Whistle: Um Detector de Defeitos para Sistema Multiagentes Utilizando a *Framework* JADE

Adailton de J. Cerqueira Jr.[1], **Sérgio Gorender**[1] , **Diego Frías**[2]

[1]Laboratório de Sistemas Distribuídos (LaSID)
Universidade Federal da Bahia (UFBA) - Salvador/BA

[2]Universidade do Estado da Bahia (UNEB) - Salvador/BA

adailton.junior@gmail.com, gorender@ufba.br, diegofriass@gmail.com

Abstract. *In multi-agent systems, as in distributed systems, the occurrence of a failure can cause the entire system to display errors in their results. To tolerates fault, a multiagent system needs fault tolerant mechanisms. Mechanism such as replication and failure detector, developed to distributed systems, can be used in multiagent systems. On the other hand, multiagent systems are frequentely developed using agent programming frameworks, and artificial inteligent algorithms. To be used in multiagent systems, fault tolerance mechanisms must be adapted to new requisitions. This paper presents a failure detector service developed for the JADE framework.*

Resumo. *Em sistemas multiagentes, de forma similar a um sistema distribuído, a ocorrência de uma falha pode fazer com que todo o sistema venha a apresentar erros em seus resultados. Para que um sistema multiagente possa executar de forma correta tolerando falhas, torna-se necessário o uso de mecanismos para tolerância a falhas. Podem ser utilizados mecanismos estudados para sistemas distribuídos. Por outro lado, como os sistemas multiagentes são frequentemente desenvolvidos em ambientes próprios para a programação de agentes, e são criados usando algoritmos de inteligência artificial, mecanismos para prover tolerância a falhas aos sistemas precisam ser adaptados a estas novas características. Este artigo apresenta um detector de defeitos para sistemas multiagentes desenvolvido para o framework JADE.*

1. Introdução

Em sistemas distribuídos, a ocorrência de uma falha em um dos processos, caracterizada pela interrupção na execução do processo, pode fazer com que todo o sistema venha a apresentar erros em seus resultados, caracterizando um defeito neste sistema. O mesmo pode ocorrer em sistema multiagentes (SMAs), nos quais os processos que compõem o sistema são agentes inteligentes autônomos, desenvolvidos com técnicas de inteligência artificial. Nos SMAs, de forma similar aos sistemas distribuídos, os agentes se comunicam via troca de mensagens, e coordenam suas ações para alcançar seus objetivos.

Mecanismos para tolerância a falhas têm sido propostos para que sistemas distribuídos possam continuar a executar de forma correta, mesmo que falhas venham a ocorrer. Considerando sistemas multiagentes como sendo sistemas distribuídos, de uma maneira geral estes mesmos mecanismos podem ser utilizados para prover tolerância a

falhas neste sistemas. Um mecanismo bastante utilizado para tolerância a falhas em sistemas distribuídos são os detectores de defeitos.

Em um trabalho bastante referenciado [Chandra and Toueg 1996], Chandra e Toueg apresentaram os detectores de defeitos não confiáveis para sistemas assíncronos. Neste artigo é apresentado o conceito de detectores de defeitos não confiáveis, e apresentadas 8 classes de detectores de defeitos para sistemas distribuídos.

Embora detectores de defeitos e outros mecanismos desenvolvidos para sistemas distribuídos tolerantes a falhas possam também ser aplicados a agentes, os agentes apresentam características específicas em seu funcionamento. Agentes inteligentes são em geral auto-interessados e possuem um comportamento autônomo. Os agentes decidem executar ações para alcançar seus objetivos, e podem se adaptar de forma autonômica a alterações no ambiente de execução, de forma a executar com maior eficiência para a obtenção destes objetivos. Um agente pode, por exemplo, decidir por não responder à solicitações, o que em sistemas distribuídos caracterizaria uma falha por omissão.

Em [Potiron et al. 2008] o trabalho apresentado por Laprie e outros em [Laprie et al. 2004], no qual é apresentada uma taxonomia de falhas para SD, é estendido, para também considerar falhas que possam ocorrer de forma específica em SMAs. Assim, falhas geradas pelo comportamento autônomo dos agentes passam também a ser consideradas.

Neste artigo apresentamos um detector de defeitos para SMAs, desenvolvido para o framework JADE. JADE é um framework para a programação de agentes baseado na linguagem JAVA, e que disponibiliza para o desenvolver uma série de protocolos e mecanismos para a implementação de agentes. O detector de defeitos apresentado é não confiável, e executa de forma centralizada, provendo a possibilidade de detectar a falha em agentes, e propiciar informação sobre estes, no JADE.

