4 - *Framework* proposto para Sistemas Multi-Agentes Abertos

Neste capítulo propõe-se um conjunto de conceitos para a especificação do gerenciamento de contratos. O modelo proposto nesta dissertação aborda quatro tipos de contratos: contrato de serviço, contrato de componente, contrato de *assembly* e contrato de composição. A figura 19 ilustra uma visão geral dos principais elementos do modelo conceitual do *framework*. O restante do capítulo detalha esses elementos e descreve como eles interagem entre si para permitir a aplicação de contratos em ambientes multi-agentes abertos.

4.1 Modelo Conceitual

O modelo conceitual é composto por um conjunto de conceitos que permitem identificar o serviço que um agente está executando e confrontá-lo com o contrato assumido pelo agente dentro da organização. A figura 19 apresenta os principais elementos do modelo conceitual do *framework*.

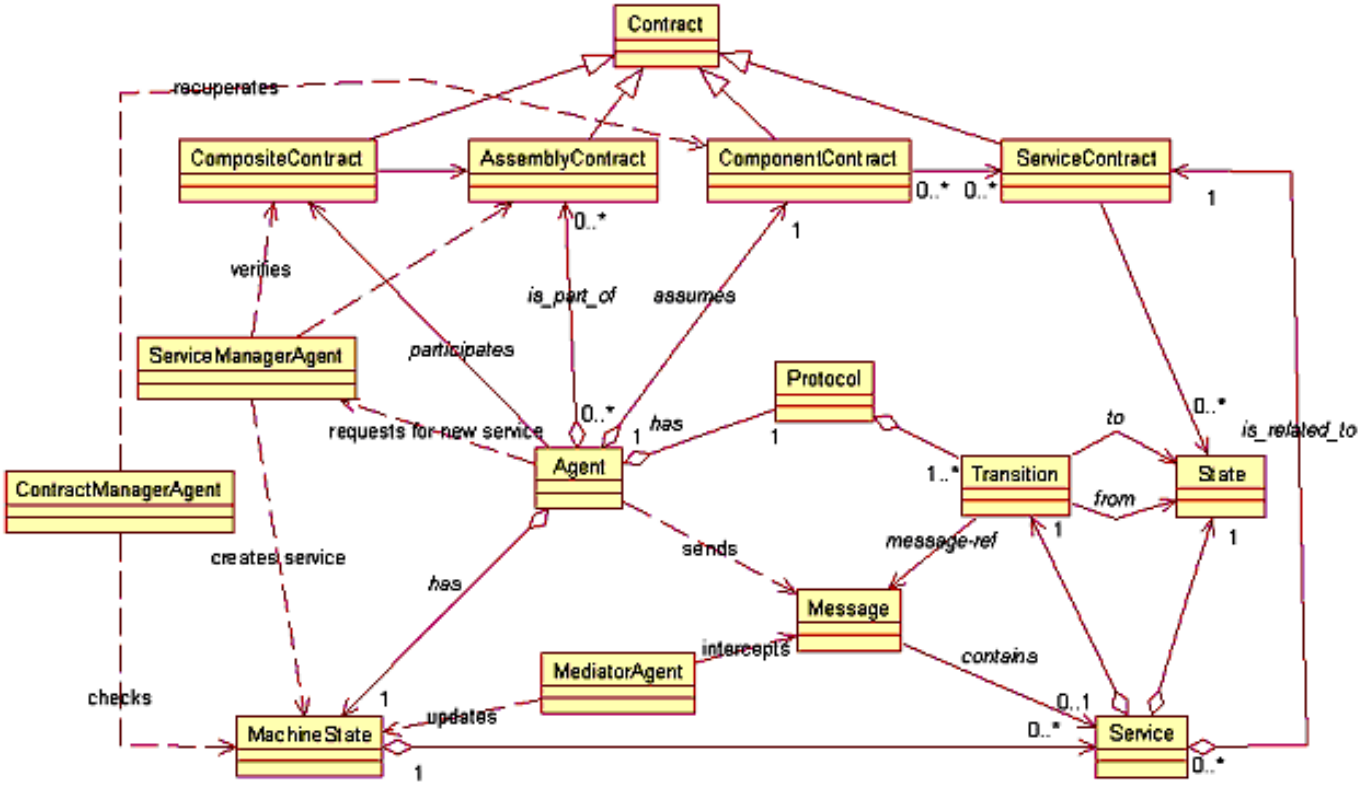

Figura 19: Modelo conceitual do *framework* proposto nesta dissertação.

