PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 12/17

Capturando Proveniência de Dados em Sistemas Multiagentes

Tassio Ferenzini Martins Sirqueira
Marx Leles Viana
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 12/17 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Agosto, 2017

Capturando Proveniência de Dados em Sistemas Multiagentes

Tassio Ferenzini Martins Sirqueira[1,2], Marx Leles Viana[1], Carlos José Pereira de Lucena[1]

[1]Pontificia Universidade Católica do Rio de Janeiro (PUC-Rio)
[2]Instituto Vianna Junior (FIVJ)

tmsirqueira@vianna.edu.br, mleles@inf.puc-rio.br, lucena@inf.puc-rio.br

Abstract. This article addresses the capture of data provenance in multiagent systems. Initially the use of logs is discussed as a way to record the software information and its application in the BDI4JADE platform. During the presentation of the platform is explained the cycle of reasoning of the BDI and the functioning of the agents, where soon after it is discussed how to understand the actions and self-adaptations that agents can perform. At this point we discuss the use of log and how a provenance model may be a better approach. It is still discussed how this capture should be done, without the need to insert code in the multiagent platform and without affecting its architecture. Two codes are presented, the first one being extracted from BDI4JADE demonstrating the use of log and the other making a comparative with the use of aspects to capture the data provenance. At the end some consideration is given and the next steps of the research are presented.

Keywords: Data Provenance; Multiagent Systems; Provenance in SMA.

Resumo. Esse artigo aborda a captura de dados de proveniência em sistemas multiagentes. Inicialmente é discutido o uso de logs como um modo de registrar as informações do software e sua aplicação na plataforma BDI4JADE. Durante a apresentação da plataforma é explicada o ciclo de raciocínio do BDI e o funcionamento dos agentes, onde logo após discute-se sobre como compreender as ações e auto adaptações que os agentes podem realizar. Neste ponto são discutidos o uso de log e como um modelo de proveniência pode ser uma abordagem melhor. Ainda são discutidos como deve ser feita essa captura, sem a necessidade de inserção de código na plataforma multiagente e sem afetar sua arquitetura. São apresentados dois códigos, sendo o primeiro extraído do BDI4JADE demonstrando o uso de log e outro fazendo um comparativo com uso de aspectos para a captura dos dados de proveniência. Ao final são feitas algumas considerações e apresentados os passos seguintes da pesquisa.

Palavras-chave: Proveniência de Dados; Sistemas Multiagentes; Proveniência em SMA.

Responsável por publicações

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

1 Introdução

O termo "proveniência" refere-se à origem ou procedência dos dados, ou seja, é um registro do histórico de derivação de dados, que possibilita reprodutibilidade de experimentos, interpretação de resultados e diagnóstico de problemas (Lim *et al*., 2010). Em suma, a proveniência de dados é uma anotação ou um conjunto de meta-informações de um dado, que podem ser associados não apenas com produtos de dados, mas com os processos que permitem a criação destes dados.

Uma das formas mais comuns de registrar informações de um software é por meio do uso de logs. Por meio dos arquivos de log é possível registrar ações dos usuários, do sistema ou de agentes de software, visto que os logs podem registrar diversos acontecimentos de um sistema. Uma ferramenta bastante conhecida para geração de logs é a Log4J (GÜLCÜ, 2003). Contudo, a grande maioria dos sistemas de log seguem as mesmas características e o gerenciamento de eventos envolve um conjunto extenso de atividades, que são: i) armazenamento; ii) compactação; iii) indexação e busca; iv) análise léxica e sobre o tipo de evento; v) visualização e em alguns casos; vi)a transmissão e recebimento, visto que os logs podem ser armazenados de forma centralizada ou distribuída, sendo elas online ou off-line, como discutida por ARAÚJO *et al.* (2014).

Conforme ARAÚJO *et al.* (2014), dependendo do software o conjunto de logs pode ser extenso e distribuído em várias máquinas, podendo muitas vezes trazer dados insuficientes sobre o contexto do sistema que não está representando os eventos por completo. Ainda conforme ARAÚJO *et al.* (2014), considerando todos os diferentes contextos, é difícil correlacionar os eventos criando links lógicos que poderiam explicar o seu comportamento, sendo necessário mais informações sobre o comportamento sistema para entende-lo e os sistemas de log trazem apenas palavras-chaves em seus registros.