Este artigo se organiza da seguinte forma: na Seção 2 apresentamos alguns trabalhos em detecção de defeitos para SMAs; na seção 3 apresentamos o detector de defeitos proposto neste trabalho; e na seção 4 apresentamos aspectos da implementação e alguns resultados. Na seção 5 apresentamos as conclusões do trabalho.

2. Detecção de Defeitos em Sistemas Multiagentes

Embora, de uma maneira geral, a pesquisa em SMAs tenha focado nos aspectos da inteligência artificial, em especial em mecanismos de coordenação e decisão entre agentes, alguns artigos têm apresentado propostas para prover mecanismos de tolerância a falhas para estes sistemas. Alguns destes trabalhos apresentaram mecanismos para detecção de defeitos, podendo ser fortemente baseados em algoritmos para detecção de defeitos para sistemas distribuídos, ou também considerando aspectos semânticos da construção de um sistema composto por agentes inteligentes autônomos [Gao et al. 2015].

No trabalho de [Haegg 1996], é apresentada uma estrutura de controle de agentes que monitora a comunicação utilizando um agente sentinela para monitorar funções e estados específicos do sistema. Esse agente tem a função de excluir agentes defeituosos, alterar parâmetros para os agentes e gerar relatórios para os operadores humanos. Para isso o sentinela utiliza a comunicação dos agentes para verificar a interação entre estes, construir modelos e detectar a falha de um agente.

Já em [Kaminka et al. 2002] é apresentado um detector baseado nos modelos mentais dos agentes. Este detector verifica inconsistências nas crenças dos mesmos. No entanto, a adaptação só é estrutural, onde os modelos de relação podem mudar, mas o conteúdo dos planos são estáticos. Com isso, essa abordagem acaba não sendo aconselhável para SMA abertos e adaptáveis.

Em [Guessoum et al. 2005] é apresentado um detector de defeitos distribuído que é integrado ao *framework* DarX. Esse mecanismo é composto por dois tipos de agentes: o *agent-monitors*, que é responsável por monitorar os agentes do ambiente e analisar as informações recebidas; e o *host-monitors*, que é responsável por coletar as informações globais do ambiente, como número total de mensagens no sistema e a troca de mensagens entre os agentes e enviar para os *agent-monitors* analisarem. Cada *agent-monitor* é associado a um agente do ambiente e se comunica com um *host-monitor* que está associado ao ambiente monitorado.

3. O Detector Whistle

O Whistle é um detector de defeitos centralizado desenvolvido para o *framework* JADE (**J**ava **A**gent **DE**velopment framework) [Bellifemine et al. 2007]. Esse *framework* foi desenvolvido em Java para facilitar o desenvolvimento de SMAs através de um *midleware* que cumpre as especificações da FIPA (*Foundation for Intelligent, Physical Agents*) [fip]. A programação no JADE está inserida no paradigma programação de agentes. No JADE os agentes são executados dentro de repositórios, que são chamados de *container* e que provêm todo o suporte para a execução do agente, abstraindo para o mesmo a complexidade do ambiente como, por exemplo, hardware e sistema operacional.

Assumimos que para este trabalho os agentes se comunicam por troca de mensagens, e que serão executados em um ambiente de comunicação assíncrono. Também assumimos que os agentes podem falhar por *crash*, quando param de funcionar, ou por omissão, quando param de responder a solicitações, apesar de continuarem em funcionamento [Laprie et al. 2004]. O detector de defeitos Whistle foi desenvolvido para detectar ambos os tipos de falhas.

