5
Trabalhos Relacioandos

Neste capítulo, são descritos alguns trabalhos relacionados enfatizando a relação entre eles e o trabalho proposto. A principal contribuição deste trabalho é a proposta de um *framework* para construção de sistemas baseados em agentes auto-adaptativos orientados a serviços. Abordagens que visam a construção de sistemas auto-adaptativos são apresentadas e analisadas neste Capítulo.

5.1
Dynamic Service-oriented Architecture (DySOA)

DySOA (Bosloper et al. 2005) é um framework para monitoramento de aplicações baseadas em serviços, avaliação das informações monitoradas, e adaptação em tempo de execução, caso a aplicação não esteja satisfazendo os requisitos de qualidade de serviço (QoS).

Figura 5.1 apresenta uma arquitetura em alto nível do DySOA. Ela consite dos componentes *Monitoring*, *Evaluation*, *Analysis* e *Configuration*. O componente *Aplication* que não pertence ao *DySOA*, é o sistema baseado em serviços que o DySOA monitora.

A fim de saber se os requisitos de QoS estão sendo satisfeitos, devemos ser capazes de avaliá-los contra os QoS esperados. DySOA fornece um componente *Evaluation* que implementa essa funcionalidade. O componente *Evalution* compara as informações de QoS com os requisitos de QoS da aplicação e determina se a aplicação respeita estes requisitos.

A fim de ser capaz de avaliar o atual QoS contra os requisitos de QoS, diferentes tipos de informações sobre o contexto da aplicação em execução têm de ser monitorados: os serviços individuais, o ambiente de execução da aplicação (por exemplo: rede, plataforma de processamento), ou o contexto do usuário (por exemplo, localização, comportamento). O componente *Monitoring* provido pelo DySOA monitora o contexto destas aplicações. Já o componente *Analysis* é responsável por formatar as informações monitoradas pelo componente *Monitoring* em informações de QoS. De maneira que elas possam ser facilmente entedidas pelo componente *Evalution*.

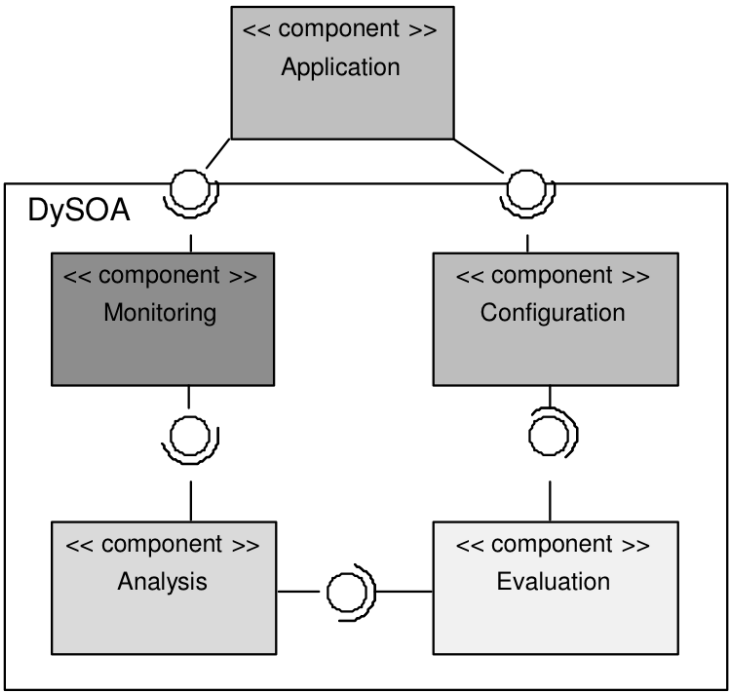

Figura 5.1: Arquitetura em Alto Nível da arquitetura DySOA

No caso da aplicação não estar satisfazendo os requisitos de QoS, uma nova configuração deve ser determinada. Neste caso, o componente *Evaluation* notifica o componente *Configuration* dos resultados da avaliação. O componente *Configuration* determina como reconfigurar a aplicação, com base na avaliação dos resultados e as configurações alternativas. Uma reconfiguração possível é a substituição de um serviço em uso. Depois de decidir o que mudar, o componente *Configuration* realiza as reconfigurações.

O JAAF-S oferece mecanismos que possibilitam a efetivação de todas as atividades realizadas pelo DySOA. Entretanto, o DySOA não faz uso de alguns dos benefícios inerentes ao paradigma de sistemas multiagentes que podem contribuir para o desenvolvimento de sistemas complexos. A exemplo sociabilidade e melhor visualização das entidades presentes no sistema. Além disto, no DySOA somente é possível determinar novas configurações (soluções) se a relação problema-solução está bem definida. Ou seja, dado que ocorreu um problema A é sabido que uma solução B deve ser aplicada. Enquanto, que o JAAF-S ao utilizar raciocínio baseado em casos para descoberta de novas soluções e uma abordagem baseada em utilidade para seleção da melhor solução oferece a flexibilidade necessária para realização de auto-adaptação mesmo diante de problemas inesperados. A importância do raciocínio baseado em casos e uma abordagem baseada em utilidade para construção de sistemas auto-adaptativos pode ser vista em (Khan et al. 2008) (Petrucci 2008), respectivamente.

