Um sistema multiagente para exames periciais em sistemas computacionais

Bruno Werneck P. Hoelz
Universidade de Brasília
bhoelz@unb.br

***Resumo*—A seleção eficaz de informações relevantes em um conjunto de mídias de armazenamento computacional durante os exames periciais é essencial para o sucesso do trabalho investigativo. Neste artigo, é proposta a aplicação de um sistema multiagente para a realização distribuída e coordenada de atividades atualmente desempenhadas pelos especialistas para a análise de grandes volumes de dados proveniente de sistemas computacionais. Com isso, os resultados podem ser obtidos de forma mais rápida e com melhor utilização dos recursos computacionais e humanos.**

***Palavras-chave*—computação forense, inteligência artificial, sistemas multiagente.**

I. Introdução

O exame pericial em um sistema computacional consiste no processo de preservar, coletar e analisar vestígios presentes no sistema de forma que possam ser apresentados e defendidos como evidência da ocorrência ou da autoria de determinado incidente ou delito. Como exemplos de incidentes podem ser citados a invasão de um servidor *web* com descaracterização de conteúdo (*defacement*) e a cópia ou acesso não autorizado de dados de sistemas informatizados. Na investigação de crimes, buscam-se vestígios dos mais diversos delitos como divulgação de pornografia infantil, falsificação de documentos e sonegação fiscal.

Com o aumento da capacidade das mídias de armazenamento e o aumento da sua presença na vida cotidiana, cresce também a demanda por exames dessa natureza, bem como o volume de material a ser analisado. Além disso, as ferramentas periciais utilizadas atualmente para o exame desse material são pouco flexíveis e não fornecem meios para analisar de maneira correlacionada um grande número de evidências. Isso impede que os especialistas realizem um exame mais abrangente e preciso.

Portanto, o que se busca é uma solução capaz de realizar procedimentos especializados de maneira flexível e abrangente, com melhor aproveitamento dos recursos computacionais e humanos. Para isso, é proposta a aplicação de um sistema multiagente para a realização de exames periciais em sistemas computacionais.

Por definição [1], um agente é um sistema computacional situado em algum ambiente e que é capaz de ações autônomas sobre esse ambiente para atingir seus objetivos. Em um sistema multiagente, de forma análoga, vários agentes trabalham em um mesmo ambiente, podendo interagir de forma competitiva ou cooperativa para atingir objetivos individuais ou globais. Neste artigo, todas as referências a agentes dizem respeito ao conceito aqui apresentado.

A seção a seguir apresenta algumas das dificuldades dos exames em sistemas computacionais. Em seguida é apresentada a proposta, incluindo um esboço da arquitetura e da forma de organização dos agentes. Na seção 4 é apresentado um protótipo limitado da proposta. A seção 5 discute a implementação e os resultados obtidos. Por fim, a seção 6 apresenta as conclusões e as possibilidades de melhoria identificadas.

II. Exame pericial de discos rígidos

Em alguns ambientes, não há um conhecimento exato do material de maior relevância para a apuração do incidente ou delito sob investigação. Um exemplo é a identificação do computador responsável por uma invasão a sítio da Internet, no caso em que o computador está em operação em um *cybercafé* ou outro local de acesso público. Outro exemplo é a identificação em uma empresa dos computadores que possuem evidências de uma fraude qualquer.

Nesses casos, uma análise prévia das máquinas suspeitas permite limitar o número de máquinas que deverão ser efetivamente examinadas. A falta de ferramentas adequadas para essa identificação resulta na coleta de computadores em excesso, aumentando o volume de material a ser examinado. Não é possível para o especialista realizar individualmente tal análise em uma grande quantidade de máquinas, devido ao tempo necessário para os exames em cada uma delas.

Após a coleta dos computadores, os mesmos são examinados em laboratório. No caso mais simples, há apenas uma máquina isolada. Assim sendo, a análise se limita ao conteúdo de um ou mais discos rígidos locais, apesar das possíveis referências a formas de armazenamento removíveis e externas. No entanto, tal caso não é o mais comum. Em geral, computadores fazem partes de redes e comunicam-se entre si. Além disso, dispositivos removíveis de grande capacidade também são facilmente inseridos e removidos.