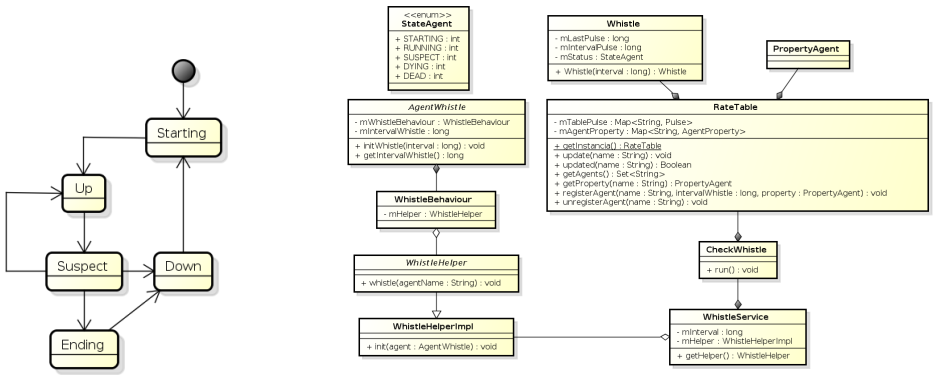
O detector proposto funciona segundo o modelo *push*, no qual cada agente envia para o detector mensagens indicando o seu correto funcionamento, a intervalos de tempo previamente definidos. O intervalo de tempo em que a mensagem deve ser enviada é determinado pelo tempo das tarefas realizadas pelo agente, ou seja, deve ser analisado o maior tempo utilizado por um agente para realizar uma tarefa. A partir deste tempo se determina o valor η, sendo esse maior que o tempo total para realizar a tarefa pelo agente. A partir do envio ou não desta mensagem, o Whistle define um estado para o agente. Os possíveis estados do agente são: **Starting**, estado de inicialização do agente; **Up**, estado de execução do agente; **Suspect**, estado onde o agente é suspeito de falha; **Ending**, estado em que o agente está sendo finalizado; e **Down**, estado onde o agente foi finalizado.

Com base no intervalo de tempo η e no estado atual do agente, o mecanismo inicia sua análise. Quando o agente fica sem enviar a mensagem por 2η o mecanismo modifica o estado do agente para *suspect* e verifica se o agente encontra-se no *container*. Neste momento existem duas possibilidades:

1ª Possibilidade: O agente não encontra-se no *container*, então o estado é modificado para *down*;

2ª Possibilidade: O agente ainda encontra-se no container, então o mecanismo considera-o em *loop* infinito, finaliza-o, modifica o estado para Ending até que ele seja finalizado.

Na Figura 1(a) são apresentadas as possíveis transições entre os estados do agente dentro do Whistle. Os estados também são utilizados pelo Whistle para informar aos demais agentes como o agente está.



(a) Transição entre os estados

(b) Diagrama de Classe do Whistle

Figure 1. Maquina de estado e diagrama de classe do Wistle

4. Implementação e Testes do Whistle

O Whistle foi desenvolvido na linguagem Java como um serviço no JADE. Na Figura 1(b) é apresentado o diagrama de classe, onde a classe **RateTable** é responsável por armazenar os estados de cada agente e deve ser atualizada por cada um no intervalo de tempo η. A classe **WhistleService** mantém o mecanismo executando e acessível aos agentes. Essa classe implementa o método **getHelper()** que devolve uma interface **WhistleHelper** que é acessível pelo agente e o permite enviar as mensagens de atualização. Além disso, a **WhistleService** executa a *thread* **CheckWhistle**, que verifica se os agentes estão enviando ou não a mensagem de atualização.

Por fim, temos a classe **AgentWhistle** que é uma classe abstrata que deve ser estendida por qualquer agente que pretenda implementar o mecanismo. O **AgentWhistle** possui um comportamento **WhistleBehaviour** que é responsável pelo envio da mensagem. Ao se iniciar o **AgentWhistle** deve-se indicar qual seu intervalo de atualização (η) para que ele possa se cadastrar no **WhistleService**. Vale ressaltar que esse intervalo de atualização deve ser maior que o tempo máximo do método que possui o maior tempo de execução do agente. Isso se deve pelo fato de que se o agente ficar muito tempo executando um método sem enviar uma mensagem o mecanismo irá finaliza-lo, pois acreditará que ele está em *loop* infinito.

Para a utilização do detector de defeitos são necessários realizar dois passos. O primeiro passo é iniciar o **WhistleService** indicando o parâmetro *whistle_interval* que indica o intervalo, em milissegundos, de checagem do mecanismo. Esse parâmetro é opcional e possui o valor padrão 1000 milissegundos. O segundo passo é indicar quais agentes deverão ser monitorados e para isto é só o agente estender a classe **AgentWhistle**

Número de Agentes	Acertos	Falsos Positivos	Falsos Negativos
10	100%	0%	0%
100	100%	8%	0%
1000	100%	32%	0%

Table 1. Resultados dos testes em relação ao número de agentes.

Intervalos de tempo (ms)	Tempo médio de detecção (ms)
10	9.2
100	96.5

Table 2. Tempo médio de resposta do detector de defeitos.

e executar o método **initWhistle(long interval)** indicando o período de atualização η em milissegundos.

