Reference Architecture Based in Complex Event Processing and Agents for Controlling Distributed Systems

Carlos Alberto Athenesi
IPT – Instituto de Pesquisas Tecnológicas
Av. Prof. Almeida Prado, 532
Cidade Universitária – São Paulo/SP – Brazil
cathenesi AT gmail.com

Fernando Antonio de Castro Giorno
IPT – Instituto de Pesquisas Tecnológicas
Av. Prof. Almeida Prado, 532
Cidade Universitária – São Paulo/SP – Brazil
fgiorno AT gmail.com

ABSTRACT
Distributed systems are increasingly being used in businesses and their effective monitoring is required to maintain desired service levels. In applications that perform massive processing, the addition of controls that act reconfiguring the system components at run time is a more convenient approach than showing the states of their components in a panel, which requires human analysis. This paper proposes an architecture to building controls that can be applied on distributed systems in operation, using Agents with Complex Event Processing. The Agents are responsible for observing the components that composes the controlled system and reconfigure them when necessary, while the Complex Event Processing mechanism provides analysis capabilities for detecting working patterns that demand the reconfigurations. The architecture is tested in a proof of concept when a controlling prototype is implemented and its control behavior is observed in some desired aspects such as performance in detecting the events that represent the reconfiguration needs, the types of events that can be identified and the types of reconfigurations that can be performed on the managed system.

Categories and Subject Descriptors
I.5.5 [**Pattern Recognition**]: Implementation – special architectures

General Terms
Design, Experimentation.

Keywords
Software Architecture, Reference Architecture, Distributed System, Control System, Agent, Event Processing.

1 INTRODUÇÃO
Um ambiente distribuído provê aumento da disponibilidade e do desempenho de um sistema, tornando natural o seu uso em empresas. Os seus componentes, alocados em diversos computadores interligados em rede, comunicam-se e coordenam as suas ações através de troca de mensagens.

A monitoração desses sistemas é essencial para cumprimento dos níveis de serviço desejados; porém, o tipo mais utilizado de monitoração é passivo, onde o estado do sistema é apresentado em um painel, demandando análise e intervenção humana em caso de falhas. Uma monitoração ativa, que exerça controle e consiga alterar o sistema em tempo de execução a partir da análise de informações, geradas por sensores, sobre os estados dos seus componentes, é mais conveniente em aplicações que executam processamento massivo e que precisam manter determinados níveis de serviço, já que fica a cargo do próprio mecanismo identificar as situações que demandam a reconfiguração do sistema controlado, quer seja para atender algum tipo de carga maior de processamento ou para executar alguma função que proteja a sua execução, como a desativação de algum componente com mal funcionamento ou o bloqueio de algum usuário.

Porém, para viabilizar este tipo de controle, a verificação de informações simples, como o uso de CPU ou memória, são insuficientes, já que é necessário também analisar variáveis relacionadas às funcionalidades do sistema controlado. Controles deste tipo devem lidar com uma quantidade considerável de dados, gerados de forma contínua pelos sensores, que devem ser analisados rapidamente para que as situações que demandem reconfigurações sejam identificadas em tempo hábil.

Este trabalho apresenta uma arquitetura de referência para implementação de controles aplicáveis a sistemas distribuídos em operação especificada como um Sistema Multiagente (MAS – *Multi-Agent System*), onde os agentes são utilizados como sensores, captando informações sobre os estados dos componentes do sistema controlado, e também como atuadores, executando as reconfigurações necessárias, enquanto o Processamento de Eventos Complexos (CEP – *Complex Event Processing*) fornece a capacidade de análise das informações geradas pelos sensores, identificando as situações que demandam as reconfigurações.