Uma vez em laboratório, por falta de ferramentas adequadas, tais computadores e mídias são examinados isoladamente. Com isso, perde-se um grande potencial de correlação de evidências. Tal situação se aplica não só ao exame pericial com implicações forenses, mas também à resposta a incidentes em redes de computadores.

III. Proposta de um sistema multiagente

A análise pericial de sistemas computacionais é uma tarefa complexa que exige habilidades altamente especializadas. Por limitações do próprio grupo de especialistas, seja em disponibilidade de recursos ou de conhecimento, os procedimentos desejáveis nem sempre são realizados. Outras limitações como a volatilidade dos vestígios, tempo disponível para a análise e os recursos materiais disponíveis podem impedir a realização de um exame competente.

Turner [5] sugere um sistema inteligente que auxilie o investigador com limitações técnicas alertando-o da presença de arquivos de interesse para o tipo de investigação em andamento e levanta dois pontos:

- como capturar e combinar o conhecimento dos especialistas dos domínios técnico e legal?
- como saber se toda a informação relevante para a investigação foi capturada ou se nenhuma evidência de outros crimes foi deixada pra trás?

Este trabalho propõe uma abordagem multiagente para a análise de ambientes computacionais, sejam eles compostos por um computador isolado ou parte de uma rede de computadores.

O objetivo da utilização de uma abordagem multiagente é permitir a separação das especialidades em agentes inteligentes e autônomos capazes de sugerir a melhor ação com base em seu conhecimento especializado. Diversas especialidades podem ser citadas como a análise de:

- arquivos de registro do Microsoft Windows;
- *logs* de servidores *web*;
- imagens (detecção de faces, cor de pele, etc.);
- histórico e *cache* dos navegadores de Internet.

Outras atividades que servem como base da análise também podem ser atribuídas a agentes especializados como:

- teste de entropia e indexação;
- quebra de senhas;
- recuperação de arquivos;
- comparação com conjunto de *hashes*.

Como nem todas as atividades e análises acima são necessárias em todos os casos, ou mesmo possíveis devido a limitações de recursos computacionais (ex.: quebra de senhas) ou de tempo (ex.: indexação), um agente é necessário para realizar a coordenação da atividade dos demais agentes e determinar quais análises são mais adequadas para o caso em questão, levando em conta os recursos disponíveis.

Os agentes especialistas também podem ter opiniões divergentes sobre determinado arquivo ou sistema. Para isso, é sugerida uma arquitetura baseada em *blackboards*. Um sistema *blackboard* é um sistema no qual um grupo de especialistas independentes deposita suas contribuições para a resolução de um determinado problema em uma estrutura de dados compartilhada chamada de *blackboard* [1]. Nesse caso, um agente deve ser responsável pela resolução de conflitos, utilizando as informações colocadas no *blackboard* para formular a solução final.

A natureza distribuída dos sistemas multiagente permite que os recursos computacionais sejam aplicados de maneira mais eficiente, uma vez que um agente especialista pode migrar de computador em computador realizando seu trabalho. A distribuição de recursos também pode ser feita com base na importância ou mesmo no tamanho (em volume de dados) de uma determinada investigação sobre outra.

Para administrar a coordenação dos agentes, a distribuição dos recursos e a resolução de conflitos é sugerida uma organização dos agentes em quatro níveis hierárquicos. Uma organização semelhante é apresentada em [4] para um sistema de apoio à tomada de decisão.

Na arquitetura proposta, o nível mais baixo, chamado de nível operacional, inclui apenas agentes que realizam atividades especializadas como as citadas anteriormente. Nos níveis superiores estão agentes responsáveis pelas atividades de coordenação, decisão e planejamento que são chamados de gerentes. A fig. 1 apresenta a hierarquia proposta.

