 250ms e 500ms apresentaram uma taxa de 98,4% de sucesso nas negociações do sistema.

Tanto o cenário 2 como o cenário 3 demonstram que existe uma relação direta entre a taxa de sucesso do sistema com as dinâmicas do modelo de TF e da ocorrência de faltas. No cenário 3, a alternativa para elevar a taxa de sucesso implica que os agentes executem o plano de envio de sinais periódicos com uma frequência maior. Isso ocorre pois o modelo de TF estipula que o plano de envio do sinal de *heartbeat*, que cada agente tolerante a faltas executa, seja executado em um intervalo cinco vezes menor do que o intervalo configurado no monitor de *heartbeat*. Essa configuração garante que, em situações onde não existe a ocorrência de faltas, o agente consiga enviar sinais de vida dentro do período estipulado pelo monitor. Por outro lado, quando o monitor é configurado para operar com períodos muito curtos, isso implica que o plano de *heartbeat* seja executado em períodos ainda mais curtos, o que pode

Figura 31 – Número de falhas identificadas durante a execução de três experimentos e média das falhas - Cenário 3.

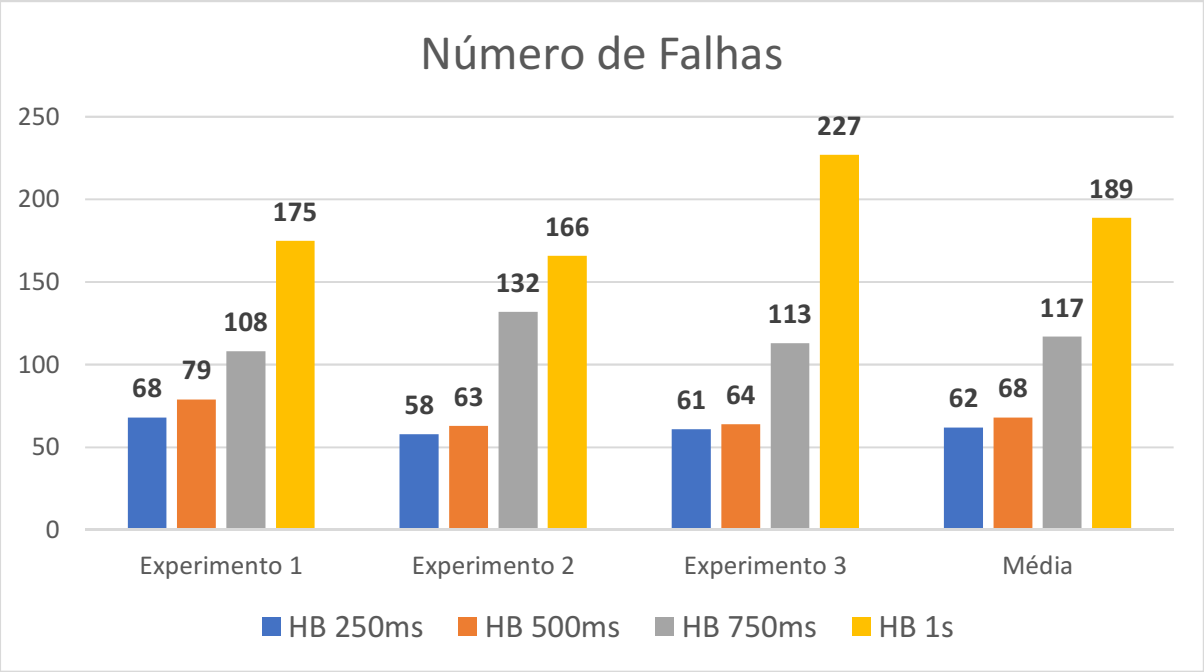
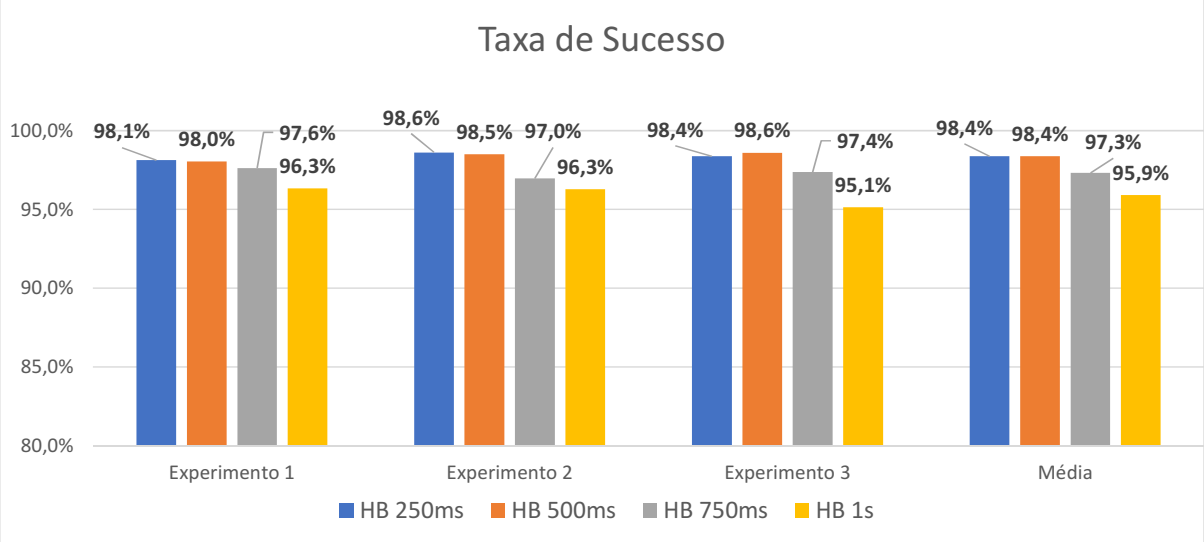


