PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 03/2019

Sistemas Autônomos Explicáveis por meio de Proveniência de Dados

Tassio Ferenzini Martins Sirqueira
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. 03/2019 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Abril, 2019

Sistemas Autônomos Explicáveis por meio de Proveniência de Dados

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

[1] Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
[2]Centro de Ensino Superior de Juiz de Fora (CES/JF)

tassiosirqueira@cesjf.br, lucena@inf.puc-rio.br

Abstract. Determining the data provenance, that is, the process that led to those data, is vital in many areas, especially when it is essential that the results or actions be reliable. With the increasing number of applications based on artificial intelligence, the need has been created to make them capable of explaining their behavior and be responsive to their decisions. This is a challenge especially if the applications are distributed, and composed of multiple autonomous agents, forming a Multiagent System (MAS). A key way of making such systems explicable is to track the agent's behavior, that is, to record the source of their actions and reasoning, as in an "omniscient debugging". Although the idea of provenance has already been explored in some contexts, it has not been extensively explored in the context of MAS, leaving many questions to be understood and addressed. Our objective in this work is to justify the importance of the data provenance to MAS, discussing which questions can be answered regarding the behavior of MAS using the provenance and illustrating, through application scenarios, to demonstrate the benefits that provenance provides to reply to these questions. This study involves the creation of a software framework, called FProvW3C, which supports the collects and stores the provenance of the data produced by the MAS. This data can then be analyzed to answer a wide variety of questions that allows the understanding of the MAS behavior. The objective of this work is to show rigorously that the use of the data provenance in MAS is a sound solution to make the agent's reasoning / action process transparent.

Keywords: Data Provenance; Multiagent System; FProvW3C; Explainable Artificial Intelligence; Provenance in MAS software framework.

Resumo. Determinar a proveniência dos dados, isto é, o processo que levou a esses dados, é vital em muitas áreas, especialmente quando é essencial que os resultados ou ações sejam confiáveis. Com o crescente número de aplicações baseadas em inteligência artificial, criou-se a necessidade de torná-las capazes de explicar seu comportamento e responder às suas decisões. Isso é um desafio, especialmente se as aplicações forem distribuídas e compostas de vários agentes autônomos, formando um Sistema Multiagente (SMA). Uma maneira fundamental de tornar tais sistemas explicáveis é rastrear o comportamento do agente, isto é, registrar a origem de suas ações e raciocínios, como em uma "depuração onisciente". Embora a ideia de proveniência já tenha sido explorada em alguns contextos, ela não foi extensivamente explorada no contexto de SMA, deixando muitas questões para serem compreendidas e abordadas. Nosso objetivo neste trabalho é justificar a importância da proveniência dos dados para SMA, discutindo quais perguntas podem ser respondidas em relação ao comportamento do SMA, utili-

zando a proveniência e ilustrando, através de cenários de aplicação, os benefícios que a proveniência proporciona para responder a essas questões. Este es- tudo envolve a criação de um framework de software, chamado FProvW3C, que suporta a coleta e armazenamento da proveniência dos dados produzi- dos pelo SMA. Esses dados podem ser analisados para responder a uma ampla variedade de perguntas que permitem a compreensão do comportamento do SMA. O objetivo deste trabalho é mostrar com rigor que, o uso da proveniência de dados em SMA é uma solução sólida, para tornar transparente o processo de raciocínio / ação do agente.

Palavras-chave: Proveniência de Dados; Sistema Multiagente; FProvW3C; Sistemas Autônomos Explicáveis; 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

Sistemas computacionais já fazem parte do nosso cotidiano, e para o desenvolvimento de novos sistemas para atender uma demanda cada vez mais pujante, por softwares eficazes e eficientes, dependemos de soluções que possam ser escaláveis, que possam lidar com o crescimento explosivo dos dados (Big Data), e que sejam adaptadas para à internet e também para o chamado "Internet das Coisas" (1).

Dado essas novas necessidades, a inteligência artificial (IA), representada por sistemas multiagentes, alinhada com técnicas de aprendizado de máquina (ML), demonstram sucesso prático em muitos domínios de aplicação, como por exemplo, em sistemas autônomos de direção, reconhecimento de fala ou de recomendação, conforme (22). Com uma série de atividades ocorrendo de forma autônoma, o mundo passa a manipular grandes quantidades de dados, que devem ser atualizados em tempo real, e que de modo geral, são utilizados por milhares de pessoas, dispositivos ou outros sistemas autônomos.

A introdução de componentes autônomos aos novos sistemas, muitas vezes passam a ser um diferencial primordial ao novo cenário em que vivemos (4) e a quebra do paradigma de desenvolvimento de sistemas simplesmente CRUDs (Create, Read, Update, Delete), emerge o uso de inteligência artificial, aprendizado de máquina, agentes de software ou mesmo de sistemas multiagentes, que podem ser entendidos como inteligência artificial distribuída, onde um agente deixa de ser um simples objeto do ponto de vista da engenharia de software (ES) (2, 3).

Conforme definido por (6) "Um agente é um sistema de computador que está situado em algum ambiente e que é capaz de ação autônoma nesse ambiente para atingir seus objetivos de projeto".

Ao longo dos últimos anos, o uso de agentes de software e sistemas multiagente tem crescido por conta da demanda dos novos sistemas (5), com necessidades de automação e da inserção de IA e ML. A inteligência artificial é um grande campo da computação que engloba lógica, probabilidade, matemática, entre outras áreas, para que as máquinas sejam capazes de resolverem problemas que até então eram reservados para os humanos. Essa 'inteligência artificial' é obtida a partir da interseção entre o Big Data, a computação distribuída (ou computação em nuvem) e a modelos de dados eficientes.

Com isso, nos últimos anos a Inteligência Artificial Explicável (XAI) tem recebido significativa atenção (25, 22, 24, 26), devido à necessidade de sistemas capazes de explicar seu comportamento e serem responsáveis por suas decisões e ações, principalmente quando tais decisões têm um impacto significativo na vida dos seres humanos.

Com um aumento no desenvolvimento de sistemas complexos, fornecendo algum grau de inteligência, os agentes de software tornaram-se mais frequente- mente incorporados em plataformas comerciais e industriais. A escolha de desenvolver sistemas complexos usando a tecnologia de agente de software, assim como as linguagens, plataformas e técnicas associadas, se deve ao fornecimento de abordagens reutilizáveis, prontas para implementar recursos desafiadores, como pro atividade, raciocínio, aprendizado e comunicação distribuída.

Esses sistemas autônomos podem funcionar isolados ou serem uma coleção de sistemas autônomos heterogêneos, que colaboram para atingir um objetivo em específico ou objetivos comuns ao grupo. Os sistemas autônomos que são compostos de várias entidades individuais heterogêneas são geralmente chamados de Sistemas Multiagentes (SMA) (23).

O problema central do uso de sistema multiagentes, é que eles são considerados uma caixa preta e mesmo que entendamos os princípios matemáticos subjacentes dos agentes, falta uma representação declarativa explícita do conhecimento, portanto, temos dificuldade em gerar estruturas explicativas subjacentes.

Um meio-chave de tornar esses sistemas multiagentes explicáveis é rastrear o comportamento dos agentes, isto é, registrar de forma clara suas ações e raciocínios. Embora a ideia de proveniência de dados tenha sido explorada em alguns contextos, principalmente e-science, ela tem sido pouco explorada no contexto de sistemas autônomos, deixando muitas questões em aberto que devem ser compreendidas e abordadas.

Esse trabalho visa o uso de proveniência de dados em sistemas autônomos, onde nas próximas seções deste capítulo serão definidos os problemas a serem atacados, os objetivos com está pesquisa e a abordagem que será utilizada.

1.1 Definição do Problema

Um agente é uma entidade autônoma que busca realizar alguma tarefa à qual foi designado. Entretanto, desenvolver software orientado a agente com apenas um agente não é o suficiente, pois uma das características fundamentais dos agentes é a comunicação.

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 (9). Conforme (10) explica, sistemas multiagentes é uma abordagem promissora para a construção de sistemas complexos, contudo, manter a qualidade do software é uma atividade árdua, devido a complexidade natural do software e pelas interações que ocorrem entre os agentes.

Sistemas multiagentes podem ser projetados sobre estruturas homogêneas, heterogêneas ou híbrida (10), onde a arquitetura do sistema multiagente deve permitir que os agentes interajam, para garantir a funcionalidade do sis- tema. A arquitetura de um agente determina sua estrutura e modo de operação, que podem ser classificados com base na aplicabilidade, na forma de interação e no grau de "inteligência" do agente, onde temos as classificações dispostas na Tabela 1.1 de (11).

Na classificação de (11) podemos concluir que a arquitetura deliberativa segue um modelo simbólico do mundo, guiado por crenças, desejos e intenções, chamado de modelo BDI (Belief-Desire-Intention) (12). Na arquitetura reativa não existe raciocínio, apenas respostas a estímulos externos e já a arquitetura híbrida, busca um equilíbrio entre as duas anteriores.

Além da classificação por arquitetura, os agentes podem ser classificados por dois tipos principais: i) agentes que não apresentam inteligência - classifica- dos apenas como agentes de software e; ii) agentes que apresentam algum grau de inteligência - classificados como agentes de software inteligentes (13). Essas classificações buscam definir padrões para implementação usando sistemas multiagentes.

Dentro do projeto de software, a arquitetura é um ponto entre a análise de requisitos e a implementação do sistema. A arquitetura provê uma visão global do sistema, o que permite: i) compreender o funcionamento do sistema; ii) especificar pontos críticos; iii) identificar possibilidade de reuso; iv) verificar e validar o sistema e; v) definir critérios de qualidade que permitam o sistema evoluir posteriormente. Todo software é construído sobre alguma abstração e em softwares que em sua arquitetura utiliza um sistema multiagente, a abstração pode ser o próprio agente (14).

Tabela 1.1: Classificação das arquiteturas de agentes

Arquitetura de Agentes	Variação do nome	Descrição
Arquitetura Deliberativa	Arquitetura BDI Arquitetura baseada em lógica Arquitetura baseada em metas	Possui um modelo simbólico do mundo. As decisões são tomadas via raciocínio lógico. Possui um conjunto de metas e intenções, onde é elaborado um plano baseando-se nessas metas. Possui certas restrições sobre a construção do modelo simbólico. Agentes complexos.
Arquitetura Reativa	Arquitetura reflexiva	Sem modelo simbólico interno. As decisões tomadas são implementadas em alguma forma de mapeamento direto da situação para a ação, usando regras de condição/ação.
Arquiteturas híbridas	Arquiteturas em camadas	As decisões tomadas via várias camadas de software. Mistura componentes das arquiteturas Deliberativa e Reativa.

A escolha de programar sistemas autônomos complexos usando agentes de software é devido a algumas de suas características, como autonomia, pró- atividade e auto adaptação. Contudo, para avaliar um sistema é necessário reunir informações suficientes de modo que as pessoas possam entender as execuções, abstraindo os detalhes irrelevantes e compreender as interações que ocorreram.

Os sistemas que utilizam agentes autônomos inteligentes têm qualidades específicas que exigem que os usuários justifiquem suas atividades, uma vez que as decisões tomadas por esse sistema devem ser consideradas confiáveis, especialmente se o sistema for amplamente utilizado (15). Em aplicações autônomas, a confiabilidade dos dados processados por outros sistemas também deve ser confiável, pois acaba impactando o resultado geral de um processo. Além disso, as aplicações podem possuir pontos de entrada onde as pessoas que usam o sistema podem inserir dados.