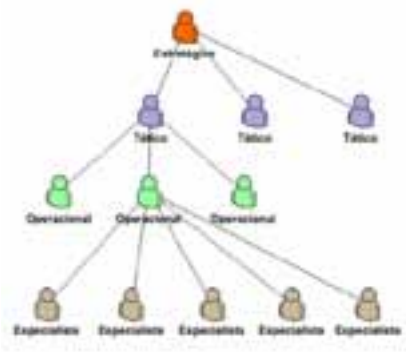

Fig. 1 – Hierarquia de agentes

A comunicação entre agentes segue a hierarquia, ou seja, um agente só se comunica com seu gerente imediato e um gerente só se comunica com seu gerente superior.

Em uma máquina isolada encontra-se um grupo de especialistas subordinados a apenas um gerente operacional, conforme apresentado na fig. 2.

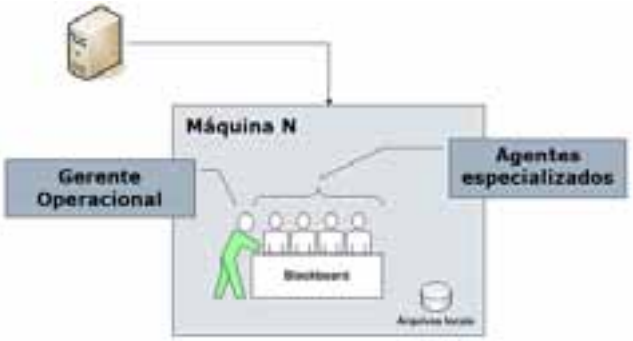

Fig. 2 – Agentes em um computador isolado

Ao agrupar diversas máquinas, um novo nível de gerencia é adicionado. O gerente tático é responsável por coordenar as atividades dos gerentes operacionais e distribuir os recursos da sua rede. A fig. 3 apresenta esse conceito.

Nesse cenário é sugerida a aplicação de um protocolo de cooperação conhecido como *Contract Nets* (CNET) [1]. Tal protocolo foi escolhido por ser um dos protocolos mais estudados e aplicados no problema de divisão de tarefas e por ser um dos protocolos de interação definidos pela FIPA (*Foundations of Intelligent Physical Agents*).

Com a utilização desse protocolo, o gerente tático é capaz de negociar o empréstimo de recursos controlados pelos gerentes operacionais, alocando-os conforme a sua necessidade. Se uma máquina X tem mais trabalho que uma máquina Y, o gerente tático pode requisitar ao gerente Y os serviços de um agente especializado necessário pela máquina X.

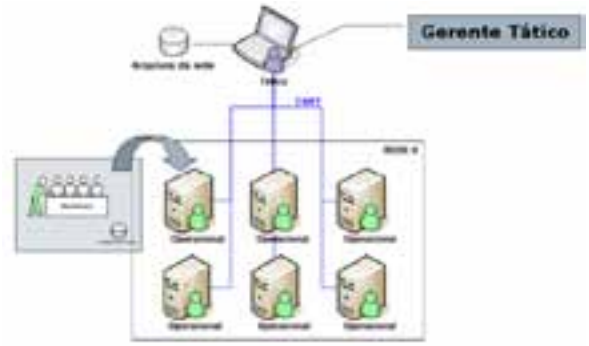

Fig. 3 – Distribuição dos gerentes em uma rede

O último nível de hierarquia é composto por várias redes. O gerente estratégico coordena da mesma forma os gerentes táticos. Portanto, a partir do topo da hierarquia, o gerente estratégico pode ter controle sobre o trabalho da organização como um todo, buscando o máximo benefício para a organização. O gerente estratégico também é o responsável por garantir que todos os agentes na organização compartilhem os mesmos significados para os mesmos objetos, ou seja, conversem entre si utilizando as mesmas definições por meio de uma base ontológica [1].