Figura 32 – Taxa de sucesso para três experimentos e média das taxas - Cenário 3.

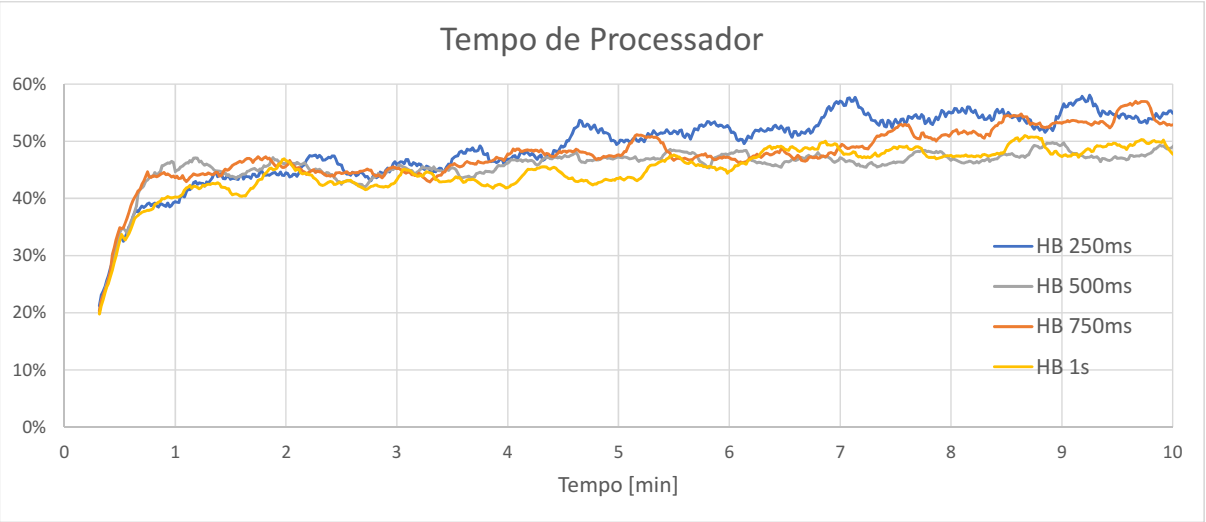


impactar na demanda computacional do SMA.

Para avaliar esse fenômeno de *trade-off* entre agilidade na detecção com a demanda computacional, a Figura 33 apresenta o monitoramento do uso de tempo de processador ao longo da execução de cada um dos testes. Nessa simulação, no entanto, não é possível constatar uma diferença considerável entre os intervalos curtos e os mais longos. As curvas similares de tempo de processador indicam que o intervalo de tempo para conferência do monitor de *heartbeat*, por se tratar basicamente da adição de um plano na base de planos do agente, é um parâmetro de impacto pequeno

no desempenho computacional do sistema.

Figura 33 – Tempo de processador ao longa das simulações - Cenário 3.