Lim *et al*. (2010), afirma que a proveniência é mais ampla e pode ser capturada de forma prospectiva e retrospectiva. A forma prospectiva captura a especificação abstrata, ou seja, uma receita para derivação de dados futuros. Já a proveniência retrospectiva captura a execução e, as informações de derivação dos dados fornecem informações importantes para a análise de resultados.

A proveniência de dados já é aplicada em vários sistemas de softwares, dos quais podemos destacar aplicações de E-Science, conforme discutido por SZOMSZOR e MOREAU (2003), JAMI e SHAIKH (2008) e SIRQUEIRA et al. (2016). Na área da computação em nuvem, como abordado por MUNISWAMY-REDDY et al. (2010) e EDWARDS et al. (2011). No campo das aplicações computacionais de E-Health, a proveniência foi aplicada para monitorar indivíduos submetidos a tratamento de modo que cada ação seja registrada e qualquer anomalia identificada, permitindo seguir cada etapa do processo (KIFOR et al., 2006), também a respostas de emergência, seja para a simulação de ambientes ou para auxiliar a tomada de decisão em situações de risco, conforme discutido em NAJA et al. (2010).

Em Garcia *et al.* (2002), os autores abordam que em softwares desenvolvidos com o paradigma de agentes de software, os comportamentos dos agentes podem evoluir ou se auto adaptarem para atender à novos requisitos do sistema. No entanto, compreender de forma mais clara os caminhos pelos quais agentes evoluem, ainda é um desafio. Um agente é uma entidade autônoma que busca realizar alguma tarefa à qual foi designado. Já um sistema multiagente (SMA) pode ser entendido como um conjunto de agentes autônomos, que buscam a solução de um problema que está além de suas capacidades individuais, conforme Jennings (1996).

De modo geral, para compreender essas evoluções em um sistema multiagente é necessário reunir informações suficientes de modo que as pessoas possam entender as execuções, abstraindo os detalhes irrelevantes e compreender às interações que ocorreram. Para isso, uma possibilidade é através do uso de proveniência de dados, que deve ser tratada como algo importante e útil, registrando cada mudança nos dados, conforme Sirqueira *et al*. (2017).

Desse modo, a proveniência fornece um olhar além das especificações dos domínios e sugere a adoção de modelos disciplinados, onde a informação de proveniência de dados pode ser usada para aprender ou compreender métodos e regras de design (Sirqueira *et al.*, 2017). Como abordado por Dalpra *et al.* (2015), o aumento de dados gerados torna a análise mais complexa e exige o uso de técnicas para permitir a análise adequada desses dados, extraindo registros que, de fato, contribuam e uma maneira de analisar esses dados é usar técnicas e modelos de proveniência. Atualmente, existem dois modelos principais de proveniência: i) o modelo OPM (Moreau *et al.*, 2008), com três vértices, cinco relações causais, e ii) o modelo PROV (Missier *et al.*, 2013), com três vértices principais e sete relações básicas, mais algumas complementares.

O trabalho aqui proposto, objetiva abordar sobre a captura de dados de proveniência em sistemas multiagentes, tendo como base a plataforma BDI4JADE (NUNES *et al.*, 2011). Para isso, na seção 2, será apresentado brevemente a plataforma e discutido como os dados devem ser capturados sem impactar na arquitetura e funcionamento da mesma. Ainda na seção 2 será discutido o uso de aspectos como uma solução para a captura dos dados de proveniência na plataforma, junto com seus pontos fortes e fracos dessa abordagem. Na seção 3 serão abordadas as vantagens da proveniência em relação aos logs e sua captura por meio de aspecto e, por fim, a seção 4 apresenta as considerações finais e os trabalhos futuros.

2 Plataforma BDI4JADE

O BDI4JADE (NUNES *et al.*, 2011) é uma plataforma de agentes implementados seguindo a arquitetura BDI (Belief-Desire-Intention) (RAO & GEORGEFF, 1995) sobre a plataforma JADE (BELLIFEMINE *et al.*, 2007).

O modelo BDI foi originado por Bratman (1987), como uma teoria para explicar o raciocínio humano de modo prático, baseado em crenças, desejos e intenções. Esse raciocínio prático é composto de duas etapas, sendo elas a de deliberação, onde são selecionados desejos do agente de acordo com a situação atual para alcançar um determinado objetivo e a segunda etapa a de atuação, onde determina ações que devem ser executadas para alcançar o resultado determinado no passo anterior.