Passada as fases de análise de requisitos e de implementação do sistema, é comum a criação de testes de software. Entretanto, se considerarmos sistemas autônomos complexos, com uso de agentes inteligentes, a criação de testes que cubram as ações dos agentes se tornam inviáveis. Shaw e Bordini (16) demostram em seu trabalho que ana-

lisar a árvore de planos de objetivos e que verificar se uma árvore de plano de metas tem um cronograma de execução com relação aos requisitos de recursos é NP-completo. Logo, não existe ainda uma solução para verificar em tempo polinomial as ações do agente presente na base do BDI, sendo assim, soluções como as de (27, 59, 60), tornam-se a mais viáveis neste caso, sendo pontual e parcial em relação a todas as ações que podem ocorrer dentro de um SMA.

Isso levará a intuição de que "agentes autônomos de software são difíceis de testar" (17), tanto com relação aos possíveis comportamentos dos agentes BDI, quanto à sua probabilidade de falhar. Essa definição derrogam a visão global das propostas supramencionadas, e impede que as pessoas possam compreender certas ações tomadas pelos sistemas multiagentes, subtraindo a confiabilidade de suas ações e levando a questões como as apresentadas por (18), que são:

1. Quem foi responsável pelo efeito X?
2. O efeito X corresponde ao que se destinava a acontecer?
3. Qual é o motivo (causal e intencional) do efeito X?

Além de outras questões envolvendo a interação dentro dos sistemas multiagentes e a seguridade de suas ações, tais como:

4. Quais foram as fontes de informação que alteraram o comportamento de um agente?
5. Quais agentes interagem uns com os outros?
6. Qual o ambiente em que o agente está inserido e o agente está ciente do meio ambiente?
7. Quais são as crenças, desejos e intenções do agente em cada momento (de falha ou sucesso)?
8. O agente está tomando ações verdadeiramente autônomas?

Para responder a estes tipos de perguntas, é necessário registrar informações sobre o comportamento do agente e do sistema multiagente no qual está inserido, e para isso, este trabalho apresenta como uma abordagem viável, o uso de proveniência de dados.

Com o uso da proveniência é possível registrar as informações e em cada situação atípica (como uma falha), podemos identificar quais dados foram recebidos e as ações tomadas pelo sistema autônomo antes da situação atípica ocorrer.

Um exemplo real em que a informação de proveniência poderia ajudar a explicar, é o caso que ocorreu com o carro autônomo da Uber (28). Nesse caso, analisando os dados coletados pelos sensores e o "raciocínio" do carro antes do acidente, seria possível identificar de onde surgiu a falha que culminou com a morte de uma ciclista. Em cenários críticos, como o de veículos autônomos, a proveniência dos dados e o histórico gerado se tornam essenciais.

No próximo capítulo teremos uma visão geral sobre sistemas de Logs e de modelos de proveniência de dados, e suas aplicabilidades, além de iniciarmos uma discussão sobre as vantagens dos modelos de proveniência em relação aos de Logs.

1.2 Objetivos da Tese

O ciclo de vida de um software é designado pelo seu processo de desenvolvimento, manutenção e evolução, culminando com seu decaimento, buscando refletir o mundo real que está em constante mudança. O processo de desenvolvimento e evolução de um

software visa atender aos requisitos especificados no seu projeto, o que muitas vezes pode não ocorrer por falha humana nesse pro- cesso, seja por uma especificação incorreta ou um desenvolvimento inadequado as especificações.

A atividade de teste, conforme (19), consiste em uma análise dinâmica do software e é relevante para a identificação e eliminação de defeitos que causarão falhas observadas. Um teste é essencialmente a aplicação de diversos exemplos de uso. Para cada exemplo existe um "oráculo" capaz de informar se o teste falhou ou não. Caso tenha falhado, precisa-se localizar e remover o defeito causador. Evidentemente, se a escolha dos exemplos for superficial, há uma elevada probabilidade de defeitos remanescerem. Contudo, conforme mencionada na seção anterior, em sistemas autônomos complexos, com uso de um sistema multiagente, é impossível testar todas as condições dos agentes cobrindo todos as planificações geradas pelo BDI.

A complicação de se testar agentes autônomos de software (ou agentes inteligentes de software), alinhado com o tamanho dos sistemas que a cada dia estão maiores e mais complexos, demandam maior atenção e dedicação dos desenvolvedores para conservar em um nível de qualidade satisfatório, que se agravam em contextos envolvendo sistemas multiagentes, visto que as realizações de determinadas tarefas serão designadas a agentes autônomos de software e esses podem se auto adaptarem em determinadas condições do sistema, visto que esse é uma característica fundamental do agente e será melhor explicada em um capítulo posterior.

Com isso, compreender de forma mais clara os caminhos pelos quais os softwares evoluem, de forma que possam ser modificados com maior facilidade e sua qualidade não seja prejudicada ainda é um desafio (20). Uma das formas mais comuns de registrar informações do sistema para compreender sua execução ou evolução é através do uso de logs.

A geração de logs é a geração de registros de eventos que proveem informações sobre a utilização do software à medida que acontecem e preservam esses registros para análise posterior.

Entretanto, alguns pontos sobre o uso dos tradicionais sistemas de logs como o Log4J (21) ainda estão em aberto, como, por exemplo, o que deve ser registrado, a frequência desse registro, o que deve conter e etc.

Considerando esses aspectos, o objetivo deste trabalho é demostrar que os sistemas de logs como o Log4J, que é utilizado em sistemas autônomos e que envolvem o uso de sistemas multiagentes, não são suficientes para coletarem dados que demostram de forma clara e precisa os acontecimentos do sistema.

Retomando a ideia apresentada na seção anterior, como é impossível verificar em tempo hábil as assertivas em um sistema multiagente por meio de testes. Para compreender as ações do sistema, identificar falhas ou adaptações do sistema e tornar esse processo inteligível, necessitamos recorrer a meios melhores do que os sistemas de logs, onde nesse trabalho buscamos o uso de proveniência de dados, tornando os sistemas autônomos explicáveis e baseado nas características supracitadas, responder as questões da seção anterior, além de outras que serão apresentadas à frente.

Na próxima serão apresentaremos a abordagem proposta e as motivações que levaram à mesma.

1.3 Abordagem Proposta

Conforme (26), existe um desafio histórico da IA, que é: i) Determinar quais são as representações apropriadas de conhecimentos que demonstram alguma veracidade

com o domínio que está sendo capturado; e ii) Quais mecanismos de raciocínio oferecem a base para transmitir uma inferência computada em termos desse modelo.

A proveniência de dados é um tipo de meta dado gerado por aplicações ou processos computacionais e descrevem os produtos de dados utilizados ou gerados no processo, permitindo que os dados sejam desambiguados e reutiliza- dos (31). O gerenciamento de dados está crescendo em complexidade conforme novos aplicativos surgem com recursos fracamente acoplados, envolvendo sistemas descentralizados, processamento em rede e fontes abundantes de informação.

Dados essas características, um mecanismo para capturar dados de proveniência precisa acessar detalhes relevantes de uma tarefa computacional, registrando seus passos, informações de execução e anotações do usuário (32). Isso é importante porque a proveniência de dados é um rico conjunto de in- formações, divergindo dos registros de logs do sistema. A coleta automática de dados em aplicações e sistemas multiagentes é fundamental para o entendimento de determinadas ações e resultados, dada a complexidade do domínio e das aplicações desenvolvidas. Essa relação entre proveniência em sistemas multiagentes e modelos existentes de proveniência é discutida no capítulo seguinte.

Este trabalho defende a importância do uso da proveniência de dados para tornar explicáveis SMA, discutindo quais questões podem ser respondidas em relação ao comportamento do SMA usando a proveniência e, visa como continuidade demonstrar os benefícios que a proveniência de dados fornecem para responder as questões supramencionadas. Este estudo envolve o uso de um framework, a saber, o FProvW3C (29), que estende a possibilidade de coletar e armazenar a proveniência dos dados, produzidos por um SMA.

A proveniência dos dados deve ser modelada e armazenada para análise posterior e estar disponível para diferentes usos, como ontologias para má- quinas de inferência, mineração de dados ou consultas específicas (30). Além disso, é importante determinar como os rastros de proveniência serão captura- dos, porque a falha em registrar uma atividade pode invalidar a replicação das etapas de uma execução do sistema, bem como a identificação da falha como um todo.

1.4 Organização do Trabalho

Este trabalho está organizado em três capítulos além desta introdução, conforme Figura 1.1. O capítulo 2 apresenta a fundamentação teórica do trabalho, abordando conceitos de sistemas multiagentes, sistemas de logs tradicionais, proveniência de dados e alguns trabalhos que rodeiam está proposta. No capítulo 3, apresenta-se uma visão geral do trabalho proposto, assim como alguns cenários de aplicação que podem demostrar sua utilidade. Já no capítulo 4, são apresentadas algumas considerações finais, as contribuições esperadas, bem como algumas limitações do trabalho.

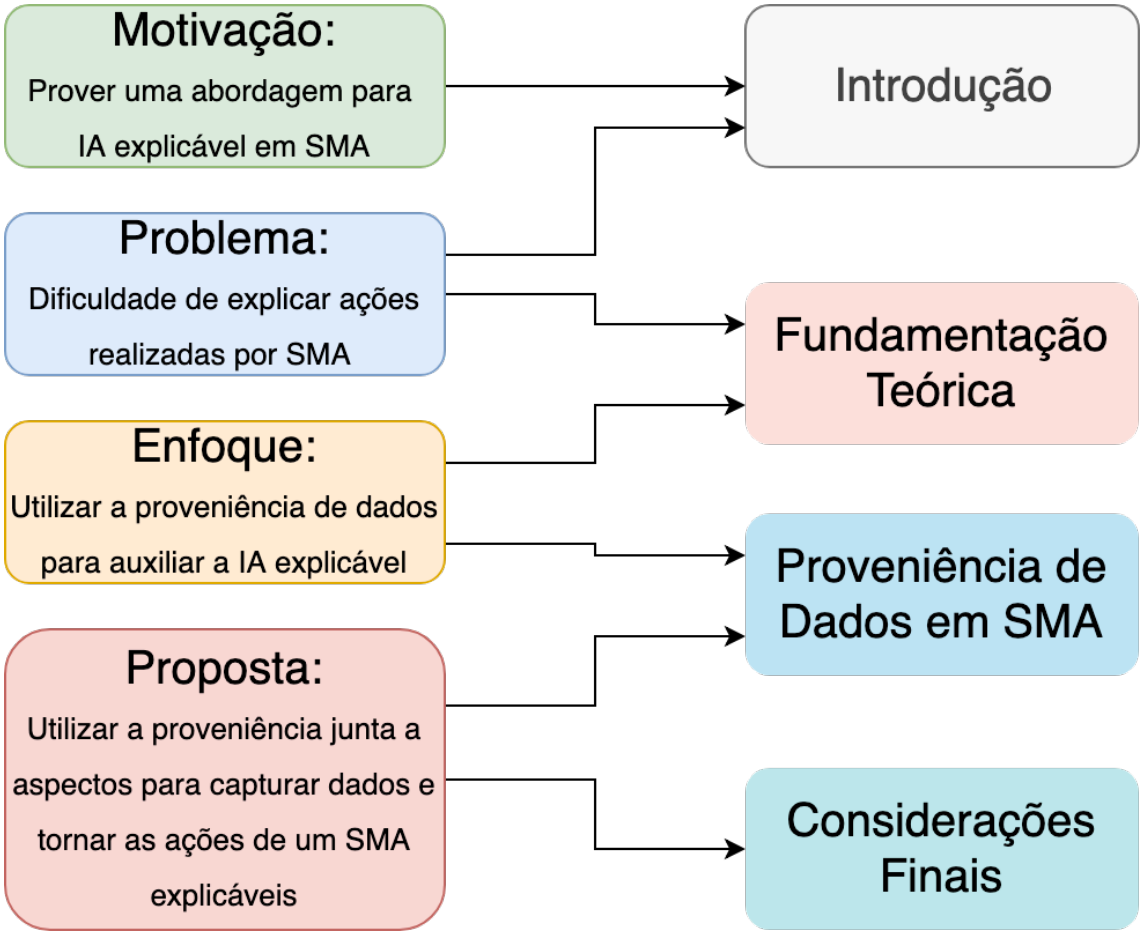


Figura 1.1: Organização do trabalho.