4.4.4 Cenário 4 - Simulação com variação no intervalo entre *checkpoints*

O cenário 4 explora o segundo parâmetro principal do modelo de TF, que é a frequência de criação de pontos de restauração. Esse parâmetro pode ter impacto no desempenho do sistema pois os pontos de restauração contém informações sobre o estado do agente e, caso eles sejam criados com uma frequência mais alta, a tendência é que eles contenham informações mais recentes sobre um agente. Esses dados mais atualizados podem auxiliar o modelo no momento em que é realizada uma restauração de um agente no sistema. Em contrapartida, o processo de *checkpoint* demanda um custo computacional representativo, pois envolve uma série de procedimentos de leitura e gravação de dados em arquivos. Para avaliar esse *trade-off* do modelo, varia-se o intervalo entre criação de *checkpoints* de 500 ms até 10 segundos, conforme apresenta os parâmetros do Quadro 7

Quadro 7 – Quadro de parâmetros para cenário 4.

Leilão	Número de participantes Número de rodadas Duração do leilão	2 leiloeiros 15 produtores 20 manufatureiros 15 consumidores 5 12min
TF	Intervalo para monitor de heartbeat Intervalo entre checkpoints	500ms 500 ms - 1s - 5s - 10s
Gerador de Faltas	Intervalo entre faltas Período com gerador de faltas ativo	1s 10min

Os desempenhos do modelo, avaliado pelas métricas do número de falhas, na Figura 34, e da taxa de sucesso, na Figura 35, indicam performances inferiores para os intervalos de *checkpoints* mais curtos e mais longos. Os intervalos de tempo intermediários apresentam desempenhos similares entre si e superiores em relação aos outros.

Figura 34 – Número de falhas identificadas durante a execução de três experimentos e média das falhas - Cenário 4.

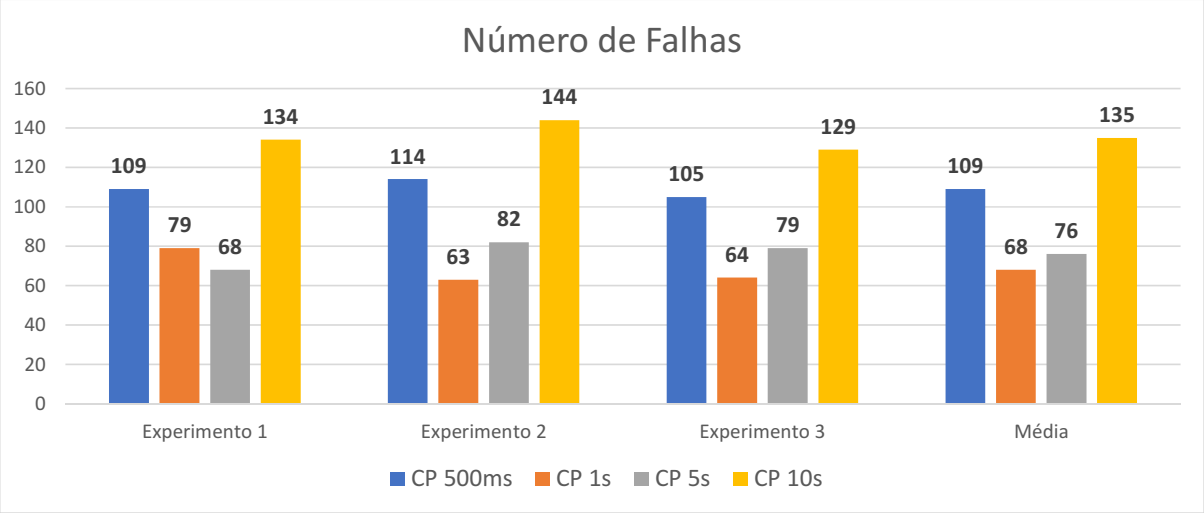
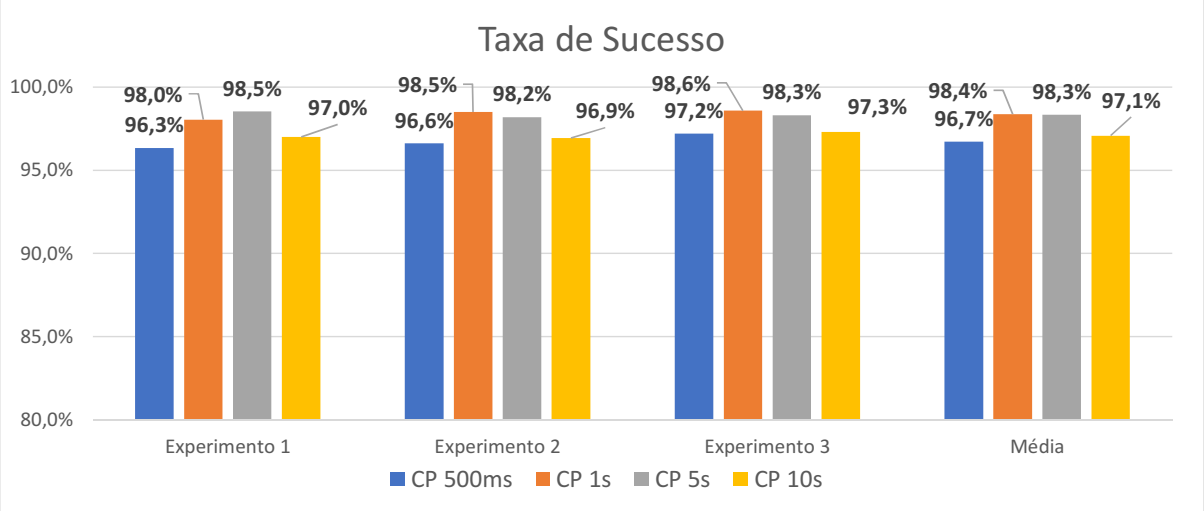