As contribuições do trabalho tangem o gerenciamento de sistemas distribuídos por viabilizar um tipo de controle que pode ser implementado de maneira pouco invasiva, possibilitando a sua aplicação de forma gradativa em sistemas em operação, e a expansão de CEP para ser utilizado como componente de análise em um MAS. A seção 2 mostra os conceitos fundamentais utilizados no desenvolvimento da proposta. Na seção 3 é apresentada a arquitetura de referência. Na seção 4 são citados aspectos observados na implementação de um protótipo. O trabalho é concluído na seção 5

2 CONCEITOS FUNDAMENTAIS

2.1 Sistemas Auto Gerenciáveis
Um sistema de *software* que opera por si próprio ou com o mínimo de interferência humana é chamado autonômico. Müller et al. [7] apresentam a arquitetura elementar de um sistema autonômico. Tais sistemas contêm um componente de controle que analisa constantemente os valores dos seus atributos e atua sobre eles a fim de manter os valores próximos às especificações

978-1-5090-2435-3/16/$31.00 ©2016 IEEE

desejadas. O conjunto de atributos analisados pertencem aos elementos gerenciados, os componentes de *software* e de *hardware* que colaboram na execução do sistema controlado; dessa forma, cada sistema distribuído tem um domínio de elementos que devem ser gerenciados. As especificações de valores dos atributos que o componente de controle deve manter expressam o **estado consistente de execução** do sistema, que ocorre quando o sistema roda de maneira satisfatória. O componente de controle atua executando laços contínuos, estabelecendo um ciclo de medição, análise e reconfiguração dos recursos computacionais do sistema, conforme ilustra a Figura 1.

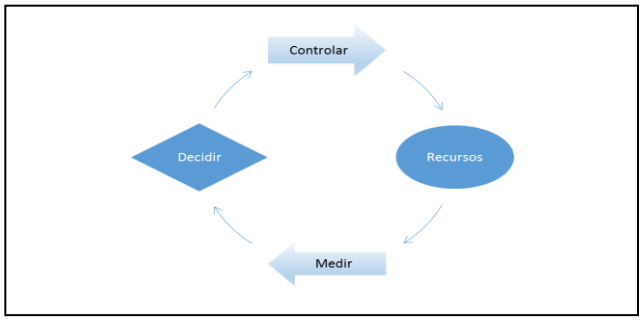

Figura 1. O ciclo de controle de Sistemas Autonômicos (baseado em [8])

O ciclo ocorre indefinidamente, enquanto o sistema operar. Assim, em tempo de execução, quando o componente de controle observa que os valores dos atributos dos elementos gerenciados saem das especificações, o estado do sistema torna-se inconsistente e ocorre uma reconfiguração dos elementos gerenciados para que os valores dos atributos fiquem novamente dentro das especificações. Kephart e Chess [3] mostram, no seu trabalho, um arcabouço arquitetural chamado gerenciador autonômico, composto por um componente controlador que se integra aos elementos gerenciados por meio de sensores e atuadores, através dos quais é estabelecido o ciclo de controle. O arcabouço determina quatro atividades; na atividade de **monitoração** são observados os sensores, que coletam dados sobre os estados dos elementos gerenciados, filtrando-os e armazenando-os em uma base de conhecimento; depois, na atividade de **análise**, os dados coletados são comparados com as especificações de valores de cada atributo e são identificados estados inconsistentes; então, a atividade de **planejamento** define as estratégias de atuação no ambiente para mantê-lo em um estado consistente; e, por fim, a atividade de **execução** coordena a reconfiguração dos elementos gerenciados através da ativação dos atuadores.

2.2 Uso de Agentes em Monitorações

A engenharia de *software* baseada em agentes fornece soluções para abordar a complexidade dos sistemas de computação que operam em ambientes que mudam com frequência. De acordo com Russell e Norvig [10], um agente é uma entidade autônoma que, quando em execução, permanece continuamente ativa observando o seu ambiente através de sensores, atualizando o seu estado interno e selecionando as ações a serem executadas utilizando-se de atuadores. São, portanto, capazes de atuar sem a intervenção humana devido ao controle que possuem sobre o seu estado e à sua capacidade de análise. Nos trabalhos pesquisados [1], [5] e [6] ocorre a preferência pelo uso de MAS em monitorações.

2.3 Processamento de Eventos Complexos

O primeiro projeto que abriu caminho para um modelo de execução genérico de CEP foi o *Rapide* [11]. Desde então, a capacidade de detecção de padrões em tempo de execução tornou CEP popular em aplicações que necessitam apresentar características preditivas ou reativas [13]. CEP é utilizado para correlacionar eventos simples, que representam ocorrências corriqueiras de um sistema, de forma a identificar padrões de comportamento. Para isto, os comportamentos que se desejam observar são definidos sob a forma de regras, que são aplicadas em um fluxo constante de eventos gerados pelos componentes do sistema. As regras especificam também quais ações devem ser executadas quando os eventos complexos são detectados, como ocorre em [2] e [13].