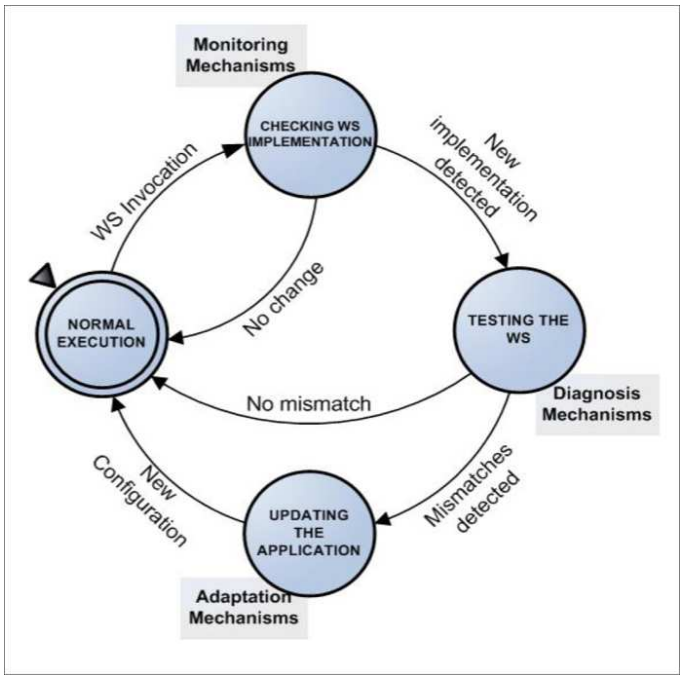

Figura 5.2: *Control-loop* de auto-adaptação

5.2 Denaro et. al.

Denaro et. al. (Tosi and Denaro 2009) propõe um abordagem para desenvolver aplicações auto-adptativas orientadas a serviços, particularmente ele foca em problemas de integração derivados de mudanças dinâmicas em serviços invocados. Para tanto, ele propõe uma abordagem que utiliza casos de teste para revelar possíveis *mismatches* entre o serviço requisidado e o provido, e dinamicamente selecionar adaptações para autonomamente ajustar o problema de integração.

Tal abordagem combina diferentes técnicas num *control-loop*. Figura 5.2 apresenta como o *control-loop* foi instanciado para permitir aplicações auto-adaptativas orientadas a serviços. A invocação de um WS dispara o mecanismo *monitoring* que verifica se a causa do disparo é um novo serviço, neste caso o mecanismo *diagnosis* executa casos de teste sobre o serviço para revelar possíveis *mismatches*. Se o mecanismo *diagnosis* revela *mismatches*, ele aciona o mecanismo *adaptation* que realiza a adaptação visando solucionar os problemas identificados.

Assim como no DySOA, a abordagem proposta por (Tosi and Denaro 2009) não faz uso de benefícios inerentes ao paradigma de sistemas multiagentes. Além disto, para realização da abordagem (Tosi and Denaro 2009) é necessário ter bem definido um conjunto de casos de testes e suas respectivas

estratégias de adaptação. Tal necessidade, assim como no DySOA, nos leva a uma abordagem que não oferece uma solução flexível o bastante para construção de sistemas auto-adaptativos. Pois, como mencionado em (Cheng and et al. 2009), uma boa solução para sistemas auto-adaptativos deve apresentar flexibilidade para lidar com incertezas inerentes à domínios que necessitam de auto-adaptação. Como já mencionado anteriormente o JAAF-S fornece uma solução flexível para lidar com tais incertezas.

Por fim, o JAAF-S sendo uma evolução do JAAF 1.0 basea-se numa abordagem onde diferentes *control-loops* podem ser implementados, diferentemente do DySOA e (Tosi and Denaro 2009) que basea-se num único *control-loop*. A necessidade de diferentes *control-loops* pode ser vista em (Costa et al. 2010).

5.3 Agent Service Oriented Architecture

O trabalho em (Poggi et al. 2007) propõe um framework baseado em agentes orientados a serviços que permiti a composição de WS em tempo de execução. Sua arquitetura é baseada em dois tipos de agentes (Figura 5.3): *Component Managers* e *Workflow Managers*.

Cada *Component Manager* está associado a um ou mais WS e é responsável pela interação com eles. Através do uso da JADE WSIG add-on (GreenWood and Callisti 2004), os *Component Manager* são capazes de invocar um WS.

