Transparência em Sistemas Multiagentes

Caique Peixoto, Maurício Serrano e Milene Serrano

Faculdade do Gama – Universidade de Brasília (UNB)
Caixa Postal 8114 – 72.405-610 – Brasilia – DF – Brazil
{caiquepeixoto1,serr.mau,mileneserrano}@gmail.com
http://fga.unb.br/

Resumo A proposta desse artigo consiste em aplicar o conceito de transparência na interação entre agentes de software, os quais simulam, em contextos do cotidiano, a socialização entre indivíduos. O intuito é proporcionar maior auditabilidade, confiabilidade, clareza, rastreabilidade, dentre outros critérios de qualidade, principalmente, à comunicação entre os agentes. Vários estudos de caso, orientados ao paradigma de sistemas multiagentes, foram desenvolvidos para identificar situações recorrentes nas interações entre os agentes. Dada a intrínseca autonomia, mobilidade, flexibilidade, dentre outros aspectos autonômicos dos agentes de software, percebeu-se, a necessidade de manter, por exemplo, o rastro (i.e log) das decisões, negociações, colaborações e demais ações desempenhadas pelos agentes. No intuito de evoluir nesse sentido, foi eleito um estudo de caso, o qual foi refinado com o desenvolvimento de um agente estratégico na comunidade multiagentes. Diante dos resultados obtidos com a pesquisa experimental, os quais são relatados nesse artigo, observa-se maior rastreabilidade, auditabilidade, clareza, e transparência na interação entre os agentes. Além disso, a proposta é dirigida à reutilização de software e à noção de retroalimentação.

Keywords: Transparência, rastreabilidade, auditabilidade, multiagentes

1 Introdução

Transparência é um conceito moderno, especialmente investigado em contextos político-sociais. Considerando a definição de Holzner&Holzner [1], o termo transparência pode ser definido como: *"valor social do acesso aberto, público e/ou individual à informação mantida e disponibilizada por centros de autoridade"*.A transparência pode ser entendida como um critério de qualidade crítico para sociedades democráticas modernas. Dado o emergente cenário, é crescente a demanda por produtos de software, os quais apoiam vários anseios da sociedade (ex. auditabilidade e acesso à informação). Portanto, a transparência de software está se tornando um critério de qualidade que demanda mais atenção por parte dos desenvolvedores.

Leite e Cappelli [2] fazem um extenso levantamento do estado da arte no que tange à transparência em contextos político-sociais, e apresentam uma abordagem recente para transparência no contexto da Engenharia de Software. Os

autores lidam com a transparência de software como um requisito não-funcional, ou meta-flexível, ou critério de qualidade.

No Laboratório de Engenharia de Software da Universidade de Brasília (UnB / FGA), a transparência de software tem sido investigada no intuito de agregar valores sociais relacionados, principalmente, com acessibilidade, usabilidade, informatividade, entendimento e auditabilidade das informações providas pelos produtos de software que permeiam nossas vidas. Nesse sentido, investiga-se o conceito de transparência de software na interação de agentes de software, os quais compõem sistemas multiagentes que simulam a colaboração, a negociação e demais atividades de socialização entre indivíduos.

Um agente é um software que automaticamente age/reage para executar ações [3]. Agentes de software têm propriedades particulares que os diferenciam dos componentes e artefatos tradicionais de software, tais como: autonomia, flexibilidade, comunicação, adaptabilidade e mobilidade. Um sistema multiagentes representa uma coleção de agentes de software (i.e. entidades autônomas com propósitos particulares). Esses agentes cooperam uns com os outros de acordo com a organização e no intuito de solucionar/desempenhar as ações demandadas. Vale ressaltar que os sistemas multiagentes [4] têm sido utilizados em diferentes áreas da Computação (ex. Inteligência Artificial e Engenharia de Software). Além disso, sua aplicabilidade se estende a diferentes domínios cognitivos (ex. jogos, telemedicina, comércio e educação).

Nos diferentes domínios, cabe aos agentes a responsabilidade de interagir e executar ações para atingir as metas delegadas aos mesmos. As interações podem ocorrer entre os agentes e o ambiente bem como entre os próprios agentes, seguindo protocolos de comunicação específicos [5]. A organização determina as ações que serão executadas pelos agentes. O ambiente provê recursos que podem ser utilizados pelos agentes na execução dessas ações. Esses recursos também impactam na forma como as ações serão desempenhadas. Cada elemento (agente, interação, organização ou recurso) contém uma esfera de influência dentro do ambiente em análise. A comunicação permite lidar com as informações trocadas entre os agentes, focando na percepção dos mesmos e no processo interativo.