2 Fundamentação Teórica

A inteligência artificial não é um conceito novo em ciência da computação e conforme (33, 34) apud (22), experimentou muitos altos e baixos desde sua introdução formal como uma disciplina acadêmica. O grande objetivo da IA é fornecer os fundamentos necessários para desenvolvermos softwares que possa aprender autonomamente a partir da experiência anterior, de forma automática e sem intervenção humana, para realizar algum tipo de tarefa (35), um sistema multiagente (SMA) pode ser entendido como sinônimo de inteligência artificial distribuída.

Com isso, entender e tornar explicáveis as ações de um sistema autônomo é um desafio que ainda permanece em aberto (25, 24, 26). No geral, a eficácia total de todo o sucesso de um SMA é limitada pela sua capacidade do algoritmo de explicar seus resultados à humanos.

Um SMA é muito mais que uma simples inteligência artificial distribuída, e por isso é complexo entender certas ações que podem ocorrer em aplicações complexas. Na próxima seção teremos uma visão geral sobre SMA e por que hoje suas ações não são claras o suficiente para permitirem que a sua inteligência artificial seja auto explicável.

2.1 Sistemas Multiagentes

Conforme abordado por (13), um agente pode se comunicar com outros agentes ou com seres humanos, reagir a mudanças no ambiente e procurar atingir um objetivo es-

pecífico. Conforme já explicado, os agentes podem ser classificados como agentes de software e agentes de software inteligentes. Não é objetivo deste trabalho discutir o que é inteligência em um sistema multiagente.

Na maioria dos casos, os agentes inteligentes são desenvolvidos usando o modelo BDI (Belief-Desire-Intention) (12). No modelo BDI as ações são derivadas de um processo de raciocínio prático, que consiste em duas etapas. O primeiro passo faz a seleção de um conjunto de desejos que devem ser alcançados, de acordo com as crenças do agente. O segundo passo é responsável por determinar como esses desejos podem ser alcançados como resultado da etapa anterior. O BDI é composto de três atitudes mentais:

- Crenças - Elas representam a visão fundamental do agente em relação ao seu ambiente, outros agentes, interações com outros agentes e crenças sobre suas próprias crenças. O último pode levar a conflitos internos;
- Desejos - Eles representam o estado motivacional do agente, onde escolhe os desejos possíveis de acordo com algum critério, encorajando-o a realizar as tarefas para as quais foi projetado. Um desejo pode estar em conflito com as crenças do agente ou com outro desejo;
- Intenções - Intenções representam o plano de ação escolhido pelo agente, ou seja, se um agente decide seguir um objetivo específico, esse objetivo se torna uma intenção. As intenções são formadas a partir de um processo de deliberação, mas também podem conter as intenções iniciais inseridas pelo usuário.

Além do BDI, temos conforme explicado por (51), a arquitetura NBDI (52), que estende a arquitetura BDI (12) ao incluir funções relacionadas a normas para apoiar o raciocínio. Além disso, as normas são consideradas um conceito primário que influenciam a decisão do agente, quando raciocinando sobre as suas crenças, desejos e intenções.

Um agente de software deve ser conforme (36) : i) autônomo: agir sem a necessidade de intervenção humana; ii) reativo: perceber e responder a mudanças no ambiente; iii) pró-ativo: executar suas ações sempre que possível; iv) social: interagir com outros agentes, seja para cumprir com seus objetivos ou com os objetivos de outros agentes e; v) intencional: possuir crenças, desejos e objetivos e a intenção de cumpri-los.

Já um software que faz uso de um sistema multiagente pode apresentar uma arquitetura simples; composta por um único agente, o que vai contra os princípios de um sistema multiagente, onde à cooperação entre si permite resolver problemas além de sua capacidade individual, como supracitado; arquitetura moderada, formada por agentes homogêneos e distribuídos e; arquitetura complexa, composta por agentes heterogêneos, distribuídos e que apresentam algum grau de inteligência (14).

Em arquiteturas com mais de um agente, a comunicação entre os mesmos podem ocorrer conforme (14) como um quadro-negro, compartilhando um repositório de perguntas e respostas para a interação entre os agentes; comunicação por troca de mensagens, o que ocorre de maneira assíncrona, mas dependendo que o agente conheça seus vizinhos para se comunicar e a comunicação federativa; onde os agentes são divididos em grupos e a comunicação ocorre entre os grupos por meio de agentes facilitados ou controladores, de modo a reduzir o excesso de comunicação.

Para este último, as coordenações entre os agentes podem ocorrer como mestre-escravo ou mecanismo de mercado. Na mestre-escravo, os mestres são os agentes responsáveis por delegar as ações aos outros agentes e os escravos são os agentes respon-

sáveis por executa-las. Já no mecanismo de mercado todos os agentes estão no mesmo nível e conhecem as ações que cada um pode executar, desta forma os agentes negociam as execuções das tarefas entre si, diretamente.

Essas classificações de arquitetura buscam definir padrões entre as arquiteturas que são implementadas usando sistemas multiagentes. Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de sistemas, visto que existem padrões arquiteturais, padrões de análise, padrões de projeto, padrões de código, entre outros (37). Entretanto, se os usos de padrões não forem bem projetados podem criar lacunas no desenvolvimento do software, e essas lacunas dificultarem entender certas ações que estão ocorrendo em um software, por exemplo.

2.1.1 Plataformas de Agentes

Existe uma grande quantidade de plataformas de agentes disponíveis (38), com diferentes características e com falta de critérios padronizados concretos, torna-se a escolha da plataforma de agentes dependente da finalidade do sistema a ser desenvolvido. Portanto, escolher a plataforma certa ou adequada para um determinado problema ainda é uma dúvida para os desenvolvedores. Nesse contexto vamos apresentar algumas das principais plataformas de sistemas multiagentes e suas características.

A JADE[1] (39) é um framework totalmente implementado em Java e segue as especificações FIPA[2] (40). A plataforma do agente pode ser distribuída em máquinas heterogêneas e a configuração pode ser controlada através de uma interface remota. A configuração pode ser alterada em tempo de execução, movendo agentes de uma máquina para outra, conforme necessário. É um software livre, de código aberto e estável, distribuído pela Telecom Itália, detentora dos direitos autorais.

O sistema do JADEX[3] BDI (41), segue o modelo BDI e permite a programação de agentes de software inteligentes em XML e Java. O projeto é conduzido pelo Grupo de Sistemas Distribuídos e Sistemas de Informação da Universidade de Hamburgo e licenciado sobre a GNU LGPL. Sua arquitetura baseia-se na noção de componentes ativos, que são conceitualmente baseados em SCA (arquitetura de componentes de serviço). Isso permite a decomposição hierárquica de componentes que interagem através de serviços, e se destina a funcionar em sistemas distribuídos, concorrentes e dinâmicos. O BDI é uma abstração que permite aos desenvolvedores gerenciar a complexidade do problema, como supracitado.

JACK[4] (42) é um ambiente multiplataforma, para construir, executar e integrar sistemas multiagentes. É construído sobre a lógica BDI e está inteiramente escrito em Java, mas ainda não suporta as especificações do FIPA. Já o Jason[5] (43) é desenvolvido em JAVA, sendo um interpretador do AgentSpeak, uma linguagem de programação lógica baseada em agente do BDI. Um dos pontos forte são os recursos customizáveis pelo usuário.

O BDI4JADE[6] (44) é uma extensão da plataforma JADE, construída em JAVA e que implementa a arquitetura BDI de agentes inteligentes. Como no JADE, cada agente tem

1 Disponível em: http://jade.tilab.com/

2 Disponível em: http://www.fipa.org/

3 Disponível em: https://www.activecomponents.org/

4 Disponível em: http://aosgrp.com/products/jack/

5 Disponível em: http://jason.sourceforge.net/

6 Disponível em: http://www.inf.ufrgs.br/prosoft/bdi4jade/

sua própria execução, composta de um conjunto de intenções e capacidades, as crenças e planos são partes da capacidade de um agente. As mensagens são enviadas e recebidas como no JADE e os eventos podem ser relacionados às crenças ou objetivos. O BDI4JADE possui seu código fonte e documentação disponíveis online, com exemplos de uso e que está em constante evolução. A plataforma segue as especificações FIPA.

Como apresentado, existem diversas plataforma para diferentes objetivos e implementadas de diferentes maneiras. Contudo, falta padrões claros e abrangentes para a especificação, construção e evolução de softwares com uso de sistemas multiagentes. Os SMA precisam ser bem projetados por causa de sua ampla quantidade de características, para se moldar a vários tipos de ambientes para atuação. Como já apresentado, os softwares devem refletir o mundo real, demandando que mudanças futuras não causem impacto na arquitetura do sistema. Para isso é necessário que o sistema multiagente tenha um foco de atuação bem definido para que os agentes possam ter objetivos concisos.

A seguir serão discutidos estes pontos e como a captura de informações do sistema multiagente podem auxiliar na compreensão das ações dos agentes, na identificação de falhas, no acompanhamento da evolução e como essas informações devem ser capturadas sem afetar a arquitetura do sistema multiagente dentro da arquitetura do software.

2.2 Logs

Uma das primeiras fases do desenvolvimento de um software independente do processo adotado é a sua modelagem, onde a especificação levantada será estruturada para representar de forma abstrata as atividades, os papéis de cada individuo envolvido e os artefatos que devem compor o software. Nesse instante o mais comum é utilizar uma linguagem padrão para a comunicação entre os membros da equipe de desenvolvimento.

Uma das linguagens para elaboração da modelagem do projeto de software bastante difundida é a UML (46). A UML[7] possui diagramas que podem ser divididos em dois grandes grupos: i) diagramas estruturais e ii) diagramas comportamentais, que buscam representar todas as interações e requisitos para construção de um software. Entretanto, como abordado por (47), a UML às vezes é insuficiente para modelar agentes e sistemas baseados em agentes, pois nenhum formalismo ainda é suficiente para especificar o desenvolvimento de softwares baseados em agentes, difundindo trabalhos como os de (47, 45, 61).

Para empregar uma programação baseada em agentes, uma técnica de especificação deve suportar todo o processo de engenharia de software, iniciando no planejamento, seguindo para a análise e design e, finalmente, culminando na construção e transição do software, sem esquecer das manutenções corretivas, adaptativas e evolutivas que podem surgir posteriormente.

Outro ponto importante no desenvolvimento de um software é a atividade de teste, onde conforme (19), é na atividade de teste que consiste em uma análise dinâmica do software e é uma atividade relevante para a identificação e eliminação de erros que persistem. Já em cenários com uso de agentes inteligentes, deve-se ter um conjunto de preocupações maiores, que abordem as ações executadas, as crenças, desejos e intenções em cada momento e sua autonomia, conforme (29).

---

7 Disponível em: http://www.uml.org/

De modo geral, para avaliar um sistema é necessário reunir informações suficientes de modo que as pessoas possam entender as execuções, abstraindo os detalhes irrelevantes e compreender às interações que ocorreram. Com base nas informações provenientes do sistema em tempo de execução, podemos identificar falhas não observadas nos testes, possibilitando melhorar a arquitetura, prolongar o seu decaimento e tornar as ações do sistema explicáveis.