Figura 35 – Taxa de sucesso para três experimentos e média das taxas - Cenário 4.

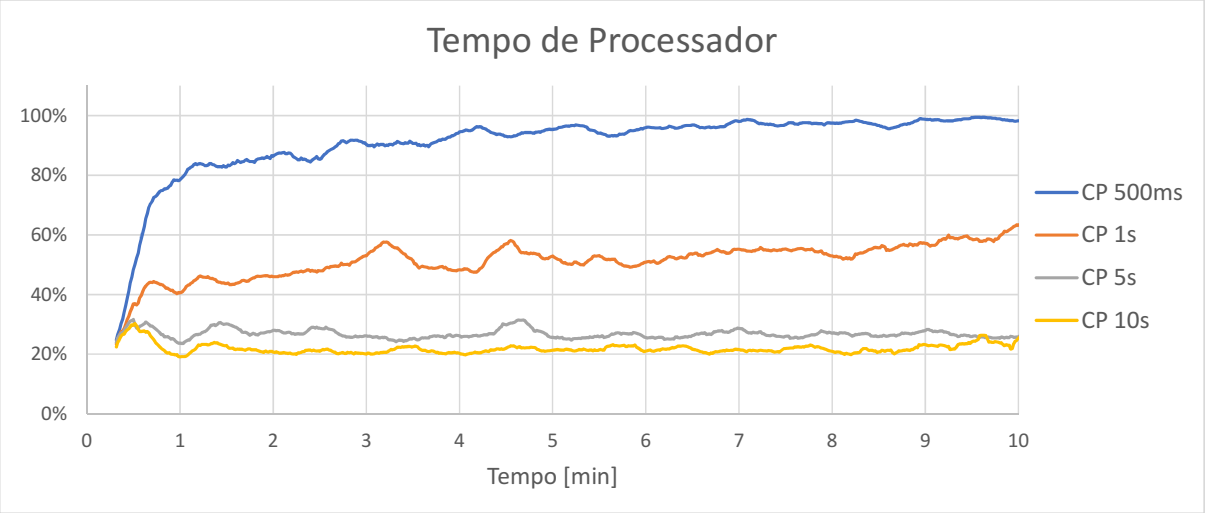


O elevado número de falhas para o intervalo de tempo de 10 segundos pode se dar pelo fato do ponto de restauração do agente conter informações que se tornaram obsoletas devido a dinâmica do restante do sistema. Pode-se imaginar, por exemplo, que um dado ponto de restauração `c1` contém informações sobre o agente em um dado instante em que esse está operando em um leilão `l1`. Após um certo tempo, esse leilão é finalizado e inicia-se um novo leilão `l2`. Na hipótese do agente sofrer uma falta

nesse instante e ser restaurado para `c1`, os estados do agente conteriam informações conflitando ao agente (e.g. o agente acredita que o leilão `l1` esta aberto), o que poderia conduzir a uma falha posterior.

As simulações com o intervalo de 500ms para criação de pontos de restauração, apesar de manterem os estados do agente mais atualizados, mostrou um desempenho inferior, se comparado aos intervalos de tempo intermediários. A justificativa para esse fato se da devido ao alto custo computacional demandado para executar todos os processos de gerenciamento de arquivos do *checkpoint*. A Figura 36 apresenta o percentual de uso de processador para cada um dos testes. Verifica-se que o cenário com alta frequência de criação de pontos de restauração auxiliou para que o tempo do processador fosse comprometido em até 100%, o que evidencia que os processos de *checkpoint* estariam comprometendo o restante dos processos do SMA. As simulações com frequência inferiores apresentaram valor de tempo de processador consideravelmente inferiores. Essa análise demonstra que o intervalo entre *checkpoints* é um parâmetro do modelo com alto impacto no desempenho computacional do sistema.