Percebe-se nas colocações supracitadas que as interações entre os agentes são intrínsecas bem como essenciais para a maior colaboração, negociação e demais relações sociais entre eles. Diante do exposto, e visando maior clareza, auditabilidade, confiabilidade, controlabilidade, dentre outros critérios de qualidade [2], propomos a aplicação do conceito de transparência de software em sistemas multigentes dirigidos pela socialização dos agentes. Durante as aulas da disciplina de Paradigmas de Programação, ministradas na UnB/FGA, vários estudos de caso, orientados ao paradigma de Sistemas Multiagentes, foram desenvolvidos com o objetivo de explorar as capacidades do paradigma. Foi identificada, nesses sistemas, a necessidade de manter, por exemplo, o rastro (i.e. log) das decisões, negociações, colaborações e demais ações desempenhadas pelos agentes. Dada essa necessidade, foi desenvolvido um agente cuja principal atuação era observar os demais agentes em suas ações cooperativas, registrando o ocorrido, ou seja, mantendo o rastro quanto as ações dos agentes, permitindo que essas fossem au-

ditáveis, e impactando positivamente em transparência de software - uma forma de retroalimentação [6]. Entretanto, nessa segunda evolução, percebeu-se a replicação de código e a falta de legibilidade, dada a necessidade de manter trocas de mensagens constantes entre a comunidade de agentes e o agente observador. Adicionalmente, replicação e falta de legibilidade impõem dificuldades de manutenção. Em função desse cenário não desejável, uma segunda evolução ocorreu, a qual proporcionou a criação de uma biblioteca de classes, ou API, de baixa intrusividade e orientada, principalmente, à rastreabilidade e à auditabilidade. Esse refinamento aprimorou a qualidade interna do sistema, reduzindo o acoplamento e aumentando a coesão dos componentes, bem como provendo maior reutilização de código, menor intrusividade, e maior legibilidade.

O artigo está organizado em Seções. Na Seção 2, são descritos os objetivos da pesquisa. As contribuições esperadas bem como os resultados obtidos são, respectivamente, apresentados nas Seções 3 e 4. Na Seção 5, as considerações finais são abordadas, seguidas pelas referências.

2 Objetivos da Pesquisa

A proposta aqui elucidada contempla vários objetivos específicos. Visando apresentar alguns desses objetivos, têm-se: (i) usar o conceito de transparência, bem como demais critérios de qualidade que impactam em transparência, para lidar com a socialização entre indivíduos, a qual é simulada com um sistema multiagentes; (ii) verificar o quanto apropriado é o uso desse conceito visando uma interação colaborativa entre os agentes que permita, por exemplo, auditar responsabilidades e manter o rastro das ações desempenhadas pelos agentes; (iii) prover uma biblioteca que facilite a implantação de transparência de software, e conceitos associados, em sistemas multiagentes; e (iv) prover suporte para a reutilização de boas práticas pela comunidade da Engenharia de Software.

3 Contribuições Esperadas

Conforme apresentado na Seção 2, o foco dessa pesquisa é melhorar o trabalho colaborativo entre os agentes de software usando como base o conceito de transparência bem como conceitos associados (ex. auditabilidade e rastreabilidade). Nesse cenário, além dos progressos já alcançados até o momento em experimentos no Laboratório de Engenharia de Software da UnB/FGA, os quais serão apresentados na Seção 4, ainda são vislumbradas: (i) a construção de um catálogo de padrões [7], contendo boas práticas em atendimento às atividades de elicitação de requisitos, projeto arquitetural e codificação de sistemas multiagentes que simulam a socialização entre indivíduos. Esses padrões serão reutilizáveis, e disponibilizados à comunidade no intuito de viabilizar a retroalimentação e a evolução sistemática e incremental do catálogo; (ii) a ampliação da pesquisa experimental, permitindo a comparação entre sistemas multiagentes autonômicos [8] orientados a comportamento [5] e orientados à meta [9][10]; (iii) a ampliação da

biblioteca de classes oferecida até o momento, provendo um *framework* para o desenvolvimento de sistemas multiagentes cada vez mais cognitivos, autonômicos, colaborativos, e preparados para execução em diferentes plataformas, incluindo dispositivos móveis. Vale ressaltar que os tópicos supracitados já se encontram em andamento pelos autores desse artigo.