Esse modelo de raciocínio prático define que as crenças representam as características do ambiente, que podem evoluir ao longo do tempo; já os desejos representam os objetivos a serem alcançados, junto com suas propriedades e custos de cumprir ou não com o objetivo; e as intenções representam os planos de ação para alcançar um objetivo especifico do sistema.

Como explicado por Nunes *et al.* (2011), o raciocínio prático de um agente BDI começa pela função de revisão das crenças, verificando as crenças que estão sendo inseridas em relação as que já existem. Tendo os agentes as crenças atuais determinadas, é gerada as opções disponíveis para um desejo do agente, levando em conta suas crenças e intensões, onde essas opções representam as possíveis ações que um agente pode tomar para cumprir com seus objetivos. Após esse processo é feito um filtro de opções para determinar as intenções do agente, que ao se comprometer com uma intenção seleciona a ação

que será executada, conforme pode ser observado na Figura 1, representando o ciclo de raciocínio BDI.

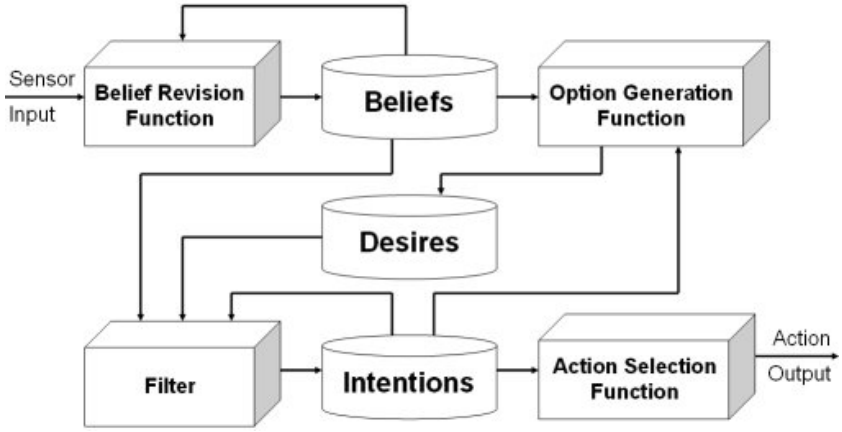

Figura 1. Raciocínio do BDI.

Acompanhar esses passos para cada agente no sistema possibilita compreender suas ações, suas adaptações e explicar os acontecimentos e evoluções no sistema. Contudo, como abordado por Sirqueira *et al.* (2017), deve-se ter um conjunto de informações que abordem as ações executadas, as crenças, desejos e intenções em cada momento e sua autonomia, garantindo que todas as informações relevantes sejam capturadas junto de seus resultados.

3 Captura de Dados de Proveniência

O BDI4JADE (NUNES *et al.*, 2011) faz uso do LOG4J (GÜLCÜ, 2003) para registrar os acontecimentos da plataforma. Um exemplo de uso de log pode ser visto na Listagem 1, do código extraído da classe *BDIAgent.java* do BDI4JADE versão 2.0. Nesse exemplo, tem-se o uso do "*log.trace*" para registrar qual trecho de código que já foi executado dentro do "*Behaviour*" do agente, entretanto, como pode ser observado no trecho de código, não são registradas informações sobre a qual agente esse comportamento pertence, quem executou essa chamada, o tempo que durou e os processos intermediários que foram executados para cumprir com essa ação, o que torna a utilidade do log baixa, uma vez que não explica claramente o comportamento do agente no software.