3 ARQUITETURA DE REFERÊNCIA

A arquitetura proposta parte de uma alteração no gerenciador autonômico de [3], de forma que o ciclo de controle passe a integrar o componente controlador aos sensores através de um fluxo que recebe eventos de todos os sensores e permite que a análise do estado do sistema relacione informações de diversos componentes. Um evento, publicado por um sensor, deve apenas representar uma mudança de estado em um componente [4] e a sua estrutura de dados deve ser simples e autocontida. Por representarem uma mudança pontual ocorrida no sistema, são denominados **primitivos** e, apenas por si, não representam um estado inconsistente do sistema, sendo necessário analisar e correlacionar vários deles para chegar a esta conclusão.

Para executar esta análise e correlação, o componente de controle precisa ter conhecimento sobre quais são os elementos gerenciados do sistema, seus atributos e os seus estados atuais. Este conhecimento deve ser armazenado em um repositório de estados instanciado na memória do componente de controle, contendo uma representação de todos os elementos gerenciados do sistema. O modelo de domínio proposto na Figura 2 propõe uma representação dos elementos de um sistema distribuído a serem mantidos pelo componente de controle.

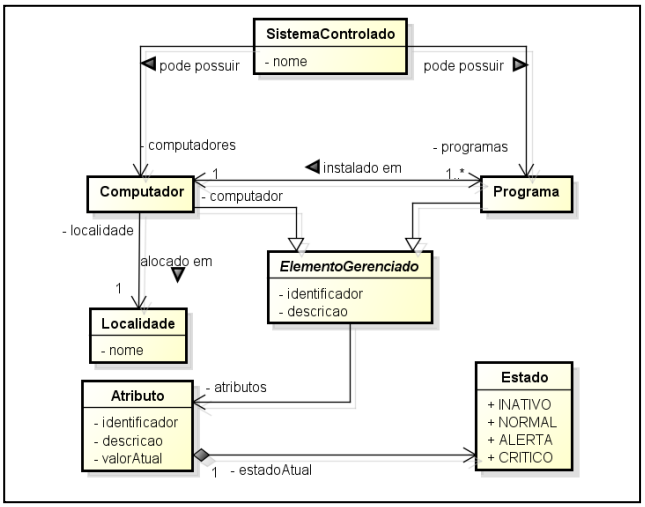

Figura 2. Modelo de Domínio dos Elementos Gerenciados do Sistema Distribuído

A Figura 3 mostra o diagrama de blocos da arquitetura de referência, que se propõe a descrever a interação entre agentes e CEP para prover um mecanismo de controle independente de tecnologias, protocolos e produtos, baseando-se no conceito de sistema auto gerenciável apresentado por [7] e visando ser genérica a fim de ser adaptada para uso em sistemas distribuídos em operação.

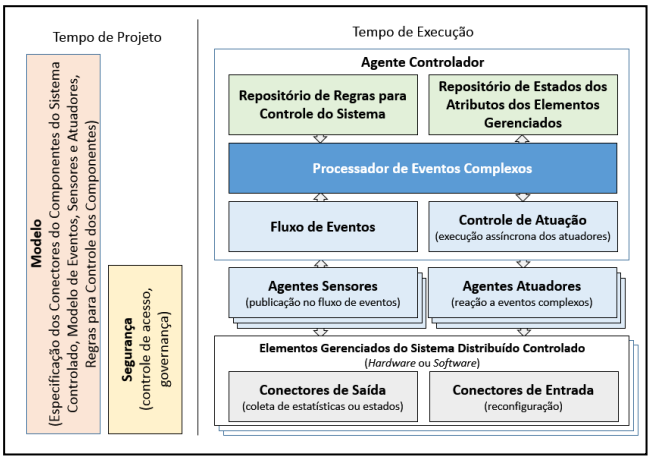

Figura 3. Blocos Conceituais da Arquitetura de Referência

Tempo de Projeto contempla atividades que devem ser executadas para implementação de um controle aderente à arquitetura proposta. Estas atividades são mostradas adiante na seção 3.1.