4 Resultados Já Alcançados

O estudo de caso escolhido para ilustrar a aplicação de transparência de software em sistemas multiagentes, denominado PRIDE [11], foi desenvolvido com o intuito de simular um combate, onde os agentes de software representam lutadores e árbitros, responsáveis por diferentes etapas da simulação.

O objetivo final do PRIDE, que detemina se as interações entre os agentes foram bem-sucedidas, consiste na impressão de mensagens de texto significativas e aderentes ao contexto do sistema, em que o conjunto dessas mensagens relata os fatos ocorridos durante a simulação, como exemplificado no trecho 1.1.

Inicialmente, as mensagens eram impressas no console, em tempo de execução, pelos próprios agentes, como exemplificado no trecho 1.2, o qual representa a entrada de um lutador ou árbitro na luta.

Trecho 1.1: Relato do combate esperado

```
1 Anderson Silva hit the opponent's face!
2 Vitor Belfort was punched by Anderson Silva
```

Trecho 1.2: Impressão de mensagens pelos agentes em tempo de execução

```
1 DFService.register(this, dfd);
2 System.out.println(name + " enter the octagon!");
```

Essa abordagem atinge o objetivo de imprimir mensagens. Entretanto, dado que com essa impressão perde-se a referência para a mensagem, é inviável implementar qualquer aspecto correlato à transparência. Essas características também foram identificadas nos demais sistemas multiagentes, desenvolvidos ao longo do primeiro semestre de 2013 na disciplina de Paradigmas de Programação, os quais abordam diversos outros domínios, como controle de tráfego aéreo, recomendação de conteúdo, simulador biológico, dentre outros.

Nesse contexto, foi proposto o desenvolvimento de um agente observador, denominado *Broadcaster*, responsável por receber mensagens de outros agentes e armazenar as mensagens, imprimindo o relato do combate ao final das interações, como demonstrado no trecho 1.3.

Trecho 1.3: Relato do combate pelo Broadcaster

```
1 for (String message : events) System.out.println(message);
```

Essa troca de mensagens entre os agentes e o *Broadcaster* passou a ser realizada pelo envio de uma *ACLMessage*[12] e por um comportamento cíclico, demonstrados nos trechos 1.4 e 1.5, respectivamente, requerendo que os agentes substituissem a utilização do método *System.out.println(message)* pelo método *reportThat(message)*.

Trecho 1.4: Método reportThat()

```
ACLMessage report = new ACLMessage(ACLMessage.INFORM);
report.addReceiver(broadcaster);
report.setLanguage("report");
report.setContent(content);
myAgent.send(report);
```

Trecho 1.5: CyclicBehaviour para receber mensagens doas agentes

```
ACLMessage report = myAgent.receive(template);

if (report != null) events.add(report.getContent());
```

A implementação desse agente tornou possível a implementação de características relacionadas à transparência de software, como auditabilidade e rastreabilidade. Entretanto, o agente por sí só apresentava uma redução na qualidade interna, uma vez que o método *reportThat* precisou ser replicado em cada um dos agentes. Visando o aumento da legibilidade, da reutilização, e consequentemente de manutenção e generalização da solução, foi desenvolvida uma biblioteca composta por três classes fundamentais: (i) *Broadcaster*: implementação concreta do agente observador; (ii) *ReportBehaviour*: implementação concreta do envio de mensagens ao *Broadcaster*, dispensando a implementação por parte dos agentes; e (iii) *ObservableAgent*: implementação abstrata que já implementa o método para identificar *Broadcasters* ativos e o método *reportThat*.

Com essa estrutura, a implementação de qualquer agente transparente necessita apenas que esse agente herde a classe *ObservableAgent*, que um agente *Broadcaster* seja inicializado junto aos demais agentes, e que o método *reportThat(message)* seja utilizado quando se desejar enviar uma mensagem.

Durante a execução, o *Broadcaster* armazena informações sobre os agentes e sobre suas interações e compõe um relatório visando a implementação de critérios de qualidade correlatos à transparência.