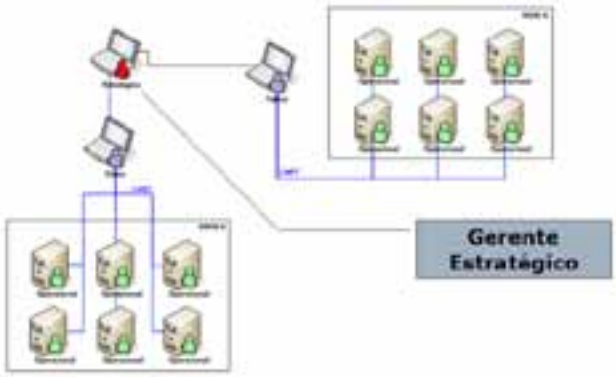

Fig. 4 – Distribuição dos gerentes em várias redes

Com essa organização, busca-se obter um sistema escalável, que possa tratar do caso mais simples onde é necessário analisar apenas uma máquina isolada até o caso mais complexo que envolva o compartilhamento de recursos de várias redes para analisar um grande volume de dados correlacionados.

IV. Protótipo

Foi implementado um protótipo limitado com o intuito de testar o conceito de hierarquia entre agentes e desempenho da distribuição de tarefas.

O protótipo apresentado implementa os dois primeiros níveis hierárquicos e um agente especialista. Um terceiro agente foi utilizado para controlar o *blackboard*. A fig. 5 a seguir apresenta a porção implementada da proposta. Os nomes dos demais agentes são apenas sugestões para análise de padrões de nomenclatura de arquivos (NamePattern), busca de palavras-chave (Keyword), linha de tempo (Timeline), perfil de navegação na Internet (WebProfile).

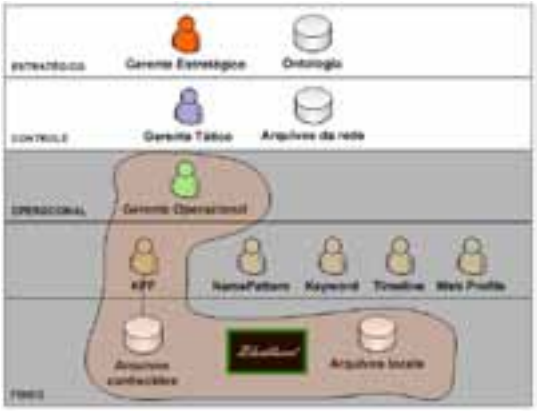

Fig. 5 – Protótipo implementado

Ao gerente operacional é atribuída a tarefa de analisar um determinado computador. Na prática, o gerente destacaria todos os agentes necessários para realizar a aquisição dos dados e quaisquer processamentos julgados necessários como cálculo de *hashes*, recuperação de arquivos e indexação de texto. No caso desse protótipo, utilizou-se como base um disco rígido de 80 GB já processado pelo Forensic Toolkit (FTK). A listagem de arquivos encontrados foi exportada para uma base de dados PostgreSQL, representando o repositório de dados de arquivos locais exibido na fig. 5.

O gerente operacional implementado realiza o particionamento da tarefa em blocos que são alocados aos agentes especialistas. Os agentes ociosos ou em excesso são dispensados. Ao final, o gerente operacional é responsável pela consulta aos fatos inseridos no *blackboard* e pela resolução de eventuais conflitos.

Na implementação atual, os conflitos são resolvidos de forma simples: sempre que um agente indicar uma evidência como importante, essa será considerada independentemente de uma opinião em contrário de outro agente. Isso é feito tendo em vista o segundo questionamento levantado por Turner [5], citado anteriormente no ínicio da seção III. Em princípio é melhor incluir alguns arquivos a mais, do que deixar algo de fora. Futuramente, deve ser implementada uma resolução de

conflito que leve em conta o nível de certeza do agente e sua taxa de acerto.

O agente especialista implementa uma funcionalidade conhecida como filtro de arquivos conhecidos ou KFF (*Known File Filter*) presente no FTK. Tal recurso serve para determinar se um arquivo é conhecido ou não. O agente considera "conhecido" o arquivo que consta de sua base de conhecimento. Tal base não armazena o conteúdo dos arquivos, mas apenas um valor de *hash* utilizando os algoritmos MD5 (*Message Digest 5*) ou SHA-1 (*Secure Hash Algorithm*), suficiente para identificá-lo unicamente de maneira estatisticamente segura, ou seja, com risco muito baixo de colisões [3]. A base também pode armazenar o conjunto de *hashes* ao qual pertence aquele arquivo (ex.: arquivo pertencente ao Microsoft Windows XP).