Figura 36 – Tempo de processador ao longa das simulações - Cenário 4.



4.4.5 Cenário 5 - Simulação prolongada com geração moderada de faltas

Por fim, o cenário 5 executa o SMA por períodos mais prolongados e diminui o tempo de intervalo entre ocorrências de faltas, na intenção de verificar o comportamento do sistema e do modelo de TF em situações onde as faltas ocorrem de maneira menos acentuada em comparação com os cenários anteriores. Os parâmetros de teste são apresentados no Quadro 8.

Nesse cenário um pouco mais conservador, mesmo com o SMA executando um número acima dos 10000 negociações, o número de falhas manteve-se considera-

Quadro 8 – Quadro de parâmetros para cenário 5.

Leilão	Número de participantes Número de rodadas Duração do leilão	2 leiloeiros 15 produtores 20 manufatureiros 15 consumidores 5 25min
TF	Intervalo para monitor de heartbeat Intervalo entre checkpoints	500ms 1s
Gerador de Faltas	Intervalo entre faltas Período com gerador de faltas ativo	15s 25min

velmente baixo, obtendo uma taxa de sucesso média de 99,98%. A justificativa para obtenção dos melhores resultados se deve a combinação de parâmetros para o modelo de TF que estabelecem uma dinâmica do modelo condizente com a dinâmica do SMA como um todo, permitindo que eventuais faltas possam ser detectadas e corrigidas em tempo hábil de restaurar os agentes ao sistema de forma que eles consigam desempenhar suas funções de maneira satisfatória. Além disso, os parâmetros escolhidos asseguram um *trade-off* saudável entre desempenho do modelo e custo computacional do sistema. A quantidade total de negociações, número de falhas e as taxas de sucesso obtidas nos experimentos do cenário 5 estão apresentadas na Tabela 1.

Tabela 1 – Desempenho para cenário 5.

	Experimento 1	Experimento 2	Experimento 3	Média
Total de negociações	10106	10141	9989	10078
Número de falhas	3	2	1	2
Taxa de sucesso	99,97%	99,98%	99,99%	99,98%

4.5 AVALIAÇÃO DOS RESULTADOS

Os cenários explorados nesse capítulo servem como base para uma série de análises do comportamento do modelo de TF quanto a sua capacidade de atuar na detecção e recuperação de agentes expostos a faltas de quebra, que são faltas que impedem o agente de executar suas funções internas.

O modelo mostrou ser capaz de realizar as funções de detecção de agentes em estado de erro e posterior restauração desses agentes no SMA para continuação dos serviços. Mesmo operando em um SMA dinâmico e em cenários com frequências elevadas de faltas, o modelo permaneceu oferecendo os mesmo serviços de TF. Todavia, fica claro que cenários mais extremos acabam reduzindo a performance do sistema e que o modelo não é capaz de garantir que o SMA tolere toda e qualquer falta. Quando se fala de sistemas bastante dinâmicos e imprevisíveis, é razoável esperar que uma

certa quantidade de faltas não sejam corrigidas pelo modelo, ocasionando falhas no sistema como consequência.

Os dois parâmetros principais do modelo (intervalo do monitor de *heartbeat* e intervalo entre criação de *checkpoints*) podem e devem ser customizados para garantir uma melhor desempenho do modelo de TF. Para escolher os valores mais adequados é indicado ter um certo entendimento sobre a dinâmica da aplicação como um todo.

Ao configurar intervalos mais curtos para o monitor de *heartbeat*, aumentam-se as chances de identificar e recuperar com sucesso um agente em estado de erro. Para esse parâmetro, indica-se utilizar valores similares ou inferiores a dinâmica do SMA.