Os agentes da organização que se encontram sob o controle de contratos possuem cada um uma máquina de estados responsável por armazenar a situação do andamento de cada serviço prestado pelo agente.

O agente Mediador (no modelo: *MediatorAgent*) é responsável por alimentar as máquinas de estados. Ele intercepta as mensagens trocadas entre os agentes, identifica o agente executor do serviço e atualiza sua máquina de estados de acordo com o protocolo de comunicação do agente executor do serviço.

O protocolo de comunicação é um ponto flexível do *framework*. Cada agente sob o controle de contratos possui seu próprio protocolo de comunicação. Os protocolos definem as transições e estados relevantes para a execução dos serviços. Os conceitos de protocolo, transição e estados serão discutidos mais profundamente na seção 4.3.2.

O modelo subdivide-se em dois módulos principais: o módulo de gerenciamento de contratos e o módulo de gerenciamento de serviços.

O módulo de gerenciamento de contratos é composto pelos quatro tipos de contratos proposto nesta dissertação: contrato de serviço, contrato de componente, contrato de *assembly* e contrato de composição; pelo agente mediador (no modelo: *MediatorAgent*) e pelo agente de gerenciamento de contratos (no modelo: *ContractManagerAgent*), que é responsável por percorrer periodicamente as maquinas de estados dos agentes e verificar a situação de seus serviços prestados frente aos contratos assumidos pelos mesmos. Este módulo será discutido na seção 4.3.4.

O módulo de gerenciamento de serviços tem como objetivo a alocação de serviços entre os agentes. A estratégia de alocação de serviços é um ponto flexível do *framework*. Um exemplo de algoritmo utilizado na alocação dos serviços é descrito na seção 4.3.5.

Elementos do Modelo Conceitual		
	Ponto Flexível	**Ponto Fixo**
Mensagem		X
Protocolo, Estado e Transição	X	
Serviço		X
Máquina de Estados		X

Tabela 2: Elementos do modelo conceitual.

Módulo de Gerenciamento de Contratos		
	Ponto Flexível	**Ponto Fixo**
Contrato de Serviço	X	
Contrato de Componente	X	
Contrato de Assembly	X	
Contrato de Composição	X	
Agente Mediador (no modelo: *MediatorAgent*)		X
Agente de gerenciamento de contratos (no modelo: *ContractManagerAgent*)		X

Tabela 3: Módulo de Gerenciamento de Contratos.

Módulo de Gerenciamento de Serviços		
	Ponto Flexível	**Ponto Fixo**
Agente Gerenciador de Serviços (no modelo: *ServiceManagerAgent*)	X	

Tabela 4: Módulo de Gerenciamento de Serviços.

4.1.1 Mensagem

O elemento *Message* permite especificar quais são as mensagens que os agentes podem trocar. No *framework* desenvolvido, este elemento possui um conjunto de atributos que todas as mensagens precisam ter, são eles:

- *Performative:* o campo *performative* serve para especificar a intenção dos agentes em uma interação com outros agentes. Os valores possíveis para este campo não está restrito a nenhum conjunto de valores específicos. A comunicação entre os agentes é geralmente baseada na teoria de atos da fala (Searle, 1969). Um grande número de linguagens de comunicação entre os agentes se baseia nessa teoria ([ACL, 2002]), (Finin et al, 1994). Elas fornecem um vocabulário que permite aos agentes expressar suas intenções. O campo *performative* permite a especificação destas intenções.
- *From:* este campo especifica o agente que está enviando a mensagem.
- *To:* este campo especifica o agente receptor da mensagem.
- *Description:* este campo representa o conteúdo da mensagem.

- *ServiceId:* este campo representa o identificador único do serviço o qual esta mensagem está relacionada.
- *Parameters:* neste campo são passados possíveis parâmetros necessários pelos agentes para a execução dos serviços.
- *InitialValues:* neste campo são passados possíveis parâmetros necessários para a inicialização do serviço.
- *Id:* este campo representa o identificador único da mensagem.

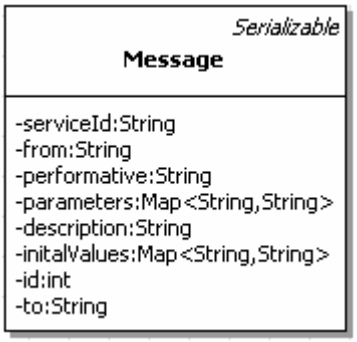

Figura 20: Mensagem.