Em **Tempo de Execução** encontram-se os componentes necessários para implementação da arquitetura. O bloco **Agente Controlador** é composto pelo **Fluxo de Eventos**, onde são publicados os eventos primitivos; pelo **Repositório de Estados dos Atributos dos Elementos Gerenciados**, que armazena o modelo de domínio, mostrado na Figura 2, instanciado em memória, representando os atributos do sistema cujos estados são atualizados a partir dos eventos primitivos; pelo **Repositório de Regras para Controle do Sistema**, onde são definidas as regras para atualização dos estados dos atributos e as regras para determinação de estados inconsistentes; pelo **Processador de Eventos Complexos**, responsável por processar os eventos recebidos, atualizando os estados dos atributos do repositório e aplicando as regras que identificam os eventos complexos que indicam estados inconsistentes; e pelo **Controle de Atuação**, responsável pelo acionamento dos atuadores. Os blocos **Agentes Sensores** e **Agentes Atuadores** representam os componentes responsáveis, respectivamente, pela coleta de informações e pela reconfiguração dos elementos gerenciados. Ambos devem estabelecer conexões com os elementos gerenciados do sistema distribuído através de **Conectores**, portas a serem implementadas nos elementos gerenciados conforme as necessidades de controle.

A arquitetura proposta, portanto, compreende um MAS composto pelo agente Controlador, pelos agente Sensores e pelos agentes Atuadores. A Figura 4 ilustra a interação entre os três tipos de agentes: o envio das mensagens pelos Sensores deve ser assíncrono para que o evento primitivo seja apenas publicado no fluxo interno do agente Controlador. A comunicação do Controlador com os agentes Atuadores deve ocorrer também de forma assíncrona, por meio do componente de Controle de Atuação.

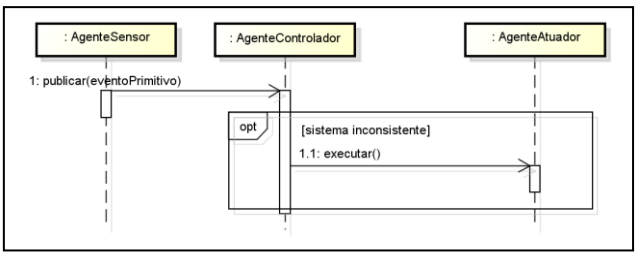

Figura 4. Interação entre os Agentes que compõem a Arquitetura

Os eventos publicados pelos Sensores devem ser usados pelo Controlador para atualizar o **Estado** de cada um dos atributos instanciados no repositório, podendo ser **INATIVO**, quando o elemento gerenciado ao qual o atributo pertence não está operante, **NORMAL**, indicando que o valor atual do atributo está dentro da faixa de valores estabelecida, **ALERTA**, indicando que o valor atual do atributo está fora da faixa de valores estabelecida, mas ainda dentro de limites aceitáveis, ou **CRITICO**, indicando que o valor atual do atributo está muito fora da faixa de valores estabelecida. Em seguida, as regras de controle devem ser aplicadas ao repositório atualizado a fim de identificar situações de mau funcionamento do sistema.

A Figura 5 apresenta o ciclo de controle, que deve ser executado internamente no agente Controlador, iniciando quando algum dos agentes Sensores publica um evento e finalizando após a análise dos estados dos atributos, se o sistema estiver em um estado consistente, ou após a execução de um agente Atuador, se for detectado um estado inconsistente.

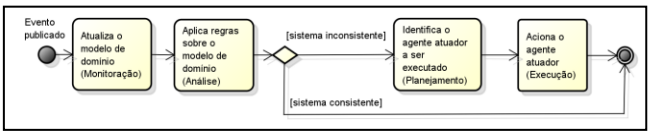

Figura 5. Comportamento Interno do Agente Controlador

3.1 Instanciação da Arquitetura

Para viabilizar a instanciação da arquitetura em sistemas em operação é proposto um processo usando um meta-modelo SPEM [8], baseado nos estágios da Metodologia Gaia [14] – voltada à implementação de sistemas baseados em agentes. A Figura 6 mostra um diagrama de pacotes contendo atividades a serem executadas para instanciar a arquitetura para criação de controles em sistemas distribuídos em operação; cada pacote corresponde a um estágio de implementação da Metodologia Gaia.