O parâmetro referente ao processo de criação de pontos de restauração apresenta um impacto na qualidade da recuperação do sistema. A tendência observada nos experimentos é que, com frequências mais altas para criação de *checkpoints*, os pontos de restauração se mantenham mais atualizados quanto aos estados do agente, o que pode contribuir para evitar falhas durante as primeiras fases posteriores a restauração do agente no sistema. Essa melhora no desempenho do modelo com o aumento da frequência de *checkpoints*, no entanto, não deve ser tomada como regra absoluta, pois cada aplicação possui especifidades que podem revelar a necessidade de estratégias diferenciadas. Como exemplo, pode-se pensar em sistemas que exijam rotinas sequenciais que só podem ser executadas uma única vez e em uma ordem específica. Nessa situação, a criação de pontos de restauração periódicos no meio do procedimento pode não ser a alternativa mais apropriada e, no lugar dessa abordagem, seria mais adequado criar um ponto de restauração ligeiramente anterior ao início do processo sequencial, de maneira a garantir que agentes restaurados no sistema executem todo o procedimento sequência de um ponto inicial mais bem definido.

Além da especificidade da aplicação, outra questão pertinente as escolhas dos parâmetros do modelo é quanto a demanda computacional que o modelo pode trazer de impacto ao sistema. Como demonstrou-se nos cenários de teste, o intervalo de monitor de *heartbeat* apresenta um impacto pequeno para o sistema. Por sua vez, a criação de pontos de restauração apresenta um impacto significativo, que deve ser levado em conta para não comprometer o bom funcionamento das outras funções do SMA.

Por fim, conforme demonstrado no último cenário deste capítulo, a escolha de parâmetros adequados, que consideram os aspectos da dinâmica do SMA, a susceptibilidade do sistema a ocorrência de faltas e também as questões de limitações computacionais, contribuem para a obtenção de rendimentos satisfatórios para o modelo de TF implementado.

5 CONCLUSÕES E SEQUÊNCIA DO TRABALHO

O presente trabalho elaborou um modelo de TF aplicado em SMA adaptado a programação multidimensional. O modelo utilizou as abstrações da dimensão dos agentes e do ambiente para fornecer serviços de TF ao SMA utilizando analogias de alto nível. A concepção do monitor de saúde e dos agentes de saúde, juntamente com o conceito de tratar as faltas no nível do SMA (abordagem cidadã) criou uma separação entre domínio da solução, onde os agentes operam suas rotinas comuns, e domínio do tratamento das faltas. Essa distinção que o modelo faz torna o serviço de TF transparente para o desenvolvedor do SMA.

O modelo elaborado apresentou três funcionalidades principais: o fornecimento de serviço de TF, a relação de monitoramento por interesse e o monitoramento contínuo do agente. Essas funcionalidades possuem características distintas que podem ser utilizadas para favorecer a dependabilidade do SMA.

O serviço de TF é a funcionalidade central do modelo, os mecanismos de detecção de erros por monitor de *heartbeat* e recuperação de agentes em estado de erro por *checkpoint* e retorno mostraram-se promissores na medida que foram testadas em cenários dinâmicos e com níveis elevados de ocorrência de faltas.

O processo de *checkpoint*, da forma como foi implementado, apresentou alguns pontos de discussão em torno do intervalo entre criação dos pontos de restauração. As simulações realizadas indicaram que a alta frequência de *checkpoints* não necessariamente leva o sistema a uma diminuição no número de falhas ao longo da operação e pode inclusive acabar inserindo gargalos computacionais no sistema. Esse tipo de análise levanta a necessidade de aprofundar a pesquisa em alternativas para criação dos pontos de restauração, podendo ser implementado a criação de pontos de restauração acionada por eventos, onde os pontos seriam criados em momentos específicos da execução do SMA (e.g. inicialização do sistema, pontos seguros conhecidos), ou então por acionado por algoritmo, onde os pontos são criados em pontos trechos do código particulares do agente (e.g. no início de um ciclo de raciocínio do agente ou antes de executar um plano atômico do agente).

As funcionalidade do monitoramento por interesse e o monitoramento contínuo foram introduzidas no modelo como ferramentas que podem ser usadas pelo programador para aumentar a confiabilidade do SMA. A incorporação de ambas as funcionalidades utilizou as abstrações que relacionam as dimensões dos agente e do ambiente. No monitoramento por interesse, adicionou-se a possibilidade do programador inserir, no código de agentes comuns do sistema, planos que são ativados conforme eventos gatilhos gerados em outros agentes do sistema. O monitoramento contínuo, por sua vez, permite ao programador personalizar o agente de saúde do sistema, inserindo planos para intervir no sistema conforme apareçam sintomas em

agentes monitorados. Essas duas ferramentas fornecem uma maior versatilidade ao modelo de TF, permitindo que possam ser criadas estratégias defensivas para tentar conter uma disseminação dos efeitos de faltas no sistema e prevenir inclusive a ocorrência de uma falta. A combinação dessas duas funcionalidades, juntamente com os serviços mais tradicionais de TF que o modelo oferece, apresenta um acréscimo considerável a dependabilidade do sistema.