4.1.2 Protocolo, Estado e Transição

As mensagens isoladamente não permitem especificar muita coisa em uma interação. É preciso estabelecer a ordem em que elas ocorrem e quais as mensagens válidas em um determinado momento da interação. Protocolos de interação representam justamente padrões de interação entre os agentes, definindo quais são as interações válidas e inválidas. O elemento *Protocol* representa protocolos de interação entre os agentes e é representado por um autômato finito não determinístico (Menezes, 1997), onde estados representam pontos na execução do protocolo e transições são as conexões entre os estados. Além de estados e transições, autômatos também possuem um conjunto finito de símbolos que são capazes de processar. No contexto dos protocolos, este conjunto de símbolos é representado por elementos *Message*. Ou seja, as transições entre os estados são ativadas por mensagens trocadas entre os agentes. Na Figura 21, destaca-se o elemento *Protocol* do modelo conceitual.

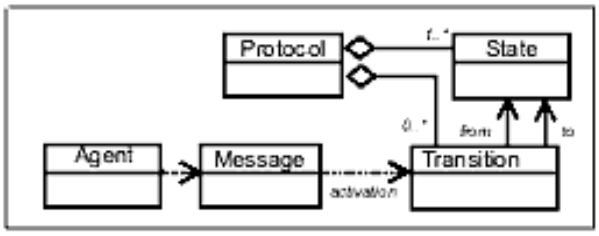

Figura 21: Foco no Protocolo.

Protocolos são representados especificando seu conjunto de estados e transições.

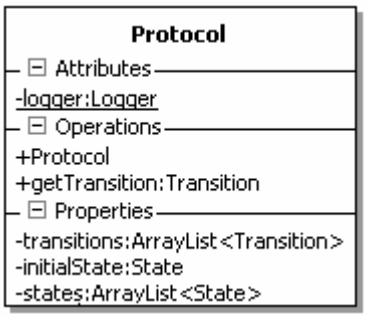

Figura 22: Protocolo.

Os estados de um protocolo indicam pontos na execução do protocolo e, portanto, podem ter diferentes significados. Entretanto, é possível identificar algumas categorias de estados: *initial*, *execution*, *success* ou *failure*. Um estado classificado como *initial* representa o estado inicial do protocolo de interação e somente um estado inicial é permitido por protocolo. Um estado *execution* significa que a execução do protocolo ainda não terminou. Estados *success* ou *failure* são estados finais e representam, respectivamente, que o protocolo foi concluído com sucesso ou com falha. Estas categorias de estados são representadas através do atributo *type*, da classe State.

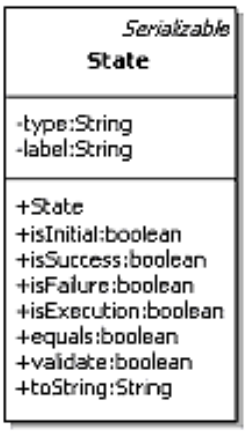

Figura 23: Estado.

As transições (*Transition*) conectam dois estados. Esta conexão é feita através dos atributos *from* e *to*, que mantêm referências para estados do protocolo. Porém, transições somente são ativadas mediante à ocorrência de uma mensagem representada pelo atributo *with-message*.

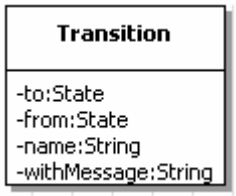

Figura 24: Transição.

4.1.3 Serviço e Máquina de Estados

Um agente pode executar diversos serviços simultaneamente para diferentes agentes requisitores. Neste contexto, é necessário armazenar para cada serviço sua transição e estado correntes. No modelo isto é feito através da máquina de estado. Cada agente possui sua própria máquina de estados, contendo a lista dos serviços que estão sendo executados pelo agente. A máquina de estados é responsável por atualizar os serviços de acordo com as mensagens trocadas entre os agentes.

Um serviço possui o seguinte conjunto de atributos:

- *Requestor:* nome local do agente que requisitou o serviço;
- *Executor:* nome local do agente que está executando o serviço;
- *Name:* nome do serviço;
- *Id:* Identificador único do serviço;
- *StartTime:* Horário no qual o serviço foi inicializado;
- *IsValid:* Chave indicando se o serviço está sendo executado ou se ele já foi terminado/cancelado;
- *Parameters:* parâmetros necessários a execução do serviço trocados entre os agentes;
- *InitialValues:* possíveis parâmetros de inicialização do serviço trocados entre os agentes;
- *CurrentTransition:* transição corrente do serviço;
- *CurrentState:* estado corrente do serviço.