Já os *Workflow Managers* tem o objetivo de apoiar os usuários no processo de construção de *workflows*, composição de WS e monitoramento de sua execução. Para realizar essas atividades os *Workflow Managers* fornecem aos usuários dois processos alternativos: (i) *workflow* pré-definido, neste caso a obrigação do *Workflow Manager* é apoiar o usuário na seleção dos serviços mais adequados para execução das tarefas que compõem o *Workflow*, e (ii) e *workflow* dinâmico, neste caso *Workflow Manager*, a partir dos requisitos do usuário, compõe um novo *workflow* apartir dos serviços disponíveis no sistema. Isto é feito aplicando um *planner*(SGP (Weld 200)) que trabalha sobre informações extraídas a partir de descrições OWL-S dos WS, providos pelo *Component Managers*.

A característica peculiar da abordagem proposta em (Poggi et al. 2007) é a integração da tecnologia de agentes com WS, regras e OWL-S. Entretanto, além do JAAF-S apresentar tais características, adicionalmente ele oferece mecanismos para descoberta (a exemplo do RBC) e seleção (a exemplo da abordagem baseada utilidade) de WS que possibilitam lidar com situações inesperadas. Além disto, o JAAF-S é um *framework* que oferece a flexibilidade

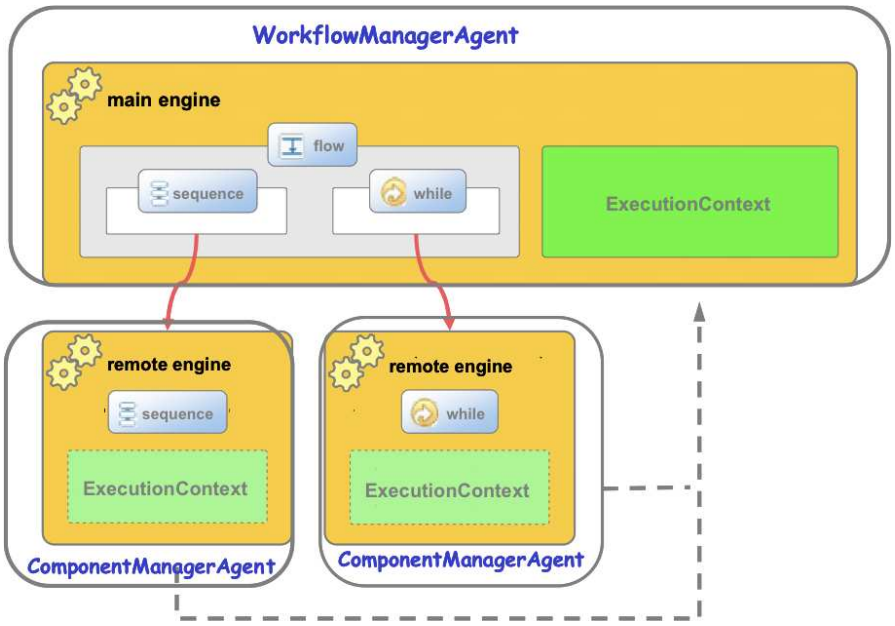

Figura 5.3: Visão Geral do Framework

necessária para que novos mecanismos para monitoramento, detecção e seleção de WS sejam acoplados facilmente e sem grande aumento na complexidade para gerenciar o processo de auto-adaptação.

5.4 Framework SEMMAS

O framework SEMMAS(Sanchez et al. 2009) é baseado em uma arquitetura multi-camadas composta por quatro camadas. São elas (Ver figura 5.4): *business logic*, *semantic web service*, *intelligent agents* e *application*.

A camada inferior (*Business Logic*) oferece as uncionalidade da lógica de negócios. Um conjunto de WS disponibiliza tais funcionalidades. A segunda camada *Semantic Web Service* adiciona anotações semânticas as capacidades de tais serviços, com isso possibilitando que entidades de software interajam com tais serviços de forma dinâmica e sem intervenção humana. Em particular, os novos serviços podem surgir e outros podem mudar a sua funcionalidade ou mesmo desaparecer em *run-time*, mas o sistema continuará funcionando e as alterações se refletirão sobre a aplicação imediatamente. Estas sofisticadas entidades de software, chamadas agentes inteligentes, que interagem e usufruem das vantagens dos serviços oferecidos estão localizados na camada *Agentes Inteligentes*. Agentes inteligentes fazem uso da anotação semântica das capacidades dos serviços para automaticamente descobrir, compor, invocar e monitorar WS. Eles também são capazes de dinamicamente propagar as mudanças