O objetivo do agente após a consulta a essa base é indicar se os arquivos devem ser ignorados ou encaminhados para análise. Para isso, ele mantém associado a cada conjunto de *hashes* um valor de "Ignore" ou "Alert". Um exemplo claro de alerta inclui imagens contendo pornografia infantil.

Em resumo, o trabalho desse especialista, chamado aqui de agente KFF ou `KFFAgent` é consultar a base de dados locais e verificar a presença de algum desses arquivos na sua base de arquivos conhecidos (fig. 5).

V. Implementação

Como base de arquivos conhecidos foi utilizada uma versão reduzida contendo aproximadamente 26 mil registros de *hashes* distintos. A base foi colocada em um banco de dados PostgreSQL e compartilhada entre todos os agentes KFF. Cabe lembrar que algumas das bases disponíveis contêm mais de 10 milhões de valores de *hash* distintos dos mais diversos softwares [3]. Quanto maior a base, maiores são os tempos de comparação.

Os agentes foram implementados utilizando a plataforma JADE (*Java Agent DEvelopment Framework*) [2] e a linguagem Java. A estrutura do *blackboard* foi implementada com o uso do Jess (*Java Expert System Shell*) [6]. As comunicações entre os agentes são realizadas utilizando as trocas de mensagens da plataforma JADE que seguem as especificações da FIPA.

Para os testes foi utilizada uma base de arquivos real contendo 50 mil arquivos diversos provenientes de um disco rígido de 80 GB. Foram realizadas várias execuções do programa variando o número de agentes especialistas e o tamanho dos blocos distribuídos para cada um dos agentes. Com isso foi possível testar o impacto do aspecto de distribuição (aumento no número de agentes) e da redução dos blocos, o que altera o volume de mensagens trocadas no sistema.

A. Funcionamento

Com base no número de agentes disponíveis inicialmente e no tamanho do bloco, ambos determinados pelo usuário, o gerente operacional divide os blocos e envia uma mensagem com o trabalho a ser realizado pelos agentes. Em uma implementação futura, o próprio gerente operacional deve determinar o tamanho ótimo dos blocos com base no desempenho observado, além de controlar a quantidade de agentes disponíveis. Atualmente, os agentes excedentes (aqueles que não recebem blocos inicialmente), são dispensados e removidos do sistema.

Ao receber a mensagem com a tarefa, o agente KFF inicia seu trabalho comparando os *hashes* dos arquivos que ele deve analisar com a base de arquivos conhecidos. Ao identificar algum arquivo, seja ele interessante ou não para a investigação, o agente envia uma mensagem para o *blackboard*, registrando sua decisão.

A fig. 6 apresenta a definição no Jess dos fatos registrados no *blackboard* pelos agentes especialistas. O campo `id` é o identificador do arquivo na base de arquivos locais, que contém as demais informações do arquivo como caminho completo, datas de criação, modificação e acesso, tamanhos lógico e físico, etc. O campo `agent` registra o nome do agente que enviou o fato ao *blackboard*. O campo `decision` registra a decisão do especialista em relação ao arquivo ("Ignore" ou "Alert"). O campo `description` é uma descrição adicional dependente do tipo de agente especialista. No caso do KFFAgent, ele contém o conjunto de *hashes* ao qual pertence o arquivo. A fig. 7 apresenta o exemplo do arquivo "mmsystem.dll", cujo identificador na base de arquivos locais é 6473.

```
(deftemplate blackboard-fact
  "Estrutura que armazena as decisões dos
   agentes com relação aos arquivos."
  (slot id (type INTEGER))
    (slot agent)
    (slot decision)
    (slot description))
```

Fig. 6 – Definição da estrutura do *blackboard*

```
(assert
  (blackboard-fact
    (id 6473)
    (decision "Ignore")
    (description "Microsoft Windows XP")
 ))
```