Mais especificamente a respeito do monitoramento contínuo do agente, foram apresentados os novos conceitos do nível mental e nível de arquitetura de um agente. Essa classificação permitiu o levantamento de informações sobre o estado de cada agente do sistema, retratando em tempo real como está a saúde mental do agente e o quanto ele está interagindo com o ambiente externo. Essas informações foram sintetizadas na equação do indicador de nível de estresse do agente. Nessa primeira versão do indicador já é possível entender a ferramenta como um ponto relevante para monitoramento e análise da evolução de um agente ao longo da execução do SMA. A partir dessas análises, é possível perceber, por exemplo, que existem períodos específicos em que o agente apresenta elevações ou decréscimos no nível de estresse. Essas informações podem servir como ferramenta de diagnóstico de gargalos e ativador de eventos para ações preditivas no agente monitorado. Acredita-se que o indicador de estresse é um dos pontos do monitoramento contínuo com potencial de pesquisa futura, onde pode-se analisar possíveis melhorias na equação e obtenção do parâmetro, buscando maneiras de quantificar o indicador e tornar ele um parâmetro mais genérico que pode ser utilizado em comparações.

A partir do desenvolvimento deste trabalho, foram concebidas algumas propostas para trabalho futuros. Alguns pontos já foram discutidos, como um aprofundamento no estudo do indicador de estresse e das estratégias de *checkpoint*. Além dessas questões, um tema para análise futura pode ser uma maneira de abordar a TF na dimensão do ambiente. O modelo hoje utiliza a dimensão do ambiente para tratar faltas dos agentes, mas uma eventual falta ocorrida em um artefato, por exemplo, não possui tratamento dedicado no modelo. Para adicionar TF em um artefato é necessário utilizar mecanismos de programação defensiva próprios como, por exemplo, o tratamento de exceções do tipo *try-catch*, do Java.

Outro tema para análise futura pode ser um estudo de como seria possível adicionar a dimensão da organização ao modelo e quais seriam os benefícios de se fazer isso. Uma proposta para utilizar a organização poderia ser, por exemplo, pensar nela participando na orquestração de planos de contingência, atuando no gerenciamento de recursos alocados e disponíveis para realização de serviços (e.g. a organização retirando ou dividindo tarefas que foram alocadas para agentes demasiadamente “estressados” do sistema). Outro exemplo para o uso da organização pode ser na geração de pontos de restauração comuns do sistema, onde todos os agentes criam seus *check-*

point de acordo com que a organização entende por um “ponto seguro”, auxiliando assim o SMA nos processos posteriores a recuperação de uma falta.

Os objetivos propostos para o trabalho foram atingidos em sua totalidade, o modelo de TF elaborado mostrou ser capaz de fornecer uma estrutura que contribui para o aumento da confiabilidade do SMA. Além disso, o modelo pôde ser implementado na plataforma JaCaMo utilizando as dimensões dos agentes e do ambiente como estrutura base de funcionamento e as funcionalidades do serviço de TF foram avaliadas em diversos cenários de testes. O trabalho desenvolvido rendeu uma publicação no evento nacional WESAAC (Workshop Escola de Sistemas de Agentes, seus Ambientes e Aplicações), ocorrido no mês de maio de 2019, sendo selecionado entre os melhores artigos do evento.

REFERÊNCIAS

AVIŽIENIS, A. Design of fault-tolerant computers. *In*: PROCEEDINGS of the November 14-16, 1967, fall joint computer conference. [*S.l.*: *s.n.*], 1967. P. 733–743.

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE transactions on dependable and secure computing**, IEEE, v. 1, n. 1, p. 11–33, 2004.

BOISSIER, O.; BORDINI, R. H.; HÜBNER, J. F.; RICCI, A. Dimensions in programming multi-agent systems. **The Knowledge Engineering Review**, Cambridge University Press, v. 34, 2019.

BOISSIER, O.; BORDINI, R. H.; HÜBNER, J. F.; RICCI, A.; SANTI, A. Multi-agent oriented programming with JaCaMo. **Science of Computer Programming**, Elsevier, v. 78, n. 6, p. 747–761, 2013.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming multi-agent systems in AgentSpeak using Jason**. [*S.l.*]: John Wiley & Sons, 2007. v. 8.