Os registros de log constituem uma fonte básica de informação, seja para a detecção de problemas, uso indevido do sistema, problemas com serviços ou hardwares, ou mesmo para rastrear (auditar) as ações executadas por um usuário ou um sistema multiagente.

Entretanto, alguns pontos sobre o uso de logs ainda estão em abertos, como por exemplo o que deve ser registrado, a frequência desse registro, o que deve conter e etc. Os logs se caracterizam por serem flexíveis, onde mediante configurações é possível registrar desde eventos críticos até praticamente todos os eventos de um sistema, mas em muitos casos eles não são modelados no projeto do sistema e cabe ao desenvolvedor determinar sobre o uso dos mesmos, pois em algum ponto do projeto por exemplo, foi especificado um requisito não funcional que o sistema deveria registrar informações da execução simplesmente.

Como já dito, registrar informações de um software em sistemas complexos pode trazer benefícios e uma das formas mais comuns é o 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 registram quem acessou os recursos computacionais, aplicativos, arquivos de dados e utilitários, quando foi feito o acesso e que tipo de operações foram efetuadas, desde que programados para registrar isso.

Uma ferramenta bastante conhecida para geração de logs é a Log4J[8] (21). Ela divide os logs em 5 níveis hierárquicos, sendo eles: i) DEBUG; ii) INFO; iii) WARN; iv) ERROR e; v) FATAL. A grande maioria dos sistemas de logs seguem essa mesma característica, entretanto o gerenciamento de eventos envolve um conjunto mais amplo 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 (20).

Conforme (20), 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 (20), 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 do sistema para entende-lo e os sistemas de log, normalmente trazem apenas palavras-chaves em seus registros, ou explicações superficiais.

Como abordado por (30), 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 para a melhoria do processo, e uma maneira de analisar esses dados é usar técnicas e modelos de proveniência.

---

8 Disponível em: https://logging.apache.org/log4j/

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 (29). Atualmente, existem dois padrões principais para a captura de dados de proveniência: i) o modelo OPM (49), e ii) o modelo PROV (50), que serão abordados na sequência.

2.3 Proveniência de Dados

Embora (18) discuta a modelagem de dados para sistemas autônomos, não trata de modelos clássicos de proveniência, como OPM (49) ou PROV (50), deixando que cada sistema possa implementar a captura de dados dos indivíduos que utilizam o sistema desenvolvido e do sistema multiagente da maneira que melhor se adequem. A falta de um modelo padrão para sistemas multiagentes causa outros problemas, como a integridade, interoperabilidade, representatividade e confiabilidade dos dados.

O OPM (49) foi o primeiro modelo de proveniência amplamente utilizado e emergiu de uma conferência específica sobre o assunto, o Workshop Inter- nacional de Proveniência e Anotações (IPAW). O modelo OPM representa o uso de dados digitais e está relacionado à documentação, derivação e anotação do mesmo. Mais tarde, um outro modelo de proveniência foi desenvolvido pelo grupo de proveniência do W3C e foi denominado PROV (50). PROV não é uma expansão do OPM. É um novo modelo desenvolvido pelo W3C que está ganhando força para se tornar o modelo padrão em sistemas de captura de proveniência. O objetivo desta seção é fazer uma comparação entre esses dois modelos.

O modelo OPM é baseado em três vértices e suas relações causais são representadas como arcos de um gráfico. Os vértices principais do OPM são:

- Artefato: entidades imutáveis que podem representar um objeto físico ou uma representação digital de um sistema informático;
- Processo: uma ação ou conjunto de ações executadas ou causadas por artefatos que resultarão em novos artefatos;
- Agente: uma entidade contextual que atua como um catalisador para um processo, facilitando, controlando ou afetando sua execução.

As relações causais do OPM indicam relações de dependência e/ou causalidade. No total, existem cinco relações que classificam: (i) uso, (ii) geração, (iii) controle, (iv) desencadeamento e (v) derivação. Todos os vértices e relações causais nos permitem trabalhar com registros temporais e são específicos para cada propósito (32). Cada uma das relações está especificada abaixo:

- used: Indica a relação de dependência de um artefato na execução de um processo;
- wasGeneratedBy: Indica que um processo foi responsável pela geração de um artefato;
- wasControlledBy: Indica que um processo foi controlado por um ou mais agentes;
- wasTriggeredBy: Indica que um processo foi iniciado por outro processo;
- wasDerivedFrom: Indica que um artefato se originou de outro artefato.

O modelo PROV consiste em uma família de doze documentos propostos pelo W3C. Esses documentos são divididos em modelo de dados, restrições, semântica, ontologia e anotações, conforme explicado por (50). O PROV procura ser amplo, abor-

dando a proveniência sob diferentes perspectivas e propósitos. Existem diferenças nos vértices em relação à OPM e novas relações causais. A proveniência pode ter três pontos principais: (i) nos agentes; (ii) em processos e, (iii) em objetos. Os vértices principais da PROV são:

- Entidade: as entidades que representam algo físico, digital, conceitual ou qualquer coisa com alguns aspectos fixos e, as entidades podem ser reais ou imaginárias;
- Atividade: promove atividades relacionadas a entidades e tem um período específico de tempo;
- Agente: Tem alguma responsabilidade por uma atividade, por uma entidade ou por atividades de outro agente. Um agente pode representar uma pessoa, uma organização ou um agente de software.

O modelo PROV possui dez relacionamentos que aumentam a representa- tividade da informação proveniente frente ao modelo OPM. As relações causais no modelo PROV são:

- used: representa um relacionamento onde uma atividade usou uma entidade;
- wasGeneratedBy: Representa uma entidade que foi gerada por uma atividade;
- wasAssociatedWith: Representa uma associação entre um agente e uma atividade;
- wasAttributedTo: Representa uma entidade que foi atribuída a um agente;
- actedOnBehalfOf: representa uma responsabilidade ou hierarquia entre os agentes;
- wasRevisionOf: Indica uma derivação, onde a entidade derivada é uma revisão de outra entidade existente, como uma correção;
- wasDerivedFrom: Indica uma derivação entre entidades com um caráter evolutivo ou adaptativo;
- wasInformedBy: Representa uma relação entre atividades, onde a ativi- dade A informa que ela usou uma entidade gerada pela atividade B;
- wasStartedBy: Representa uma relação entre atividade e entidade, registrando o início da ação da forma temporal;
- wasEndedBy: Representa uma relação entre atividade e entidade, registrando o termo da ação da forma temporal.

A vantagem de usar o PROV em comparação com o OPM é a existência de relações específicas para agentes, o modelo PROV foca na responsabilidade dos dados e questões históricas, relações de agentes e entre os outros vértices. A Tabela 2.1 resume a comparação entre os modelos PROV e OPM.

Tabela 2.1: Comparação entre o OPM e o PROV

OPM	PROV
Artifact	Entity
Process	Activity

Agent	Agent
used	used
wasGeneratedBy	wasGeneratedBy
wasControlledBy	wasAssociatedWith
wasTriggeredBy	wasStartedBy
wasDerivedFrom	wasDerivedFrom
-	wasEndedBy
-	wasRevisionOf
-	wasAttributedTo
-	wasInformedBy
-	actedOnBehalfOf

O modelo PROV é mais extenso que o OPM, além dessas relações apresentadas, existem outras relações causais, que podem ser divididas em dois subconjuntos: i) relações primárias e ii) secundárias (opcionais). A Figura 2.1 mostra as relações primárias em negrito e a figura 2.2 apresenta as (opcional) relações secundárias.

		Object		
		Entity	Activity	Agent
Subject	Entity	**WasDerivedFrom** Revision Quotation PrimarySource AlternateOf SpecializationOf HadMember	**WasGeneratedBy** WasInvalidatedBy *R T L*	**WasAttributedTo**
	Activity	**Used** WasStartedBy WasEndedBy *R T L*	**WasInformedBy**	**WasAssociatedWith** *R*
	Agent	—	—	**ActedOnBehalfOf**

Figura 2.1: Relações Primárias do PROV

		Secondary Object		
		Entity	Activity	Agent
Subject	Entity	—	WasDerivedFrom (activity)	—
	Activity	WasAssociatedWith (plan)	WasStartedBy (starter) WasEndedBy (ender)	—
	Agent	—	ActedOnBehalfOf (activity)	—

Figura 2.2: Relações Secundárias do PROV

Para o uso em sistemas multiagentes o modelo PROV W3C leva vanta- gens, visto que modelo PROV tem por objetivo armazenar dados de proce- dência de forma detalhada, focalizando as responsabilidades dos agentes em cada item de proveniência. O modelo PROV (50) consiste em 12 documentos que definem sua especificação. Entre os

principais documentos estão o PROV- DM, que especifica o modelo de captura de dados; O PROV-CONSTRAINTS, que é o conjunto de restrições aplicáveis ao modelo de dados (PROV-DM) e o PROV-O, uma ontologia para o mapeamento do modelo de dados, como explicado por (53).

2.4 Trabalhos Relacionados

Esta seção tem como objetivo apresentar os principais trabalhos que servem de base e inspiram esta proposta. Tais trabalhos envolvem proveniência de dados, modelagem, debugger ou explicação de ações em sistemas autônomos.

Começando pelo trabalho de (15), este pode ser considerado um trabalho chave na proposta, onde os autores abordam a modelagem de dados em sistemas autônomos e, justificam sua necessidade como uma forma de documentar os acontecimentos nas aplicações. O trabalho apresentou as questões levantadas na seção 1.1 desta proposta e um modelo para descrever as ações de sistemas autônomos. Entretanto, esse trabalho é de 2007 e, por isso, não baseia-se em modelos de proveniência de dados atualmente amplamente di- fundidos e utilizados como o OPM ou o Prov, limitando a solução ao caso apresenta no trabalho.

Já o trabalho de (62) baseia-se na modelagem AUML e, com base na modelagem e execução dos agentes, utiliza redes de petri para verificar o comportamento dos agentes dentro da aplicação, apesar de ser uma proposta interessante para 2002, todas as ações são explicadas por meio de análise estáticas, que verificam apenas se o agente está cumprindo o que lhe foi designado. Vale ressaltar que está abordagem não explica o comportamento do sistema, não permite reprodução ou análise dinâmica. Essas características foram melhor exploradas no trabalho de (63), onde baseado na coleta de informações do sistema autônomo, permite realizar um debugger onisciente, recriando os passos executados, uma vez que as ações são registradas. Entretanto, o trabalho de (63), não utiliza nenhum formalismo ao se armazenar as informações coletadas, além de não mostrar de forma explicita como os dados são coletado e se geram algum impacto na arquitetura da aplicação com uso de agentes de software. O trabalho de (63) segue o trabalho de (64), mantendo as mesmas lacunas.

Outro trabalho interessante a está proposta é o de (65), onde aborda- se o modelo 5W2H, apesar de levantar pontos interessantes e características diferentes a desta proposta, serve de inspiração e como forma de apoio as questões levantadas.

Por fim, o trabalho de (25) aborda a construção de sistema autônomos explicável, ou seja, que documenta suas ações e permite compreender o raciocínio dos agentes. Assim como nos trabalhos supracitados, o mesmo não aborda o uso de modelos de proveniência como forma de armazenar e permitir a reprodutibilidade, rastreamento ou tornar explicáveis as ações. Apesar de nenhum formalismo na coleta e armazenamento dos dados para análise, utiliza como proposta uma base centralizada para os dados coletados, o que em sistemas distribuídos como o caso dos SMA, torna a análise mais fácil do ponto de vista do usuário.

Esses trabalhos apresentam apenas um esboço da base dessa proposta, e como a solução aqui proposta baseiam-se em abordagens já consistentes, melhorando suas lacunas e abordando uma nova visão a um problema que já é discutido a tempos.