Fig. 7 – Mensagem do especialista para o *blackboard*

Ao terminar o trabalho, o agente solicita um novo bloco. Caso não haja blocos disponíveis, o agente é dispensado. Ao final do trabalho de todos os agentes, o gerente operacional pode consultar os resultados presentes no *blackboard*. A fig. 8 apresenta a definição de uma consulta ao *blackboard*. Se o gerente operacional deseja consultar os arquivos com a decisão "Alert" é só enviar uma mensagem ao *blackboard*: `(consulta ("Alert"))`. O resultado atual será uma mensagem do tipo *"Agent kff-0 recommends ALERT for file 1234"*.

Outra opção possível é definir regras que são ativadas quando um item é inserido no *blackboard*, o que permitiria ao gerente operacional tomar decisões imediatas. No exemplo da fig. 9, uma mensagem é impressa sempre que um agente registra uma decisão de alertar sobre um determinado arquivo.

```
(defquery query-files
  "Consulta arquivos por decisão."
  (declare (variables ?d))
  (blackboard (id ?id) (agent ?ag)
    (decision ?d)
    (description ?h)))

(deffunction consulta (?valor)
(printout t (facts) crlf)
(bind ?result (run-query* query-files
?valor))
(while (?result next)
  (printout t "Agent " (?result getString
ag)
  " recommends " (?result getString d)
  " for file " (?result getInt id) crlf)))
```

Fig. 8 – Consultas ao *blackboard*

```
(defrule alert-file
  "Arquivo com alerta"
  (blackboard { decision == "Alert" }
  (id ?id)) => (printout t "Arquivo com
alerta
  inserido no blackboard" crlf))
```

Fig. 9 – Regras no *blackboard*

Em um caso prático, o gerente poderia informar o usuário imediatamente da presença de tal arquivo, o que em alguns casos de investigação criminal pode significar a prisão imediata do suspeito.

B. Resultados

Com relação ao desempenho foram realizados dois testes. No primeiro, variou-se o número de agentes, mantendo fixa a quantidade de um bloco por agente. No segundo, foi utilizado um tamanho pequeno de bloco, variando-se o número de agentes. O objetivo do primeiro teste era observar os ganhos com a distribuição do trabalho. O objetivo do segundo era observar o impacto da troca de mensagens sobre o desempenho do sistema. As tabelas I e II apresentam os resultados obtidos para cada um dos testes.

O tamanho do bloco é fornecido em número de arquivos e os tempos foram registrados em segundos. O tempo mínimo ($T_{mín}$) registra o tempo com que o primeiro agente especialista termina sua execução. O tempo máximo ($T_{máx}$) registra o tempo necessário para o término do trabalho do último agente especialista. Os tempos apresentados são médias de pelo menos três execuções com os mesmos parâmetros.

TABELA I
TEMPOS DE EXECUÇÃO COM UM BLOCO POR AGENTE

Número de agentes	Tamanho do bloco	$T_{mín}$	$T_{máx}$
1	50000	50s	50s
2	25000	40s	42s
4	12500	29s	43s
8	6250	28s	45s
16	3125	5s	48s

Observou-se que com a aplicação de muitos agentes com blocos igualmente distribuídos, há a tendência de que alguns agentes terminem antes dos outros. Isso porque alguns blocos de arquivos podem conter mais arquivos conhecidos ou arquivos que não possuem valor de *hash* (ex.: como diretórios) e por isso são rapidamente processados.

TABELA II
TEMPOS DE EXECUÇÃO COM BLOCOS DE TAMANHO FIXO

Número de agentes	Tamanho do bloco	T_{mn}	$T_{máx}$
1	3125	58s	58s
4	3125	28s	42s
8	3125	29s	40s

Nota-se pelos resultados da Tabela II que o custo das mensagens que os agentes enviam ao gerente pedindo mais bloco afeta o desempenho dos agentes. Com oito agentes e blocos pequenos obtiveram-se resultados mais consistentes. Esses resultados indicam a necessidade de um controle dinâmico do número de agentes e do tamanho de blocos. A fig. 10 mostra o protótipo em execução em uma única máquina com algumas das mensagens trocadas pelos agentes.