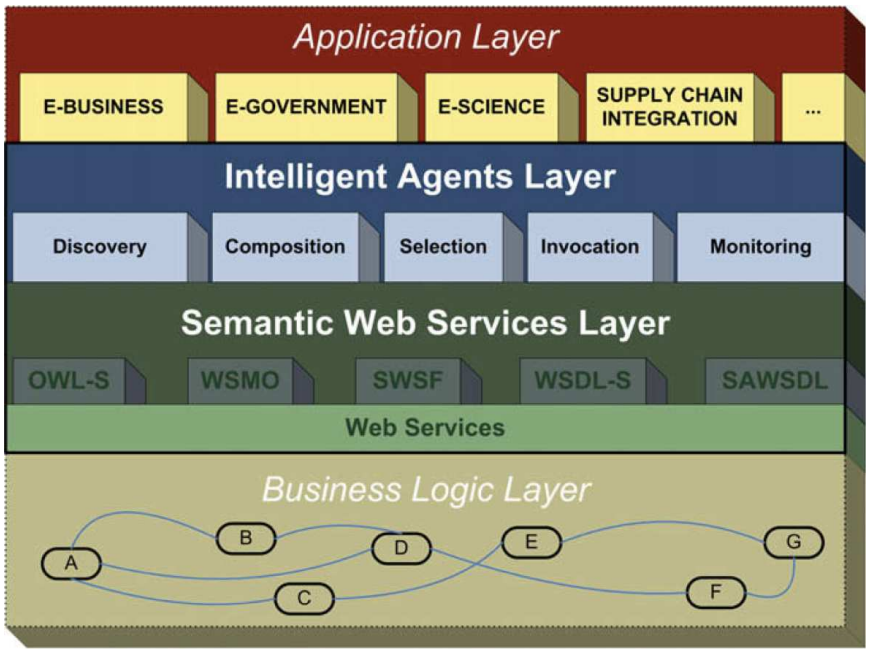

Figura 5.4: Arquitetura do Framework SEMMAS

ocorridas nas funcionalidades providas pela camada *Business Logic*. Por fim, a camada *Application* é responsável por coordenar os agentes de maneira que eles realmente provejam atividades úteis para os usuários.

O framework SEMMAS foca nas camadas *Intelligent Agents* e *Semantic Web Service*. Na camada *Intelligent Agents*, SEMMAS oferece um conjunto de agentes, são eles: (Provider Agent) atua como uma representação do provedor de serviços; (Service Agent) atua como uma representação do serviço; (Customer Agent) atua como uma representação do usuário; (Discovery Agent) responsável por procurar no repositório de SWS serviços que satisfaçam os requisitos providos por usuários; (Selection Agent) responsável por selecionar o serviço mais apropriado dentre os vários descobertos pelo (Discovery Agent) a partir das preferências do usuário; (Broken Agent) responsável por solucionar problemas de inteoperabilidade e (Framework Agent) responsável por monitorar e assegurar o correto funcionamento do sistema.

Embora SEMMAS forneça uma infra-estrutura que possibilite a realização de tarefas importantes no âmbito de auto-adaptação orientada a serviços, tais como monitoramento, descoberta, seleção e invocação de serviços em tempo de execução. O JAAF-S apresenta uma abordagem simples e facilmente estendida. (Simples) No JAAF-S tais tarefas são realizada através de um *control-loop* gerenciado por um único agente, já no SEMMAS é necessário um conjunto de agentes para realizar tais tarefas. (Facilmente estendida) No JAAF-S novos mecanismos que facilitem a realização de auto-adaptação

podem ser adicionados implementando as interfaces *IReasoningStrategy* (Ver Seção 3.3.2), *IActuator* (Ver Seção 3.3.4), estendendo as classes abstratas *SelectionStrategy* (Ver Seção 3.3.3), *Monitor* (Ver Seção 3.3.1) ou criando novas atividades e definindo novos *control-loops* (Ver Seção 3.3), já no SEMMAS é necessário a criação de novos agentes com isso trazendo um aumento na complexidade para coordenar um maior número de agentes.

5.5 *The Agent Building and Learning Environment*(ABLE)

ABLE (Bigus et al. 2002) é um framework JAVA que permiti a construção de agentes inteligentes utilizando *machine learning* e raciocínio. Ele fornece um conjunto de interfaces JAVA e as classes base para construir um biblioteca de mecanismos capazes de ler e escrever em arquivos com extensão ".txt", *databases*, realizar inferência baseada em regras utilizando lógica booleana e *fuzzy*, aplicar técnicas de *machine learning*, a exemplo redes neurais e árvore de decisão.

O JAAF-S fornece uma infra-estrtura flexível o bastante para inserção dos diferentes mecanismos providos pelo ABLE. Além disto, como o JAAF-S estende o JADE, tais mecanismos estariam em completa sincronia com os mecanismos já fornecidos pelo JADE. Desta forma, permitindo a construção de SMA muito mais poderosos.

Por fim, o ABLE não faz uso dos benefícios trazidos pela tecnologia de SWS.