Na fase de testes do mecanismo foram construídos dois cenários: um cenário onde um número controlado de agentes falhavam e outro sem nenhuma falha. O objetivo destes testes foram identificar a taxa de acerto do Whistle, o percentual de falsas suspeitas (quando o mecanismo identifica uma falha erroneamente) e o percentual de falsas detecções (quando o mecanismo não identifica uma falha). Para cada cenário foram realizados testes com 10, 100 e 1000 agentes; com intervalos de tempo de detecção de 10 milissegundos e 100 milissegundos. Cada teste teve um total de 100 iterações. Na Tabela 1 são apresentados os resultados dos testes realizados em relação à ocorrência de falsos positivos, falsos negativos e número de acertos.

Nos testes realizados, os índices de falsos positivos ficaram em aproximadamente 40%, e ocorreram, em sua maioria, nos cenários com 1000 agentes. Cerca de 80% das ocorrências dos falsos positivos foram nos ambientes com 1000 agentes e o restante no ambiente com 100 agentes. Isso ocorre devido ao fato de que, em ambientes com muitos agentes e com o intervalo de detecção curto, o mecanismo não consegue analisar todos e detecta alguns como falhos.

O tempo médio de detecção da falhas é relativo ao intervalo de tempo η e segue a regra de $T_m = \eta - \delta$, onde δ é o momento em que ocorreu a falha relativa ao último envio de sinal do agente. Ou seja, o defeito será detectado na próxima verificação do mecanismo. Como o intervalo de atualização η está relacionado aos métodos que o agente possui, deve-se implementar métodos que não bloqueiem o envio da mensagem de atualização. Na tabela 2 é apresentado o tempo médio dos intervalos de tempos utilizados durante os testes.

5. Conclusão

Apresentamos neste artigo um detector de defeitos para SMAs, implementado no framework JADE. Utilizando o JADE para a construção de agentes, segundo o paradigma programação de agentes, é agora possível dispor de um detector de defeitos que possa informar aos agentes a ocorrência de falhas nos demais agentes do sistema. Este detector foi desenvolvido para servir como bloco de construção para um dispositivo para tolerância a falhas em SMAs que, a partir da detecção de agentes com falha, possa reiniciar estes agentes, recuperando o seu estado da falha. Este detector também será utilizado por um mecanismo de replicação para SMAs, o qual está em desenvolvimento.

O detector de defeitos é não confiável quando executado em ambiente assíncrono de comunicação, e detecta falhas por *crash* e por omissão. A implementação do agente passará por um processo de otimização no sentido de melhorar a sua qualidade de serviço. Como trabalho futuro pretende-se, baseado no conceito de QoS de detectores de defeitos [Chen et al. 2002], tornar o intervalo de tempo de detecção adaptável a aspectos do ambiente e requisitos do sistema.

O Whistle está sendo utilizado na plataforma multiagente *Service Quality Multi-Agent System* (SQ-MAS). O SQ-MAS é um sistema de monitoramento e avaliação da qualidade do serviço em redes de distribuição de energia elétrica [Sanchez Dominguez et al. 2015]. Em todos os testes realizados o detector funcionou de forma satisfatória, detectando e corrigindo as falhas e sem apresentação de falsos positivos.

References

FIPA foundation for intelligent, physical agents. `http://www.fipa.org/`. Accessed: 2014-01-25.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*, volume 7. Wiley. com.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267.

Chen, W., Toueg, S., and Aguilera, M. K. (2002). On the quality of service of failure detectors. *Computers, IEEE Transactions on*, 51(5):561–580.

Gao, Z., Cecati, C., and Ding, S. X. (2015). A survey of fault diagnosis and fault-tolerant techniques part i: Fault diagnosis with model-based and signal-based approaches. *Industrial Electronics, IEEE Transactions on*, 62(6):3757–3767.

Guessoum, Z., Faci, N., and Briot, J.-P. (2005). Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–6.

Haegg, S. (1996). *A sentinel approach to fault handling in multi-agent systems*. Springer.

Kaminka, G. A., Pynadath, D. V., and Tambe, M. (2002). Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research*.

Laprie, J.-C., Avizienis, A., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

Potiron, K., Taillibert, P., and Seghrouchni, A. E. F. (2008). A step towards fault tolerance for multi-agent systems. In *Languages, Methodologies and Development Tools for Multi-Agent Systems*, pages 156–172. Springer.

Sanchez Dominguez, J., Cerqueira Junior, A. J., Sanchez Dominguez, D., Frias, D., and Marrero Iglesias, S. (2015). Using a multi-agent system for monitoring indicators of quality of service in power distribution networks. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 13(4):1048–1054.