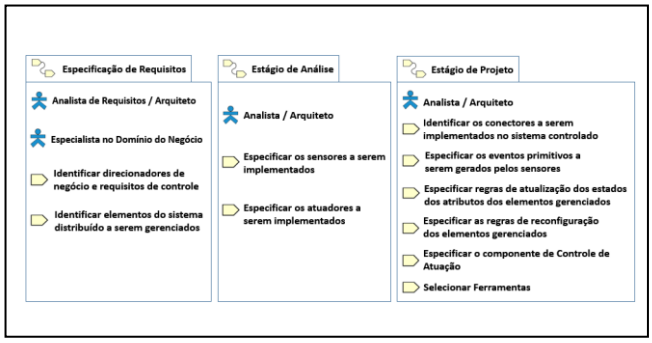

Figura 6. Atividades para Instanciação da Arquitetura

4 IMPLEMENTAÇÃO DE PROTÓTIPO

Para verificar a aplicação prática da arquitetura, foi construído um pequeno sistema distribuído emulando um cenário de negócio hipotético, distribuído em máquinas virtuais Java. O sistema foi composto por duas instâncias de uma aplicação que respondiam a requisições HTTP em intervalos de tempos entre 900ms e 1s e por um balanceador de carga, que distribuía as requisições entre as duas instâncias, simulando um processamento de transações *online*.

A partir deste sistema hipotético, composto de três elementos gerenciados, foi implementado um protótipo de controle, aderente à arquitetura proposta, simulando um caso de uso onde as instâncias da aplicação deveriam ser ativadas ou desativadas, em tempo de execução, para atender demandas variáveis de processamento. Na implementação do protótipo foram utilizadas ferramentas identificadas nos trabalhos referenciados: Jade [12], plataforma de agentes e Drools Fusion [9], processador de eventos complexos. Usando a sintaxe da ferramenta Drools, foram especificadas duas regras: uma para ativar a instância número 2 quando o número de requisições simultâneas recebidas no balanceador de carga ultrapassasse um determinado valor, e outra para desativá-la quando o número de requisições caísse abaixo do valor. A Figura 7 ilustra a distribuição do sistema distribuído e dos componentes de controle.

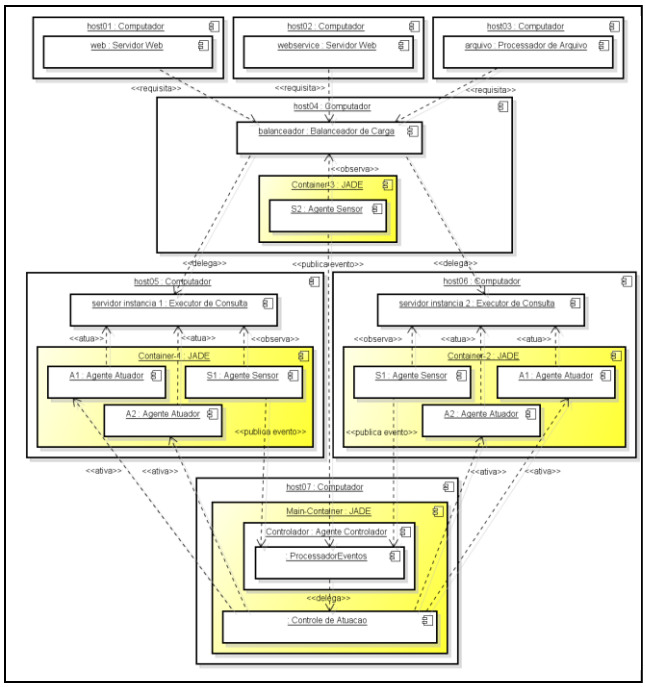

Figura 7. Sistema Distribuído e Controle

Foi executado um caso de teste que simulou volumes variáveis de processamento, aumentando e diminuindo o número de requisições no balanceador, ocasionando a ativação das regras predefinidas e a execução das reconfigurações no sistema controlado. A execução dos testes ocorreu em um computador com processador Intel i7 de 3,6 GHz, de 8 CPU's, e 8 GB de memória RAM, rodando Linux Ubuntu 14.04, IDE Eclipse Luna e Java Oracle 1.8.0_25. O balanceador de carga usado foi o Haproxy versão 1.5.14.