Listagem 1. Uso de logs no BDI4JADE.

```
@Override
public void action() {
        log.trace("Beginning BDI-interpreter cycle.");
        log.trace("Reviewing beliefs.");
        beliefRevisionStrategy.reviewBeliefs(BDIAgent.this);
        synchronized (intentions) {
                Map<Goal, GoalStatus> goalStatus = new HashMap<Goal, GoalStatus>();
                Iterator<Intention> it = intentions.iterator();
                while (it.hasNext()) {
                        Intention intention = it.next();
                        GoalStatus status = intention.getStatus();
                        switch (status) {
                        case ACHIEVED:
                        case NO_LONGER_DESIRED:
                        case UNACHIEVABLE:
                                intention.fireGoalFinishedEvent();
                                it.remove();
                                break;
                        default:
                                goalStatus.put(intention.getGoal(), status);
                                break;
                        }
                }
```

```
                    Set<Goal> generatedGoals = optionGenerationFunction
                                    .generateGoals(goalStatus);
                    Set<Goal> newGoals = new HashSet<Goal>(generatedGoals);
                    newGoals.removeAll(goalStatus.keySet());
                    for (Goal goal : newGoals) {
                            addGoal(goal);
                    }
                    Set<Goal> removedGoals = new HashSet<Goal>(goalStatus.keySet());
                    removedGoals.removeAll(generatedGoals);
                    for (Goal goal : removedGoals) {
                            it = intentions.iterator();
                            while (it.hasNext()) {
                                    Intention intention = it.next();
                                    if (intention.getGoal().equals(goal)) {
                                            intention.noLongerDesire();
                                            intention.fireGoalFinishedEvent();
                                            it.remove();
                                    }
                            }
                    }
                    goalStatus = new HashMap<Goal, GoalStatus>();
                    for (Intention intention : intentions) {
                            goalStatus.put(intention.getGoal(), intention.getStatus());
                    }
                    Set<Goal> selectedGoals = deliberationFunction
                                    .filter(goalStatus);
                    log.trace("Selected goals to be intentions: "
                                    + selectedGoals.size());
                    for (Intention intention : intentions) {
                            if (selectedGoals.contains(intention.getGoal())) {
                                    intention.tryToAchive();
                            } else {
                                    intention.doWait();
                            }
                    }
                    if (intentions.isEmpty()) {
                            log.trace("No goals or intentions - blocking cycle.");
                            fireStateIdle();
                            this.block();
                    }
            }
            log.trace("BDI-interpreter cycle finished.");
}
```

Nesse exemplo, a proveniência pode ser implementada por meio de aspectos, onde não será necessário modificar a arquitetura da plataforma multiagente para a inserção da captura dos traços de procedência e com isso a proveniência é tratada como uma preocupação transversal, de modo que a plataforma não sofre nenhuma alteração na sua estrutura, e a proveniência atua nos pontos de execução identificáveis entre as funções, registrando cada acontecimento. Essa captura deve compreender na plataforma multiagente e no raciocinador BDI, conforme a Figura 2.

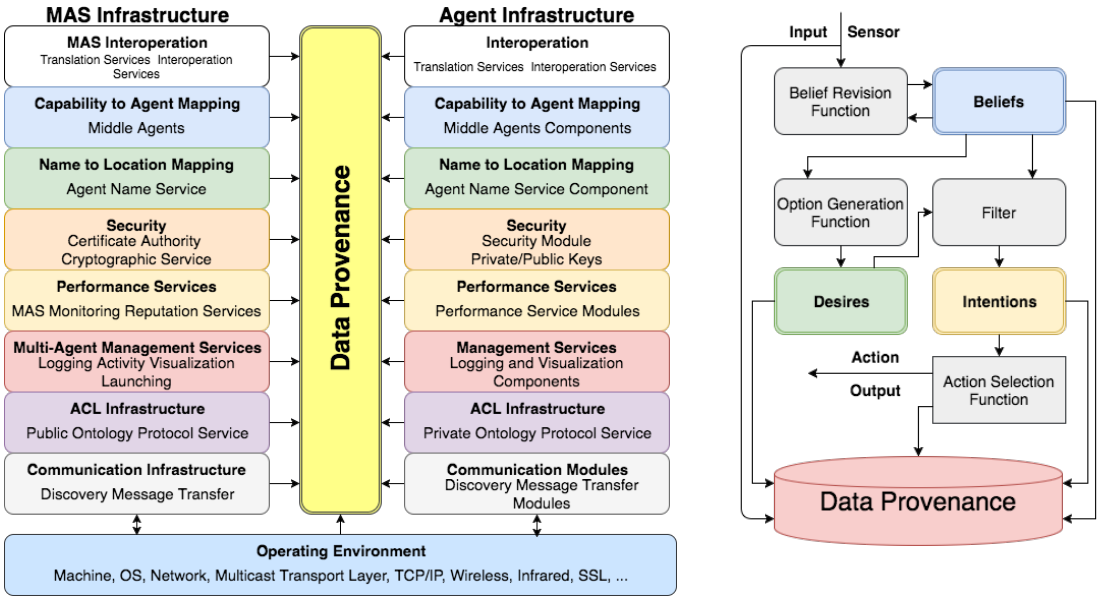

Figura 2. Captura de proveniência na plataforma multiagente e no BDI.