3 Proveniência de Dados em Sistemas Multiagentes

A proveniência dos dados nos permite acompanhar as ações do sistema autônomo e sua interação com outros sistemas e humanos. Para isso, um mecanismo de captura de

proveniência precisa acessar os detalhes relevantes de uma tarefa computacional, registrando seus passos, informações de execução e anotações de usuários. Isso envolve os dados de entrada dos eventos, a sequência de interações que ocorreram e os resultados gerados. Uma visão geral do modelo PROV pode ser visto na figura 3.1.

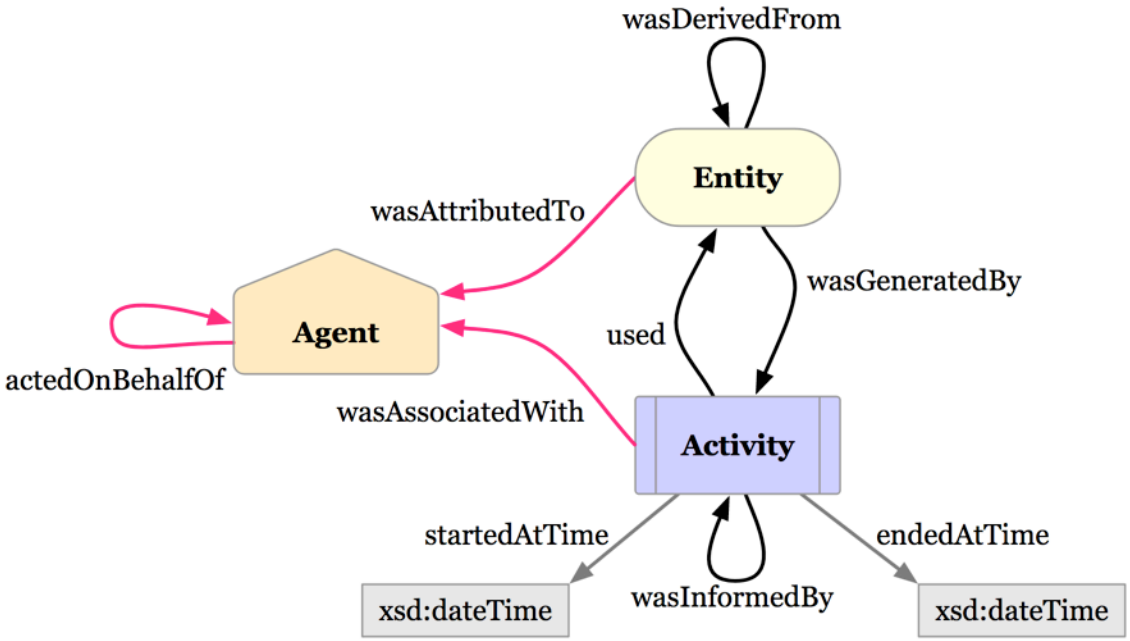


Figura 3.1: Modelo PROV W3C

O uso de log perde conhecimento por não representar a ligação entre os eventos, pois o log em muitos casos são limitados a linhas inseridas no meio do código que registram um determinado acontecimento ou passagem por um ponto. Um exemplo disso pode ser visto na Listagem 3.1, do código extraído da classe BDIAgent.java do BDI4JADE (44). Nesse exemplo temos o uso do trace para registrar qual trecho de código 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.

```
@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."); }
```

Listing 3.1: Uso de logs no BDI4JADE

Uma possibilidade para termos um melhor registro dos eventos é com o uso de um modelo de proveniência como o PROV, visto que o mesmo já é projetado para registrar informações desse tipo e possibilita a análise por ontologias e mineração.

Entretanto, uma questão que persiste é como capturar esse dados de modo que a arquitetura do software não seja afetada e o sistema não fique poluído com o uso excessivo de código inserido para capturar todas essas informações.

Em um software que faz uso de um sistema multiagente, uma solução viável pode ser o uso de uma abordagem usando aspecto, visto que por meio de aspecto podemos definir as funções a serem monitoradas e, com isso, independente do software, as informações são colhidas da plataforma do agente, dado que sua arquitetura é estáveis.

Um framework para capturar a proveniência dos dados, denominado FProvW3C (53), foi proposta e instanciado em um sistema que utiliza agentes para classificação de gases expelidos por humanos. A experiência obtida com o desenvolvimento deste sistema levou à especificação do método 5W2H, para determinar os dados que devem ser registrados para modelar a proveniência dos dados. Este método suporta a identificação dos dados, processos e ações mais importantes dentro de um projeto, unidade de produção ou sistema a ser gravado. O 5W2H é um método composto por sete questões que, além de identificar os dados, processos e ações, também permite identificar quem é responsável por uma ação, o que o responsável pela ação fez e por que o fez. As sete questões da técnica 5W2H adaptadas para sistemas multiagentes são:

- O que o agente fez?
- Quem (ou qual agente) fez isso?
- Onde aconteceu (ação ou falha)?
- Quando a ação ou falha ocorreu?
- Por que a ação ou falha ocorreu?
- Como a ação ou falha ocorreu?
- Quanto isso custou ao sistema?

Embora simples, a técnica 5W2H (54) apoia a análise e a aquisição de conhecimento sobre um determinado processo, problema ou ação que ainda não foi realizado ou que já ocorreu. Quando aplicada à SMA, ela pode auxiliar de três formas diferentes, conforme discutido a seguir:

– Diagnóstico: A técnica 5W2H permite a investigação de um problema ou processo, a detecção de falhas e pontos de vulnerabilidade. Para lidar com esses problemas, é necessário aumentar a quantidade de informações do sistema e, nesse caso, o framework do FProvW3C pode ser usada.

- Plano de ação: O procedimento de teste de software em SMA não é uma tarefa trivial (51), porque o agente deve ter um comportamento autônomo mesmo seguindo normas. A proveniência dos dados auxilia na construção de uma base de informações dos agentes, onde é possível por exemplo: i) verificar seu plano de ação (suas decisões) sobre o que deve ser feito e ii) eliminar comportamentos irregulares que possam causar alguma anomalia no sistema.

– Documentação: O uso de proveniência possibilita documentar as etapas do sistema, extrair relatórios de execuções, falhas, pontos de vulnerabi- lidade e resultados produzidos por uma determinada entrada ou saída.

O método 5W2H permiti dividir os dados em diferentes partes, que por sua vez mostravam quais pessoas, ou agentes de software, estavam operando em cada situação e cada ação. Mostra também em que momento certas atividades estavam acontecendo, qual a sequência de ações que produziu o resultado, como a ação foi executada e quais recursos foram utilizados.

Alinhando as questões apresentadas pelo método 5W2H com o uso de proveniência de dados em SMA, como resultado, é possível capturar de forma dinâmica o comportamento do sistema e responder às questões levantadas na Seção 1.1. Essa coleta e armazenamento de dados usando o método 5W2H pode ser feito com o framework FProvW3C (53), que se baseia na programação orientada a aspectos para interceptar código de uma plataforma multiagente e registrar dados para a proveniência. Mais detalhes na seção 3.1.

3.1 Framework FProvW3C

O framework FProvW3C trabalha com uma abordagem ansiosa (55), para que os dados sejam coletados a todo momento e possam ser consultados a seguir. Atualmente, em sua arquitetura (figura 3.2), o framework FProvW3C apresenta toda a especificação do PROV-DM com as anotações em Java Persistence API (JPA).

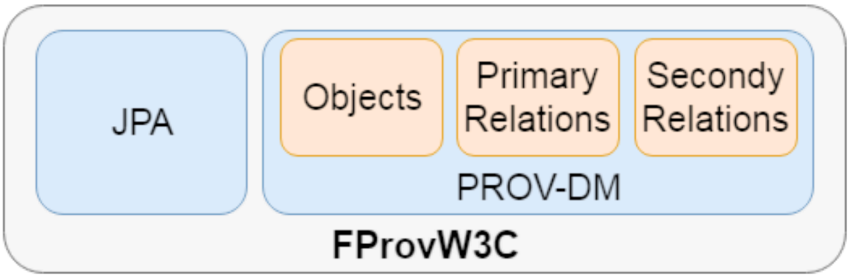


Figura 3.2: Arquitetura do FProvW3C

Além da modelagem, o framework FProvW3C traz anotações de persistência. Isso reduz o trabalho dos usuários e evita erros de mapeamento por aqueles que não têm extenso conhecimento do PROV W3C. Por ser um framework, torna possível integrá-lo

em diferentes sistemas. Os frozen-spots do framework são os vértices principais e os hot-spots são as relações causais, ambas definidas pelo modelo PROV, e se adaptam a diferentes contextos de aplicação.

O FProvW3C é uma estrutura de mapeamento de objetos relacionais, encarregada de criar o banco de dados, as tabelas e seus respectivos atributos no SGBD. O modelo PROV determina as classes e como elas se relacionam, além dos atributos básicos. Desta forma, o framework fornece as classes com os atributos básicos (frozen-spots) e, além disso, permite a criação de hot-spots, estendendo os atributos de cada classe e os relacionamentos. Estes atributos destinam-se a representarem as características do sistema no qual o FProvW3C será aplicado.

Como o framework executa mapeamento de dados de acordo o PROV, seu uso é facilitado para o usuário, onde é possível aplicar a proveniência dos dados em suas aplicações. O FProvW3C cria uma camada intermediária entre o aplicação e banco de dados. Esta camada permite tratar e mapear os dados a serem armazenados, vinculando a fonte de informações a eles. Portanto, todos os dados manipulados ou ações realizadas pelo sistema multiagente podem ser catalogadas.

3.2 Aplicabilidade da Proposta

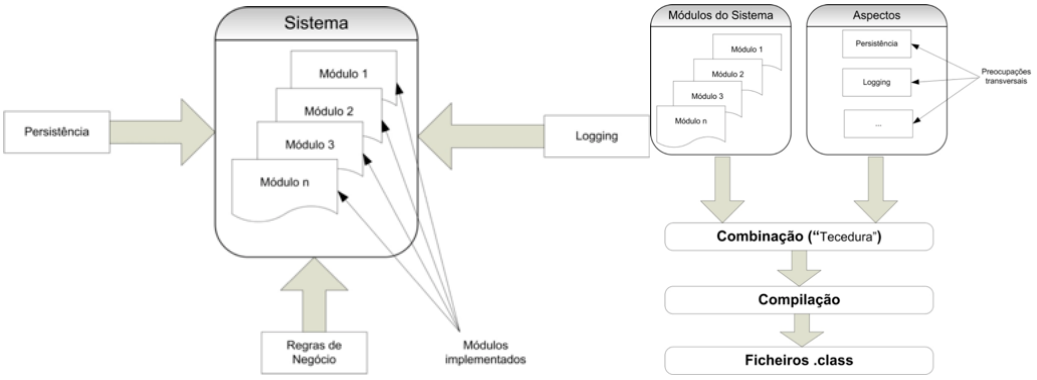
Atualmente um paradigma muito utilizado na construção de software é o de orientação a objeto (OO), que possibilita que na programação seja utilizada herança, polimorfismo e interfaces. Entretanto, uma característica importante a ser observada é com relação a sintaxe e a semântica.

A sintaxe é o formato utilizado pelas linguagens de programação para definir o funcionamento do código, ou seja, a estrutura que o código deve possuir. Já a semântica indica como o software deve se comportar quando executadas num determinado ambiente. Esse conhecimento é importante pois quando os códigos do software são compilados, temos a garantia que apenas a sintaxe está correta. Isso nos leva a muitas vezes para inspecionar o código por meio do uso de depuradores, pontos de interrupção e uso de logs.