Foram observadas algumas características que determinam as capacidades de análise e adaptação de agentes [10]:

a) **Base de Conhecimento:** a definição dos estados dos elementos do sistema em nível de atributos e a especificação de regras usando a ferramenta Drools tornou possível relacionar estados de diversos atributos possibilitando um amplo leque de combinações e aumentando a capacidade de conhecimento do controle;

b) **Capacidade de Adaptação:** da forma proposta, qualquer agente Atuador pode ser acionado a partir de uma regra e, portanto, qualquer reconfiguração que possa ser executada por um agente é suportada;

c) **Desempenho:** nos testes, executados a partir do protótipo com sensores que geravam eventos a cada 1s, foi verificado que o tempo desde a detecção de uma situação que demandava reconfiguração até a ativação do respectivo agente Atuador foi imperceptível. Isto, entretanto, não significa que as atuações sobre os elementos gerenciados ocorreram instantaneamente: o funcionamento verificado sugere que o desempenho da atuação depende diretamente da infraestrutura de rede onde o controle atua, já que em um cenário real os agentes atuadores funcionariam remotos ao componente de controle, e também da própria ação a ser executada pelo atuador. Quanto ao uso de recursos, o componente de controle central consumiu entre 15% e 18% de CPU e entre 100 MB e 200 MB de memória RAM. O cenário de teste foi repetido com a frequência modificada para gerar duas e três vezes mais eventos e, ainda assim, não foram verificadas alterações no consumo de CPU e memória. Um ponto a ser observado é esses consumos dependem do tempo de expiração dos eventos, configurado nas regras da ferramenta Drools, e deve ser ajustado de acordo com cada sistema distribuído a ser controlado.

5 CONCLUSÃO

É notória a importância de um gerenciamento efetivo de sistemas distribuídos, dada a sua complexidade e diversidade de uso. Porém, o gerenciamento mais utilizado é passivo, baseado na exibição do estado geral do sistema em algum tipo de painel, demandando intervenção humana na resolução de falhas de funcionamento ou sobrecargas de uso.

O objetivo deste trabalho foi a especificação de uma arquitetura de referência para implementar controle em sistemas distribuídos de forma a agregar capacidades de auto gerenciamento aos sistemas controlados, permitindo que os componentes sejam reconfigurados na iminência de ocorrerem situações predefinidas.

Através da revisão bibliográfica foi identificado que o uso de agentes, em controles como o proposto, se torna viável devido às características desses componentes, como a execução autônoma e a capacidade de colaboração. Porém, o controle de vários componentes de um sistema geraria um volume grande de dados coletados sobre o estado em tempo de execução de cada um desses componentes. Para este fim, foi identificado o uso da técnica de processamento de eventos complexos (CEP). Entretanto, dentre as referências consultadas, embora tenha sido apurada a disponibilidade de ferramentas para implementação, não foi identificado o uso de agentes em conjunto com CEP, fato que adiciona a este trabalho um item inovador.

A arquitetura proposta, bem como o processo definido para instanciá-la, foram testados na implementação de um protótipo que simula a reconfiguração de um sistema distribuído. A implementação, feita em linguagem Java e utilizando as ferramentas JADE, como plataforma de agentes, e Drools, como processador de eventos complexos, se mostrou viável.

A arquitetura, da forma proposta, mantém o controle centralizado no agente Controlador. Porém, a centralização do controle em um sistema composto por muitos elementos gerenciados pode causar gargalos de processamento, que se torna indesejável. Além disso, em um sistema distribuído composto por diversos subsistemas, pode ser necessário segmentar o controle para torná-lo mais racional.

Uma alternativa, que pode ser estudada em trabalhos futuros, seria a criação de grupos de elementos gerenciados com comunicação entre os componentes de controle, de forma a descentralizar o processamento de eventos. A Figura 8 apresenta essa possibilidade: os agentes Controladores de cada subsistema atuariam também como elementos gerenciados do sistema controlador maior, disponibilizando conectores para que o Controlador central obtenha dados sobre os estados dos subsistemas e possa atuar também sobre eles.