A captura de dados de proveniência por meio de aspectos na plataforma multiagente possibilita que: i) a arquitetura da plataforma não sofra alterações; ii) possibilita interceptar qualquer ponto de execução da plataforma; iii) qualquer falha no processo de captura da proveniência não afeta a plataforma; iv) possibilita vincular os eventos que ocorrem por meio de links seguindo o modelo PROV; v) garante reuso do código de captura em qualquer software que utilize a mesma plataforma de sistema multiagente; vi) facilita o processo de manutenção da captura dos dados e; vii) é de fácil acoplamento com a plataforma multiagente. Contudo, também podem surgir alguns pontos negativos com o uso dessa abordagem como: i) a garantia que o ponto de corte está ocorrendo no local certo; ii) a conformidade que o dado capturado está correto; iii) o excesso de chamadas a funções que podem ocorre durante a captura da proveniência e; iv) tornar o código complexo após a integração da plataforma com os aspectos usados para a captura de proveniência.

O acesso aos dados da plataforma multiagente e o fato de não ser necessário a inserção de código para a captura da proveniência junto ao código original do software, dispensa do programar a função de inserir a captura da proveniência em meio ao seu código, reduzindo seu trabalho e evita a captura dos dados de maneira equivocada, restringe que os detalhes relevantes estejam de fato sendo capturados e corretamente armazenados. Em exemplo da captura dos dados de proveniência na plataforma BDI4JADE por meio de aspecto e usando o framework FProvW3C proposto por Sirqueira *et al.* (2017) pode ser visto na Listagem 2.

Listagem 2. Capturando proveniência por meio de aspecto no BDI4JADE.

```
@Aspect
public class PROVBDIAgent {
        @Around("execution(* br.pucrio.inf.les.bdi4jade.core.BDIAgent.BDIInterpreter.action(..))")
        public void PROVBDIAgentAction(ProceedingJoinPoint joinPoint) throws Throwable {
                ProvActivity ac = new ProvActivity();
                Timestamp startT = new Timestamp(System.currentTimeMillis());
                ac.setStartTime(startT);
                joinPoint.proceed();
                Timestamp endT = new Timestamp(System.currentTimeMillis());
                ac.setEndTime(endT);
                ac.setDescription("BDI-interpreter cycle");
                new ProvActivityDAO().save(ac);
        }
}
```

No exemplo da Listagem 2, está descrita a captura da atividade "*action*" apresentada na Listagem 1, contudo, além desses dados, para o exemplo apresentado na Listagem 1, deveríamos capturar as chamadas que manipulam as crenças e os objetivos do agente e seu processo deliberativo, referenciando a atividade que ocorreu, o agente que a executou e a entidade que foi impactada pela ação, além da ocorrência de algum erro caso exista.

Esse trabalho é o ponto de início para a discussão sobre a captura e o uso de proveniência de dados por meio de aspecto em sistemas multiagentes, pois entender o comportamento e a evolução de um sistema multiagente nos permite ir mais longe, criando software complexos mais estáveis, seguros e de qualidade.

4 Considerações Finais e Trabalhos Futuros

A proveniência de dados possibilita registrar informações em diferentes níveis de abstração e seu uso por meio de aspectos e uma possibilidade para que os dados sejam capturados sem que haja a necessidade de inserção direta de código na plataforma multiagente.

Atualmente algumas plataformas apresentam o uso de logs, de maneira simples e que não permite uma clara compreensão da execução do sistema em softwares complexos. Contudo, a proveniência dos dados pode ser usada na construção de bases de conhecimento do sistema multiagente que pode ajudar na: i) rastreabilidade das ações; ii) identificação de erros; iii) acompanhamento das etapas de um sistema, e iv) determinação da viabilidade de análise e, v) verificação dos resultados. Hoje em sistemas multiagentes há muitas oportunidades para aplicar a proveniência dos dados e esse trabalho é o passo inicial da discussão sobre a captura dos dados de proveniência em sistemas multiagentes e como isto deve ser feito.

Os próximos passos da pesquisa é mapear a plataforma BDI4JADE e determinar os aspectos importantes à serem capturados, utilizando o framework FProvW3C para auxiliar na modelagem e armazenamento da proveniência. Vale ressaltar que ao usar a proveniência de dados para capturar os dados em um sistema multiagente, será possível suportar: i) semântica e sintaxe de dados, ii) ontologia, iii) mineração de dados e iv) exportação de dados para outros formatos, como XML ou JSON.