Como já apresentado, o uso de logs pode não ser uma boa opção para captura de informações em sistemas complexo, como é o caso de software que usam alguma inteligência artificial distribuída. O uso de uma abordagem usando aspecto pode agregar recursos em uma programação OO, como abordado por (56).

Uma abordagem comum para o desenvolvimento de softwares complexos é a modularização, onde conforme (57), torna o software mais flexível e pelos módulos serem de certo modo independentes, facilita na compreensão do sistema. Contudo, registrar as ações nesses sistemas podem ser um desafio, visto que podem ser distribuídos, apresentar espalhamento de código ou emaranhamento entre os módulos. Outro ponto que torna a gestão dos logs um desafio maior é a integração entre eles, que também deveriam compor os registros, dado que o software é o mesmo, como representado na Figura 3.3(a), conforme (58).

Nesses casos, a proveniência pode ser implementada usando aspectos, onde não será necessário modificar a arquitetura do sistema multiagente para a inserção da captura dos traços de procedência. Dessa forma a proveniência é tratada pelo aspecto como uma preocupação transversal, adicionada ao código do software no processo de compilação, conforme a Figura 3.3(b).



3.3(a): Interceptação 3.3(b): Integração

Figura 3.3: Interceptação dos dados e Integração da captura ao SMA

Com isso, o código da plataforma de agentes e do software 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 ser feita para os agentes, para o sistema multiagente , conforme a Figura 3.4 e também dentro do BDI, conforme a Figura 3.5.

A diferença da abordagem proposta nesse trabalho para a que foi utilizada por (29), é o acesso ao dados da plataforma do agente e não ser necessário a inserção de código para a captura da proveniência junto ao código original do software.

Dispensar do programar a função de inserir a captura da proveniência em meio ao seu código, reduz seu trabalho, evita a captura dos dados de maneira equivocada, restringe que os detalhes relevantes estejam de fato sendo capturados e corretamente armazenados.

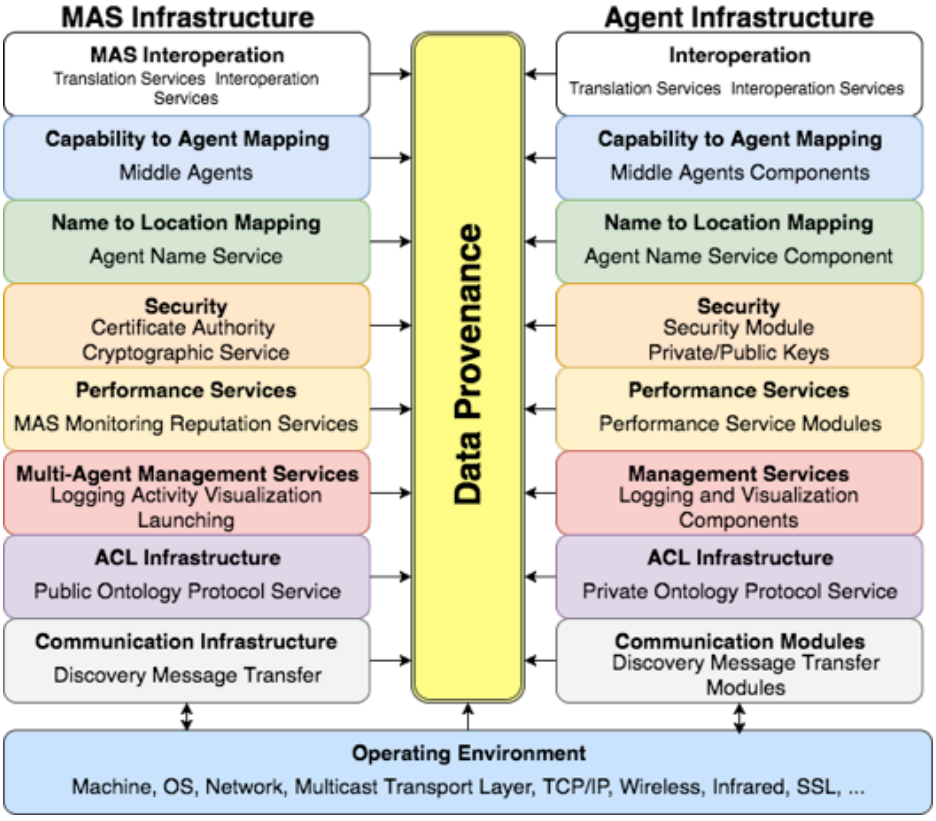


Figura 3.4: Captura dos dados no SMA

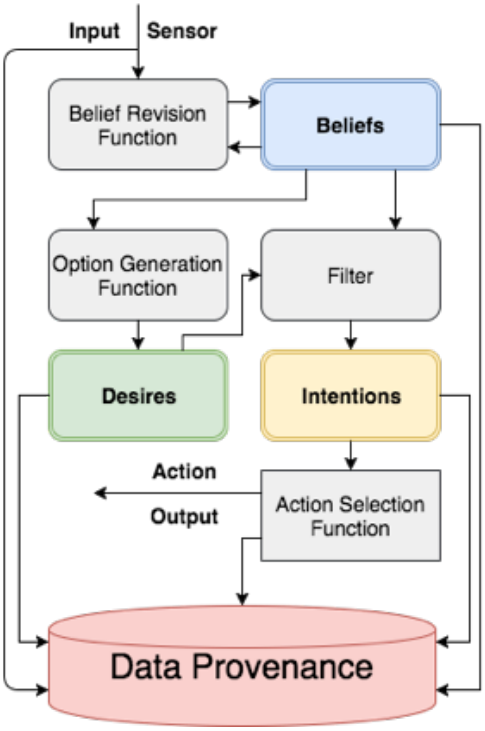


Figura 3.5: Captura dos dados no BDI

Em exemplo da captura dos dados de proveniência no BDI4JADE por meio de aspecto usando o FProvW3C pode ser visto na Listagem 3.2.

```
@Aspect
public class PROVBDIAgent {
@Around("execution(* br.pucrio.inf.les.bdi4jade.core. BDIA-
gent.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);
}
}
```

Listing 3.2: Capturando proveniência por meio de aspecto no BDI4JADE

Nesse exemplo da Listagem 3.2, está descrita a captura da atividade action apresentada na Listagem 3.1, onde são registrados os dados de inicio e fim da atividade e uma descrição à mesma. Além desses dados, para o exemplo apresentado na Listagem 3.1, deveríamos capturar as chamadas que manipulam as crenças e os objetivos do agente e de seu processo deliberativo, referenciando a atividade que ocorreu, o agente que a executou e a entidade que foi impactada pela ação.

Essa abordagem é o ponto de início para a discussão sobre a captura e o uso de proveniência de dados em sistemas que utilizam de inteligência artificial distribuída. Entender o comportamento e tornar explicável essa 'inteligência', nos permite ir mais longe, criando software complexos mais estáveis, seguros e de qualidade.

Apesar de aspectos ser uma solução viável para essa abordagem, alguns pontos devem ser considerados sobre o seu uso e serão apresentados na seção 4.2.

3.3 Cenários de Aplicação

Para descrever o primeiro cenário, podemos apresentar o caso demostrado em (29), onde o framework FProvW3C foi empregado para criar uma camada intermediária entre o aplicativo e o banco de dados. Esta camada permite tratar e mapear os dados a serem armazenados, vinculando a fonte de informações a eles, assim como os passos executados pelos agentes. Com isso, o FProvW3C cria uma base de conhecimento baseada no uso do aplicativo, tornando-o autoexplicativo. A figura 3.6 mostra que toda vez que um sensor de gás captura a variação de um gás, a persistência dos dados no banco de dados é invocada, e via framework FProvW3C, são registradas as proveniências dos dados. Desta forma, toda a captura e manipulação de dados são registradas, como uma filmagem, formando a base histórica da aplicação.

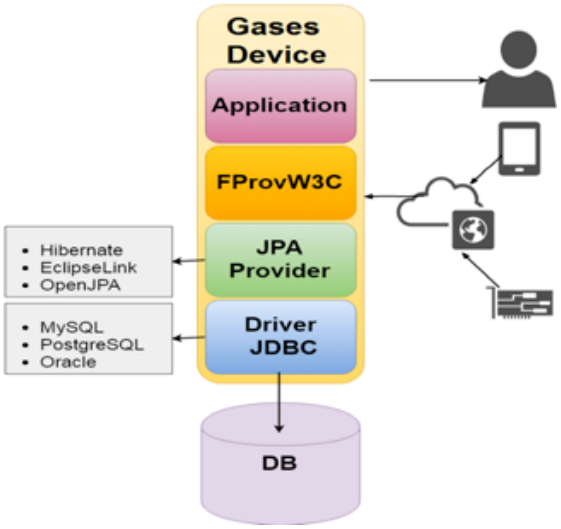
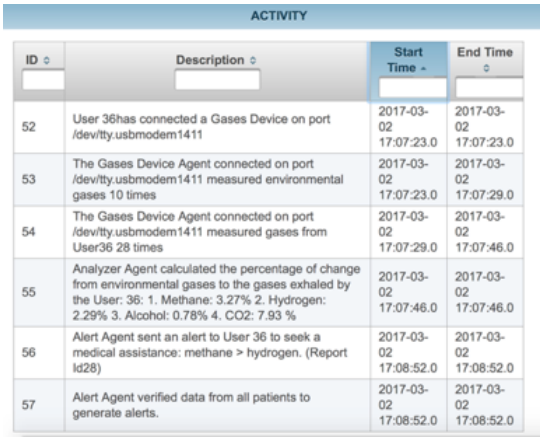


Figura 3.6: Arquitetura do Smell App com FProvW3C

Os dados registrados deste aplicativo incluem informações de usuários do sistema, sensores, gases monitorados e agentes de software (66). Os dados registrados foram vinculados às ações dos usuários e agentes de software, auxiliando na rastreabilidade de cada ação executada. Por exemplo, a figura 3.7(a), o sistema atribuiu o ID 36 a um usuário registrado. Então, usando a estrutura do FProvW3C, foi possível rastrear todos os dados gerados durante esta operação de registro, conforme figura 3.7(b). Como os dados vêm de várias fontes, é difícil identificar a origem de um problema. No entanto, como mostrado na figura 3.7(b), a proveniência facilitou a avaliação deste sistema multiagente, permitindo-nos acompanhar todas as atividades que foram realizadas durante a sua execução.

Outro cenário que pode demostrar a importância da proveniência de dados e, permitir compreender as ações de sistemas autônomos é o apresentado em (60), onde a proveniência de dados poderia ser utilizada para explicar ações que forem tomadas neste cenário e em acontecimentos como os de (28). No cenário de (60), temos um exemplo extraído do Código Brasileiro de Trânsito (CBT), onde em um cruzamento sem semáforo, um conjunto de carros autônomos deve definir de quem é a prioridade. Como os agentes de software são construídos sobre normas, devem respeitar o CBT e neste caso, existe um conflito entre os artigos 29 e 38 do CBT, deixando a resolução deste conflito a cargo dos motoristas.