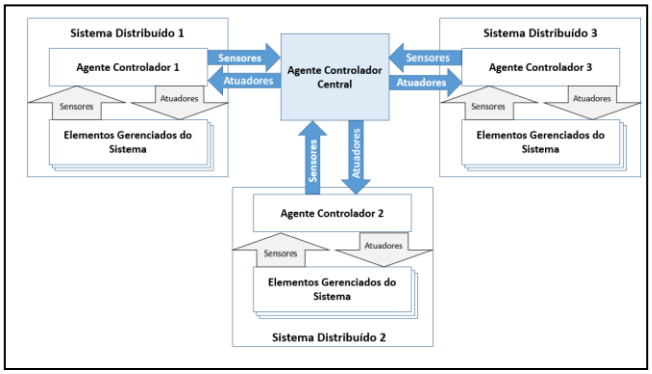

Figura 8. Composição para Controle de Subsistemas

Outra sugestão de trabalho futuro seria analisar como as atividades do presente trabalho se relacionariam com as atividades em tempo de projeto de um novo sistema distribuído, que já seria criado com capacidades de auto gerenciamento.

Mais um ponto a ser explorado futuramente refere-se às regras. A arquitetura mostrada no presente trabalho considera que as regras que indicam as situações de reconfiguração dos componentes do sistema devem estar predefinidas. Uma sugestão de pesquisa seria explorar o uso de técnicas de aprendizado de máquina para criar as regras de forma dinâmica, tornando a arquitetura ainda mais robusta.

Um outro estudo importante, que pode ser explorado futuramente, refere-se à identificação de quais técnicas de segurança seriam mais adequadas para proteção da interação entre os agentes e os conectores do sistema controlado, dada a relevante preocupação atual com questões de segurança da informação.

Também, a implementação da arquitetura poderia ser avaliada em outras plataformas de agentes, em outras linguagens de programação e controlando modelos de sistemas distribuídos diferentes do abordado neste trabalho.

6 REFERÊNCIAS

[1] Andreolini, M., Colajanni, M. and Pietri, M. 2012. A Scalable Architecture for Real-Time Monitoring of Large Information Systems. In *Proceedings of the Second Symposium on Network Cloud Computing and Applications* (London, England, 2012).

[2] Fengjuan, W., et al. 2013. The Research on Complex Event Processing Method of Internet of Things. In *Proceedings of the Fifth International Conference on Measuring Technology and Mechatronics Automation* (Hong Kong, China, 2013).

[3] Kephart, J. and Chess, D. 2003. The Vision of Autonomic Computing. *IEEE Computer*, volume 36, USA.

[4] Luckham, D. 2001. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, USA.

[5] Manzoor, U. and NEFTI, S. 2008. Agents Based Activity Monitoring System (ABAMS). In *Proceedings of the IEEE 20th International Conference on Tools with Artificial Intelligence* (Dayton, USA, 2008).

[6] Mechtri, L., Tolba, F. and Ghanemi, S. 2012. MASID: Multi-Agent System for Intrusion Detection in MANET. In *Proceedings of the 2012 Ninth International Conference on Information Technology - New Generations* (Las Vegas, USA, 2012).

[7] Müller, H. A., O'Brien, L., Klein, M. and WOOD, B. 2006. Autonomic Computing. Technical Note. Software Engineering Institute, Carnegie Mellon University, USA. Retrieved September, 2014, from http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tn006.pdf

[8] Object Management Group. Software & Systems Process Engineering Metamodel – SPEM. Retrieved June, 2015, from http://www.omg.org/spec/SPEM/

[9] Red Hat. Drools: the Business Logic integration Platform. Retrieved February, 2015, from: http://drools.jboss.org

[10] Russell, S. and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Pearson, USA.

[11] Stanford University. 2002. The Stanford Rapide Project. Stanford University, USA. Retrieved February. 2015. From http://complexevents.com/stanford/rapide

[12] Telecom Italia. Jade: an open source platform for peer-to-peer agent based applications. Retrieved February, 2015, from http://jade.tilab.com

[13] Wang, D., et al. 2011. Active Complex Event Processing Infrastructure: Monitoring and Reacting to Event Streams. In *Proceedings of the IEEE 27th International Conference on Data Engineering* (Hannover, Germany, 2011).

[14] Wooldridge, M., Jennings, N. R. and Kinny, D. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. Department of Computer Science. University of Oxford, UK. Retrieved January, 2015, from http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/jaamas2000b.pdf