5 Conclusão

Esse artigo descreve uma abordagem quanto ao uso do conceito de transparência de software na comunicação em sistemas multiagentes. Nessa proposta, destaca-se o desenvolvimento de uma biblioteca de classes, centrada no paradigma multiagentes, e orientada à rastreabilidade, acessibilidade e auditabilidade.

Dentre outras contribuições, têm-se: (i) a possibilidade de reutilização de código em aplicações com necessidades semelhantes bem como melhorias quanto à manutenibilidade, modularização, coesão e acoplamento; (ii) o desenvolvimento inicial de um catálogo de boas práticas no uso de transparência de software em sistemas multiagentes, em atendimento a diferentes critérios de qualidade, tais como rastreabilidade, acessibilidade e auditabilidade, e (iii) capacidade de retroalimentação da abordagem, uma vez que que os autores defendem a abertura da solução à comunidade, permitindo contribuições de terceiros.

Tomando como base alguns trabalhos relacionados [13] [14], temos que a solução proposta nesse artigo faz uso dos conceitos e fundamentos defendidos

pelos autores supracitados. Além disso, a proposta agrega ao tópico um suporte tecnológico reutilizável, aberto à contribuição de terceiros, e específico para transparência de software em contextos sociais baseados em diferentes domínios cognitivos e simulados por comunidades de agentes de software colaborativos.

No intuito de evoluir a abordagem proposta, os autores vislumbram, dentre outros trabalhos futuros: (i) o refinamento do catálogo de boas práticas; (ii) a melhoria cognitiva dos agentes, usando como base o paradigma da Orientação à Meta, e (iii) a ampliação da biblioteca de classes, acompanhando novas tendências tecnológicas e demandas observadas.

Referências

1. Holzner, B., Holzner, L.: Transparency in Global Change: The Vanguard of the Open Society. University of Pittsburgh Press, 1st edition (2006)
2. Leite, J.C.S.d.P., Cappelli, C.: Software transparency. Business & Information Systems Engineering **Vol. 2, nro. 3, pp. 127–139,** (2010)
3. Jennings, N.R.: On agent-based software engineering. Elsevier Science, Artificial Intelligence **Vol. 117, Issue 2, pp. 277-296,** (Mar. 2000)
4. Wooldridge, M.: An introduction to MultiAgent systems. 366 pages, (2002)
5. Bellifemine, F., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. John Wiley & Sons, ISBN 0470057475, 286 pages, (Apr. 2007)
6. Lehman, M.: Laws of software evolution revisited. EWSPT 1996, LNCS 1149, Springer Verlag, pp. 108–124, (1997)
7. Leal, A.L.d.C., et al;: Aplicação de modelos intencionais em sistemas multiagentes para estabelecer políticas de monitoração de transparência de software. RITA **Vol. 20, nro. 2, pp. 111–138,** (2013)
8. Rolón, M., Martínez, E.: Agent-based modeling and simulation of an autonomic manufacturing execution system. **Vol. 63, pp. 53-78,** (2012)
9. Braubach, L., Lamersdorf, W., Pokahr, A.: Jadex: Implementing a BDI infrastructure for JADE agents. Dist. Systems and Inf. Systems **Vol. 3, nro. 3, pp 76-85,** (2003)
10. Mylopoulos, J.: Goal-oriented requirements engineering. XI Conferencia Iberoamericana de Software Engineering, pp. 13-17, (Feb. 2008)
11. PRIDE: Disponível em: https://github.com/caiquerodrigues/pride-simulator
12. JADE: Documentation. disponível em: http://jade.cselt.it/doc/api/index.html
13. Iskanius, P., Helaakoski, H., Alaruikka, A.M., Kipina, J.: Transparent information flow in business networks by using agents. Proceedings of IEEE International Engineering Management Conference **Vol. 3, pp. 1342–1346** (2004)
14. Lam, D.N., Barber, K.S.: Comprehending agent software. Conference of Autonomous Agents and Multiagent Systems (AAMAS), pp. 25-29, (Jul 2005)
15. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. **Vol. 5 ISBN 978-0-7923-8666-7, 476 pages,** (2000)
16. Holmes, R., Walker, R.: Systematizing pragmatic software reuse. ACM Transactions on Software Eng. and Methodology **Vol. 21, nro. 4, 20 pages,** (2012)