3.7(a): Interface Smell App



ACTIVITY

ID	Description	Start Time	End Time
52	User 36has connected a Gases Device on port /dev/tty.usbmodem1411	2017-03-02 17:07:23.0	2017-03-02 17:07:23.0
53	The Gases Device Agent connected on port /dev/tty.usbmodem1411 measured environmental gases 10 times	2017-03-02 17:07:23.0	2017-03-02 17:07:29.0
54	The Gases Device Agent connected on port /dev/tty.usbmodem1411 measured gases from User36 28 times	2017-03-02 17:07:29.0	2017-03-02 17:07:46.0
55	Analyzer Agent calculated the percentage of change from environmental gases to the gases exhaled by the User: 36: 1. Methane: 3.27% 2. Hydrogen: 2.29% 3. Alcohol: 0.78% 4. CO2: 7.93 %	2017-03-02 17:07:46.0	2017-03-02 17:07:46.0
56	Alert Agent sent an alert to User 36 to seek a medical assistance: methane > hydrogen. (Report Id28)	2017-03-02 17:08:52.0	2017-03-02 17:08:52.0
57	Alert Agent verified data from all patients to generate alerts.	2017-03-02 17:08:52.0	2017-03-02 17:08:52.0

3.7(b): Proveniência no Smell App

Figura 3.7: Interceptação dos dados e Integração da captura ao SMA

O conflito aqui é determinar quem tem a prioridade. Neste caso, o uso da proveniência poderia explicar as interseções que ocorreram entre cada agente e em caso de um acidente, permitir rastrear as ações de cada agente de software para determinar as causas do problema. Este cenário, pode ser visto na figura 3.8.

Além destes cenário, outros como os apresentados em (67, 68, 69, 70, 71), podem se beneficiar com a proveniência de dados como mecanismo de rastrear e explicar as ações executadas nesses sistemas autônomos.

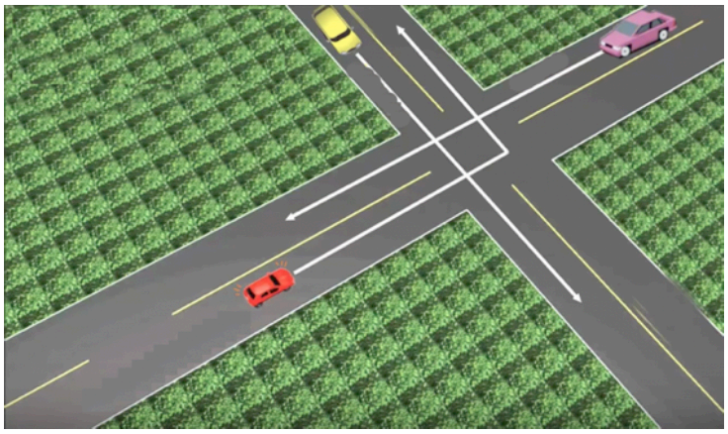


Figura 3.8: Cenário de interseção de trânsito no Brasil

4 Considerações Finais

Neste trabalho foram abordadas as arquiteturas de agentes e sistemas multiagentes existentes, sendo elas implementações de sistemas autônomos, bem como algumas das plataformas mais conhecidas de agentes. Discutiu-se sobre o desenvolvimento e sobre os softwares que fazem uso de inteligência artificial distribuída e como podemos compreender melhor a série de ações executadas por um sistema autônomo, e sua importância de serem explicáveis. Neste ponto foi abordado o registro histórico do sistema com de logs e por meio de proveniência, abordando as vantagens e limitações de cada um.

O uso de proveniência de dados pela informática vem crescendo e sua aplicação em sistemas autônomos podem trazer bons resultados; ligando as linhas de engenharia de software, banco de dados e inteligência artificial. A engenharia de software experimental é outra vertente que pode se beneficiar com a proveniência, visto que para avaliar software de modo empírico deve- se observar o máximo de detalhes possíveis, sendo este o objetivo do uso da proveniência.

A proveniência de dados possibilita registrar informações em diferentes níveis de abstração, e seu uso por meio de aspectos é uma possibilidade para que os dados sejam capturados, sem que haja a necessidade de inserção direta de código na arquitetura da plataforma multiagente. O uso de aspectos se posiciona como uma abordagem viável para este caso, contudo, deve-se ter a consciência das consequências dessa abordagem.

A proveniência dos dados pode ser usada na construção de bases de conhecimento do SMA para 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 verificação de resultados, conforme discutido anteriormente. Em sistemas multiagentes há muitas oportunidades para aplicar a proveniência dos dados.

Atualmente algumas plataformas apresentam o uso de logs, de maneira simples e que não permitem uma clara compreensão da execução do sistema em softwares complexos. A proveniência de dados é uma solução emergente, que ainda não foi abordada pela comunidade e que pode trazer bons resultados.

Esse trabalho conforme abordado em (53), é o passo inicial da discussão sobre o uso de proveniência de dados em sistemas multiagentes e como isto deve ser feito. Um mecanismo único de captura para todas as plataformas é utópico, visto a diferença entre as mesma e, a inserção de código direto na plataforma pode ser ainda pior, pois torna a manutenção da captura complexa, cresce o sistema em número de classes, métodos, números de linhas e dependendo do caso, em complexidade.

4.1 Contribuições Esperadas

Esta pesquisa encontra-se em andamento e busca pelos melhores resultados sobre o modo de captura do dados em sistemas autônomos, de forma não invasiva. Os dados de um sistema multiagente são valiosos para entender a dinâmica do sistema, e tanto a captura quanto o armazenamento são atividades críticas que devem ser bem planejadas e executadas para não invalidar as ações do sistema.

A proveniência de dados permite rastrear a origem dos dados e os processos de derivação que ocorreram entre a origem dos dados e o estado em que os dados são encontrados em um determinado momento. Se considerarmos que o modelo de proveniência contribui para a avaliação da qualidade dos dados e, consequentemente, do processo que os gerou, ele suporta maior confiança nas ações tomadas pelos agentes dentro de um sistema autônomo.

Como contribuições dessa pesquisa temos:

– Uma comparação entre os principais modelos de proveniência e qual melhor se aplica a sistemas autônomos;

– Desenvolvimento do Framework FProvW3C;

– Formalização e aplicação de um modelo de proveniência para sistemas autônomos;

– Tornar o comportamento de sistemas autônomos explicáveis por meio de proveniência.

Vale ressaltar que ao usar a proveniência para capturar os dados em um sistema autônomo, será possível avançar em outros campos, que podem envolver: i) semântica e sintaxe de dados, ii) ontologia, iii) mineração de dados e iv) dados em outros formatos, como XML ou JSON.

4.2 Limitações do Trabalho

A proposta dessa tese é utilizar a proveniência de dados capturada por meio de aspectos para tornar o comportamento de sistemas autônomos explicáveis. Como todo trabalho, está seção tem por objetivo definir alguns pontos sobre essa proposta, tanto positivos como negativos que podem de alguma forma serem tratados como limitação.

Começando pelos pontos positivos da proposta em utilizar a proveniência de dados para explicar o comportamento de sistemas multiagentes, apoiada em uma abordagem de aspectos para este caso específico, temos:

– Não afeta a arquitetura da plataforma de sistema multiagente;

– Permite interceptar qualquer ponto de execução do sistema;

– Falha no processo de captura da proveniência não afeta o software;

– Permite vincular os eventos capturados usando o modelo PROV, por meio do FProvW3C;

– Permite o reuso do código de captura em qualquer software que utilize a mesma base de plataforma de sistema multiagente (JADE, DBI4JADE);

– Facilidade para manutenção;

– Fácil acoplamento com o sistema multiagente;

Contudo, também devemos destacar alguns pontos negativos com o uso da abordagem com aspecto, que podem ser:

– Garantia que o ponto de corte está ocorrendo no local certo;

– Garantia que o dado capturado está correto;

– Excesso de chamadas na captura da proveniência em uma função;

– Código complexo após a integração da plataforma com os aspectos usados para a captura de proveniência;

Assumir que o uso de aspectos é a melhor opção para a captura da proveniência seria ingênuo ou imaturo, visto que depende do acesso ao código da plataforma e não foi discutida outas abordagens para essa captura neste trabalho além do uso de logs. Contudo, essa se mostra uma solução viável, conforme supramencionado.

Referências Bibliográficas

[1] GUBBI, J.; BUYYA, R.; MARUSIC, S. ; PALANISWAMI, M.. **Internet of things (iot): A vision, architectural elements, and future directions**. Future generation computer systems, 29(7):1645–1660, 2013.

[2] JENNINGS, N. R.; WOOLDRIDGE, M.. **Agent-oriented software engi- neering**. Handbook of agent technology, 18, 2001.

[3] JENNINGS, N. R.. **An agent-based approach for building complex software systems**. Communications of the ACM, 44(4):35–41, 2001.

[4] YU, H.; SHEN, Z. ; LEUNG, C.. **From internet of things to internet of agents**. In: GREEN COMPUTING AND COMMUNICATIONS (GRE- ENCOM), 2013 IEEE AND INTERNET OF THINGS (ITHINGS/CPSCOM), IEEE INTERNATIONAL CONFERENCE ON AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING, p. 1054–1057. IEEE, 2013.

[5] ZAMBONELLI, F.; JENNINGS, N. R.; OMICINI, A. ; WOOLDRIDGE, M. J.. **Agent-oriented software engineering for internet applications**. In: COORDINATION OF INTERNET AGENTS, p. 326–346. Springer, 2001.

[6] WOOLDRIDGE, M.; JENNINGS, N. R.. **Intelligent agents: Theory and practice**. The knowledge engineering review, 10(2):115–152, 1995.

[9] JENNINGS, N. R.. **Coordination techniques for distributed artificial intelligence**. Foundations of distributed artificial intelligence, p. 187–210, 1996.

[10] GIRARDI, R.. **Engenharia de software baseada em agentes**. In: PROCEDIMENTOS DO IV CONGRESSO BRASILEIRO DE CIÊNCIA DA COMPUTAÇÃO (CBCOMP 2004). sn, 2004.

[11] WOOLDRIDGE, M.. **An introduction to multiagent systems**. John Wiley & Sons, 2009.

[12] RAO, A. S.; GEORGEFF, M. P. ; OTHERS. **Bdi agents: from theory to practice.** In: ICMAS, volumen 95, p. 312–319, 1995.

[13] SILVA, J. C.. **Um modelo para avaliação de aprendizagem no uso de ferramentas síncronas em ensino mediado pela Web**. PhD thesis, Tese de Doutorado, PUC-RIO, 2004.

[14] FERREIRA, S. L. C.; GIRARDI, R.. **Arquiteturas de software baseadas em agen-**

tes: do nível global ao detalhado. Revista Eletrônica de Iniciação Científica da SBC, 2002.

[15] MILES, S.; GROTH, P.; MUNROE, S.; LUCK, M. ; MOREAU, L.. **Agent- prime: Adapting mas designs to build confidence**. In: INTERNATI- ONAL WORKSHOP ON AGENT-ORIENTED SOFTWARE ENGINEERING, p. 31–43. Springer, 2007.

[16] SHAW, P. H.; FARWER, B. ; BORDINI, R. H.. **Theoretical and experi- mental results on the goal-plan tree problem**. In: PROCEEDINGS OF THE 7TH INTERNATIONAL JOINT CONFERENCE ON AUTONO- MOUS AGENTS AND MULTIAGENT SYSTEMS-VOLUME 3, p. 1379–1382. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[17] WINIKOFF, M.; CRANEFIELD, S.. **On the testability of bdi agent systems**. Journal of Artificial Intelligence Research, 51:71–131, 2014.

[18] MILES, S.; MUNROE, S.; LUCK, M. ; MOREAU, L.. **Modelling the provenance of data in autonomous systems**. In: PROCEEDINGS OF THE 6TH INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS, p. 50. ACM, 2007.

[19] DELAMARO, M.; JINO, M. ; MALDONADO, J.. **Introdução ao teste de software**. Elsevier Brasil, 2017.

[20] DE ARAÚJO, T.; VON STAA, A.. **Supporting failure diagnosis with logs containing meta-information annotations**. Technical report, Technical Reports in Computer Science. PUC-Rio. ISSN 0103-9741, 2014.

[21] GÜLCÜ, C.. **The complete log4j manual**. QOS. ch, 2003.