Registrar a proveniência de dados é muitas vezes uma necessidade em cenários que possuem execução complexa, tornando o histórico de cada passo uma atividade fundamental. O modelo PROV permite armazenar dados de procedência de forma detalhada, focalizando as responsabilidades dos agentes em cada item de proveniência.

Agradecimentos

Os autores agradecem a CAPES, Faperj, CNPq, PUC-Rio, FIVJ e LES pelo apoio e incentivo na pesquisa.

Referências

ARAÚJO, T. P.; CERQUEIRA, R. and STAA A. V., "Supporting failure diagnosis with logs containing meta-information annotations". **Technical Reports in Computer Science**. PUC-Rio. ISSN 0103-9741, vol. 14, p. 21, 2014.

BELLIFEMINE, Fabio Luigi; CAIRE, Giovanni; GREENWOOD, Dominic. **Developing multi-agent systems with JADE**. John Wiley & Sons, 2007.

BRATMAN, Michael. Intention, plans, and practical reason. 1987.

DALPRA, Humberto LO et al. Using Ontology and Data Provenance to Improve Software Processes. **Proceedings of the Brazilian Seminar on Ontologies**, p. 10-21, 2015.

EDWARDS, Pete; PIGNOTTI, Edoardo; CORSAR, David. Provenance on the Web, Leaving the Walled Garden Behind.. In: **Proceedings of the 3rd International ACM Conference on Web Science**. Web Science Trust, 2011.

GARCIA, Alessandro et al. Engineering multi-agent systems with aspects and patterns. **Journal of the Brazilian Computer Society**, v. 8, n. 1, p. 57-72, 2002.

GÜLCÜ, Ceki. **The complete log4j manual**. QOS. ch, 2003.

JAMI, S.; SHAIKH, Zubair. A multi-agent-based architecture for data provenance in semantic grid. In: **Proceedings of the International Multi-Conference of Engineers and Computer Scientists (IMECS).** Hong Kong: Newswood Publications. 2008.

JENNINGS, Nick R. Coordination techniques for distributed artificial intelligence. **Foundations of distributed artificial intelligence**, p. 187-210, 1996.

KIFOR, Tamas et al. Provenance in agent-mediated healthcare systems. **IEEE Intelligent Systems**, v. 21, n. 6, p. 38-46, 2006.

LIM, Chunhyeok et al. Prospective and retrospective provenance collection in scientific workflow environments. In: **2010 IEEE International Conference on Services Computing (SCC)**. IEEE, 2010. p. 449-456.

MISSIER, Paolo; BELHAJJAME, Khalid; CHENEY, James. **The W3C PROV family of specifications for modelling provenance metadata**. In: Proceedings of the 16th International Conference on Extending Database Technology. ACM, 2013. p. 773-776.

MOREAU, Luc et al. **The open provenance model: An overview**. In: International Provenance and Annotation Workshop. Springer Berlin Heidelberg, 2008. p. 323-326.

MUNISWAMY-REDDY, Kiran-Kumar; MACKO, Peter; SELTZER, Margo I. Provenance for the Cloud. In: **FAST**. 2010. p. 15-14.

NAJA, Iman; MOREAU, Luc; ROGERS, Alex. Provenance of decisions in emergency response environments. **Provenance and Annotation of Data and Processes**, p. 221-230, 2010.

NUNES, Ingrid; LUCENA, C. J. P. D.; LUCK, Michael. BDI4JADE: a BDI layer on top of JADE. **ProMAS 2011**, p. 88-103, 2011.

RAO, Anand S & GEORGEFF, M. P. BDI Agents: From Theory to Practice. In: **ICMAS**. 1995. p. 312-319.

SIRQUEIRA, Tassio et al. A Software Framework for Data Provenance. In: **29th International Conference on Software Engineering & Knowledge Engineering (SEKE'2017)**. SEKE/Knowledge Systems Institute, 2017. p. 615-619.

SIRQUEIRA, Tassio FM et al. E-seco proversion: An approach for scientific workflows maintenance and evolution. **Procedia Computer Science**, v. 100, p. 547-556, 2016.

SZOMSZOR, Martin; MOREAU, Luc. Recording and reasoning over data provenance in web and grid services. In: **Coopis/doa/odbase**. 2003. p. 603-620.