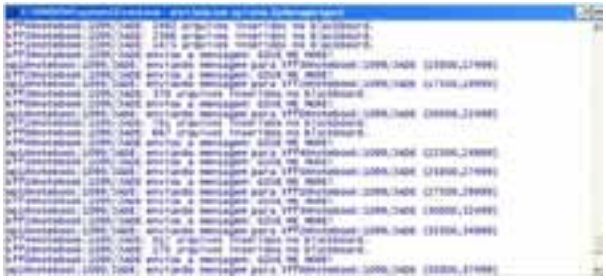

Fig. 10 – Execução do protótipo

Para efeitos de comparação, o mesmo caso foi processado utilizando a ferramenta Forensic ToolKit 1.70.1 (FTK) com a mesma base de arquivos conhecidos. O resultado médio obtido com o FTK foi de 58 segundos.

VI. CONCLUSÃO

O uso de um sistema multiagente para o exame de sistemas computacionais apresenta-se como uma abordagem interessante para melhor aproveitar os recursos computacionais à disposição dos examinadores e investigadores.

O protótipo ainda deve ser estendido para os níveis superiores da hierarquia apresentada, o que deve introduzir maior complexidade na coordenação entre gerentes. A criação de novos agentes especialistas também introduz desafios na resolução de conflitos e divisão de recursos, que devem ser incorporados aos gerentes operacionais atuais.

A aplicação de sistemas multiagente nesse caso apresentou uma redução significativa no tempo total de processamento pelos agentes KFF. A verificação de arquivos conhecidos é um dos melhores exemplos dos ganhos com a distribuição do processamento de uma evidência. Algumas melhorias poderiam ser introduzidas como a multiplicação de bases KFF para cada agente, reduzindo o custo do acesso concorrente a uma base única.

Com relação à comunicação entre os agentes, observou-se que o custo de troca de mensagens é relativamente alto. A proposta deste trabalho prevê que uma quantidade variável de

agentes decida sobre um conjunto dos arquivos. Cada agente é capaz de decidir sobre um número variável de arquivos, conforme a sua especialidade. Assim, é esperado um volume muito grande de mensagens sendo enviadas ao *blackboard* simultaneamente. Portanto, melhorias também serão necessárias na comunicação entre agentes e com o *blackboard*.

A distribuição do processamento também permite que procedimentos que consomem maior tempo possam ser realizados mais rapidamente utilizando o tempo ocioso das máquinas disponíveis, assim como já é feito no caso de aplicativos de quebra de senha distribuída.

Além disso, as possibilidades de criação de agente são muito amplas. É possível conceber agentes especializados em analisar imagens que contenham faces, que indiquem o percentual de correlação do texto de um documento com um conjunto de palavras-chave, que reconstrua eventos com base nas datas dos arquivos, que identifique sítios da Internet de interesse com base no *cache* dos navegadores. Com a arquitetura proposta, o que se busca é a possibilidade de que todas essas informações possam ser correlacionadas entre computadores de uma mesma investigação, atingindo um nível de abrangência muito maior do que o praticado atualmente.

REFERÊNCIAS

[1] G. Weiss. Multiagent Systems – A Modern Approach to Distributed Modern Approach to Artificial Intelligence. MIT Press, 1999.

[2] JADE - Java Agent DEvelopment Framework. <http://jade.tilab.com>

[3] S. Mead. Unique file identification in the national software reference library. Digital Investigation, 3(3):138–150, September 2006.

[4] S. Pinson and P. Moraïtis. An intelligent distributed system for strategic decision making. Group Decision and Negotiation, 6:77–108, 1996.

[5] P. Turner. Selective and intelligent imaging using digital evidence bags. In The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), volume 3, pages 59–64. Elsevier, September 2006.

[6] Jess - Java Expert System Shell. <http://www.jessrules.com>