[22] HOLZINGER, A.; BIEMANN, C.; PATTICHIS, C. S. ; KELL, D. B.. **What do we need to build explainable ai systems for the medical domain**. arXiv preprint arXiv:1712.09923, 2017.

[23] LOPEZ, F. L.. **Social power and norms**. Diss. University of Southampton, 2003.

[24] GUNNING, D.. **Explainable artificial intelligence (xai)**. Defense Advanced Research Projects Agency (DARPA), 2017.

[25] CORE, M. G.; LANE, H. C.; VAN LENT, M.; GOMBOC, D.; SOLOMON, S. ; ROSENBERG, M.. **Building explainable artificial intelligence systems**. In: AAAI, p. 1766–1773, 2006.

[26] GOEBEL, R.; CHANDER, A.; HOLZINGER, K.; LECUE, F.; AKATA, Z.; STUMPF, S.; KIESEBERG, P. ; HOLZINGER, A.. **Explainable ai: the new 42?** In: INTERNATIONAL CROSS-DOMAIN CONFERENCE FOR MACHINE LEARNING AND KNOWLEDGE EXTRACTION, p. 295–303. Springer, 2018.

[27] COELHO, R.; KULESZA, U.; VON STAA, A. ; LUCENA, C.. **Unit tes- ting in multi-agent systems using mock agents and aspects**. In: PROCEEDINGS OF THE 2006 INTERNATIONAL WORKSHOP ON SOFT- WARE ENGINEERING FOR LARGE-SCALE MULTI-AGENT SYSTEMS, p. 83–90. ACM, 2006.

[28] LEE, D.. **Sensor firm Velodyne 'baffled' by Uber selfdriving de- ath**. https://www.bbc.com/news/technology-43523286, 2018. [Online; Acessado em: 05-Janeiro-2019].

[29] SIRQUEIRA, T.; VIANA, M.; NASCIMENTO, N. ; LUCENA, C.. **A soft- ware framework for data provenance**. In: 29TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING & KNOWLEDGE ENGI- NEERING (SEKE'2017). SEKE/KNOWLEDGE SYSTEMS INSTITUTE, p. 615–619, 2017.

[30] DALPRA, H. L.; COSTA, G. C. B.; SIRQUEIRA, T. F.; BRAGA, R. M.; CAMPOS, F.; WERNER, C. M. L. ; DAVID, J. M. N.. **Using ontology and data provenance to improve software processes.** In: ONTOBRAS, 2015.

[31] SIMMHAN, Y. L.; PLALE, B. ; GANNON, D.. **A survey of data pro- venance techniques**. Computer Science Department, Indiana University, Bloomington IN, 47405:69, 2005.

[32] FREIRE, J.; KOOP, D.; SANTOS, E. ; SILVA, C. T.. **Provenance for computational tasks: A survey**. Computing in Science & Engineering, 10(3), 2008.

[33] HOLLAND, J. H.. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. MIT press, 1992.

[34] RUSSELL, S. J.; NORVIG, P.. **Artificial intelligence: a modern approach**. Malaysia; Pearson Education Limited,, 2016.

[35] SHAHRIARI, B.; SWERSKY, K.; WANG, Z.; ADAMS, R. P. ; DE FREITAS, N.. **Taking the human out of the loop: A review of bayesian optimization**. Proceedings of the IEEE, 104(1):148–175, 2016.

[36] HUHNS, M. N.; STEPHENS, L. M.. **Multiagent systems and societies of agents**.

Multiagent systems: a modern approach to distributed artificial intelligence, 1:79–114, 1999.

[37] MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F. S. R. ; MASIERO, P. C.. **Padrões e frameworks de software**. Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil, 2002.

[38] KRAVARI, K.; BASSILIADES, N.. **A survey of agent platforms**. Journal of Artificial Societies and Social Simulation, 18(1):11, 2015.

[39] BELLIFEMINE, F.; POGGI, A. ; RIMASSA, G.. **Jade–a fipa-compliant agent framework**. In: PROCEEDINGS OF PAAM, volumen 99, p. 33. London, 1999.

[40] O'BRIEN, P. D.; NICOL, R. C.. **Fipa—towards a standard for software agents**. BT Technology Journal, 16(3):51–59, 1998.

[41] POKAHR, A.; BRAUBACH, L. ; LAMERSDORF, W.. **Jadex: A bdi reasoning engine**. In: MULTI-AGENT PROGRAMMING, p. 149–174. Springer, 2005.

[42] HOWDEN, N.; RÖNNQUIST, R.; HODGSON, A. ; LUCAS, A.. **Jack intelligent agents-summary of an agent infrastructure**. In: 5TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2001.

[43] BORDINI, R. H.; HÜBNER, J. F.. **Bdi agent programming in agents- peak using jason**. In: INTERNATIONAL WORKSHOP ON COMPUTA- TIONAL LOGIC IN MULTI-AGENT SYSTEMS, p. 143–164. Springer, 2005.

[44] NUNES, I.; LUCENA, C. ; LUCK, M.. **Bdi4jade: a bdi layer on top of jade**. Pro-MAS 2011, p. 88–103, 2011.

[45] BAUER, B.; ODELL, J.. **Uml 2.0 and agents: how to build agent- based systems with the new uml standard**. Engineering applications of artificial intelligence, 18(2):141–157, 2005.

[46] BOOCH, G.; RUMBAUGH, J. ; JACOBSON, I.. **UML: guia do usuário**. Elsevier Brasil, 2006.

[47] ODELL, J.; PARUNAK, H. V. D. ; BAUER, B.. **Extending uml for agents**. Ann Arbor, 1001:48103, 2000.

[49] MOREAU, L.; FREIRE, J.; FUTRELLE, J.; MCGRATH, R. E.; MYERS, J. ;

PAULSON, P.. **The open provenance model: An overview**. In: INTERNATIONAL PROVENANCE AND ANNOTATION WORKSHOP, p. 323–326. Springer, 2008.

[50] MISSIER, P.; BELHAJJAME, K. ; CHENEY, J.. **The w3c prov family of specifications for modelling provenance metadata**. In: PROCEE- DINGS OF THE 16TH INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, p. 773–776. ACM, 2013.

[51] DA CUNHA, F. J. P.; SIRQUEIRA, T. F. M.; VIANA, M. L. ; PEREIRA, C. J.. **Extending bdi multiagent systems with agent norms**. International Journal of Computer and Information Engineering, 12(5):302–309, 2018.

[52] NETO, B. F. D. S.; DA SILVA, V. T. ; DE LUCENA, C. J. P.. **Nbdi: An architecture for goal-oriented normative agents.** In: ICAART (1), p. 116–125, 2011.

[53] SIRQUEIRA, T. F. M.; VIANA, M. L.; CUNHA, F. J. P. D.; NUNES, I. ; LUCENA, C. J. P. D.. **Data provenance in multi-agent systems: relevance, benefits and research opportunities**. International Journal of Metadata, Semantics and Ontologies, 13(1):9–19, 2018.

[54] PERIARD, G.. **O que é o 5w2h e como ele é utilizado?** http://www.sobreadministracao.com/ o-que-e-o-5w2h-e-como-ele-e-utilizado/, 2009. [Online; Aces- sado em: 05-Janeiro-2019].

[55] TAN, W. C.. **Research problems in data provenance.** IEEE Data Eng. Bull., 27(4):45–52, 2004.

[56] GARCIA, A.; SANT'ANNA, C.; FIGUEIREDO, E.; KULESZA, U.; LUCENA, C. ; VON STAA, A.. **Modularizing design patterns with aspects: a quantitative study**. In: TRANSACTIONS ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT I, p. 36–74. Springer, 2006.

[57] VON STAA, A.. **Programação Modular: Desenvolvendo programas complexos de forma organizada e segura**. Campos, 2000.

[58] LADDAD, R.. **Aspectj in action: enterprise AOP with spring applications**. Manning Publications Co., 2009.

[59] CUNHA, F.; DA COSTA, A. D.; VIANA, M. ; DE LUCENA, C. J. P.. **Jat4bdi: An aspect-based approach for testing bdi agents**. In: WEB INTELLIGENCE AND INTELLIGENT AGENT TECHNOLOGY (WI- IAT), 2015 IEEE/WIC/ACM INTERNATIONAL CONFERENCE ON, volumen 2, p. 186–189. IEEE, 2015.

[60] CUNHA, F.; VIANA, M.; SIRQUEIRA, T.; ROSEMBERG, M. ; LUCENA, C.. **Understanding normative bdi agents behavior**. In: SEKE, p. 221–224, 2018.

[61] VIANA, M.; ALENCAR, P. ; LUCENA, C.. **A modeling language for adaptive normative agents**. In: MULTI-AGENT SYSTEMS AND AGREEMENT TECHNOLOGIES, p. 40–48. Springer, 2016.

[62] POUTAKIDIS, D.; PADGHAM, L. ; WINIKOFF, M.. **Debugging multi- agent systems using design artifacts: The case of interaction protocols**. In: PROCEEDINGS OF THE FIRST INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYS- TEMS: PART 2, p. 960–967. ACM, 2002.

[63] KOEMAN, V. J.; HINDRIKS, K. V. ; JONKER, C. M.. **Omniscient de- bugging for cognitive agent programs**. In: PROCEEDINGS OF THE TWENTY-SIXTH INTERNATIONAL JOINT CONFERENCE ON ARTIFI- CIAL INTELLIGENCE, IJCAI, volumen 17, 2017.

[64] BÚRDALO, L.; TERRASA, A.; JULIÁN, V. ; GARCÍA-FORNES, A.. **Tram- mas: A tracing model for multiagent systems**. Engineering Applica- tions of Artificial Intelligence, 24(7):1110–1119, 2011.

[65] BALDONI, M.; BAROGLIO, C.; MASCARDI, V.; OMICINI, A. ; TORRONI, P.. **Agents, multi-agent systems and declarative programming: what, when, where, why, who, how?** In: A 25-YEAR PERSPECTIVE ON LOGIC PROGRAMMING, p. 204–230. Springer, 2010.

[66] DO NASCIMENTO, N. M.; VIANA, M. L. ; DE LUCENA, C. J. P.. **An iot-based tool for human gas monitoring**. In: IXV CONGRESSO BRASILEIRO DE INFORMATICA EM SAUDE (CBIS), volumen 1, p. 96–98, 2016.

[67] KIFOR, T.; VARGA, L. Z.; VAZQUEZ-SALCEDA, J.; ALVAREZ, S.; WILL- MOTT, S.; MILES, S. ; MOREAU, L.. **Provenance in agent-mediated healthcare systems**. IEEE Intelligent Systems, 21(6):38–46, 2006.

[68] NAJA, I.; MOREAU, L. ; ROGERS, A.. **Provenance of decisions in emergency response environments**. In: INTERNATIONAL PROVE- NANCE AND ANNOTATION WORKSHOP, p. 221–230. Springer, 2010.

[69] KOHWALTER, T.; CLUA, E. ; MURTA, L.. **Provenance in games**. In: BRAZILIAN SYMPOSIUM ON GAMES AND DIGITAL ENTERTAINMENT (SBGAMES), p. 11, 2012.

[70] SIRQUEIRA, T.; VIANA, M.; FERNANDES, C.; NASCIMENTO, N.; SAS- TRE, J.; MIRANDA, P. A.; AUGUSTO, V. ; LUCENA, C.. **Design de uma plataforma para a gestão de informações médicas distribuídas baseada em sistemas multiagentes e proveniência de dados**. Monografias em Ciência da Computação, 11, 2018.

[71] MILES, S.; WONG, S. C.; FANG, W.; GROTH, P.; ZAUNER, K.-P. ; MO- REAU, L.. **Provenance-based validation of e-science experiments**. Web Semantics: Science, Services and Agents on the World Wide Web, 5(1):28–38, 2007.