DE FLORIO, V. **Application-layer fault-tolerance protocols**. [*S.l.*]: IGI Global, 2009.

DELLAROCAS, C.; KLEIN, M. An experimental evaluation of domain-independent fault handling services in open multi-agent systems. *In*: IEEE. PROCEEDINGS Fourth International Conference on MultiAgent Systems. [*S.l.*: *s.n.*], 2000. P. 95–102.

DÍAZ, Á. F. **eJason : a Framework for Distributed and Fault-tolerant Multi-Agent Systems**. 2018. F. 197. Tese (Doutorado).

FEDORUK, A.; DETERS, R. Improving fault-tolerance by replicating agents. *In*: ACM. PROCEEDINGS of the first international joint conference on Autonomous agents and multiagent systems: part 2. [*S.l.*: *s.n.*], 2002. P. 737–744.

HÄGG, S. A sentinel approach to fault handling in multi-agent systems. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 1286, p. 181–195, 1997.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. A model for the structural, functional, and deontic specification of organizations in multiagent systems. *In*: SPRINGER. BRAZILIAN Symposium on Artificial Intelligence. [*S.l.*: *s.n.*], 2002. P. 118–128.

ISONG, B. E.; BEKELE, E. A Systematic Review of Fault Tolerance in Mobile Agents. **American Journal of Software Engineering and Applications**, v. 2, n. 5, p. 111–124, 2013.

JENNINGS, N. R.; SYCARA, K.; WOOLDRIDGE, M. A roadmap of agent research and development. **Autonomous agents and multi-agent systems**, Kluwer Academic Publishers, v. 1, n. 1, p. 7–38, 1998.

KLEIN, M.; RODRIGUEZ-AGUILAR, J.-A.; DELLAROCAS, C. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 7, n. 1-2, p. 179–189, 2003.

LAPRIE, J. C. Dependability: Basic Concepts and Terminology. Springer, Vienna, p. 3–245, 1992.

LEE, P. A.; ANDERSON, T. Fault tolerance. *In*: FAULT Tolerance. [*S.l.*]: Springer, 1990. P. 51–77.

MARIN, O.; BERTIER, M.; SENS, P. DARX-a framework for the fault-tolerant support of agent software. *In*: IEEE. 14TH International Symposium on Software Reliability Engineering, 2003. ISSRE 2003. [*S.l.*: *s.n.*], 2003. P. 406–416.

OMICINI, A.; RICCI, A.; VIROLI, M. Artifacts in the A&A meta-model for multi-agent systems. **Autonomous agents and multi-agent systems**, Springer, v. 17, n. 3, p. 432–456, 2008.

OUYANG, F.-R.; WANG, X.-J.; YANG, H. Agent bidding strategy and simulation in double auctions. *In*: IEEE. PROCEEDINGS of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826). [*S.l.*: *s.n.*], 2004. v. 3, p. 1836–1840.

POTIRON, K.; SEGHROUCHNI, A. E. F.; TAILLIBERT, P. **From fault classification to fault tolerance for multi-agent systems**. [*S.l.*]: Springer, 2013.

PULLUM, L. L. **Software fault tolerance techniques and implementation**. [*S.l.*]: Artech House, 2001.

RICCI, A.; PIUNTI, M.; VIROLI, M. Environment programming in multi-agent systems: an artifact-based perspective. **Autonomous Agents and Multi-Agent Systems**, Springer, v. 23, n. 2, p. 158–192, 2011.

RICCI, A.; VIROLI, M.; OMICINI, A. CArtAgO: An infrastructure for engineering computational environments in MAS. **Weyns et al.[31]. To appear**, 2006.

SHEHORY, O.; SYCARA, K.; CHALASANI, P.; JHA, S. Agent cloning: an approach to agent mobility and resource allocation. **IEEE communications Magazine**, IEEE, v. 36, n. 7, p. 58–67, 1998.

SIERRA, K.; BATES, B. **Head first java**. [*S.l.*]: "O'Reilly Media, Inc.", 2003.

STANKOVIĆ, R.; ŠTULA, M.; MARAS, J. Evaluating fault tolerance approaches in multi-agent systems. **Autonomous agents and multi-agent systems**, Springer, v. 31, n. 1, p. 151–177, 2017.

WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. **Relatório técnico, Instituto de Informática UFRGS**, 2002.

WOOLDRIDGE, M. Intelligent agents. **Multiagent systems**, MIT Press Cambridge, v. 6, 1999.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. **The knowledge engineering review**, Cambridge University Press, v. 10, n. 2, p. 115–152, 1995.