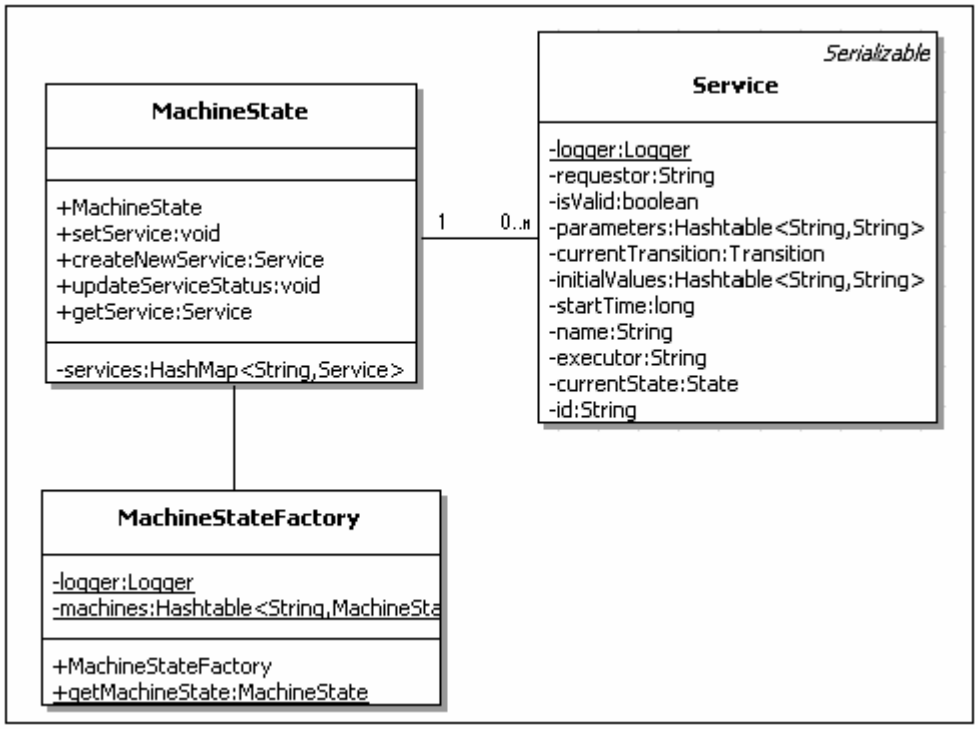

Figura 25: Serviço e Máquina de Estados.

O método *createNewService* da classe *MachineState* é resposável pela criação de um novo serviço e seu registro na lista de serviços do agente. Já o método *updateServiceStatus* utiliza o protocolo de interação do agente para atualizar o status do serviço, representado por sua transação e estado corrente, sempre que uma nova mensagem referente ao serviço sendo executado é disparada.

4.1.4 Módulo Gerenciamento de Contratos

O módulo de gerenciamento de contratos visa permitir a implementação de quatro tipos de contratos: contrato de serviços, contrato de componente, contrato de assembly e contrato de composição. Este módulo é composto pelos agentes Mediador (no modelo: *MediatorAgent*), responsável por alimentar a máquina de estados dos agentes; e pelo agente de gerenciamento de contratos (no modelo: *ContractManagerAgent*), responsável pela validação dos serviços executados pelos agentes frente aos contratos assumidos na organização.

4.1.4.1 Contrato de Serviço

Um contrato de serviço define as propriedades (estados observáveis) de um serviço, isto é, as pré e pós-condições necessárias para o correto

cumprimento do serviço. Os seguintes atributos compõem um contrato de serviço:

- *ServiceName:* nome único do serviço;
- *Duration:* duração máxima para a conclusão do serviço pelo agente executor;
- *Pré-Condições:* pré-condições necessárias para a execução do serviço. A lista de pré-condições é composta por uma lista de estados *(States)*;
- *Pós-Condições:* pós-condições necessárias para a execução do serviço. A lista de pós-condições é composta por uma lista de estados *(States)*.

As operações *validatePrecondition* e *validatePostcondition* recebem como parâmetro o identificador do serviço e o nome local do agente executante. O diagrama de seqüência da figura 26 mostra como é realizada a verificação das pré e pós-condições de um contrato de serviço:

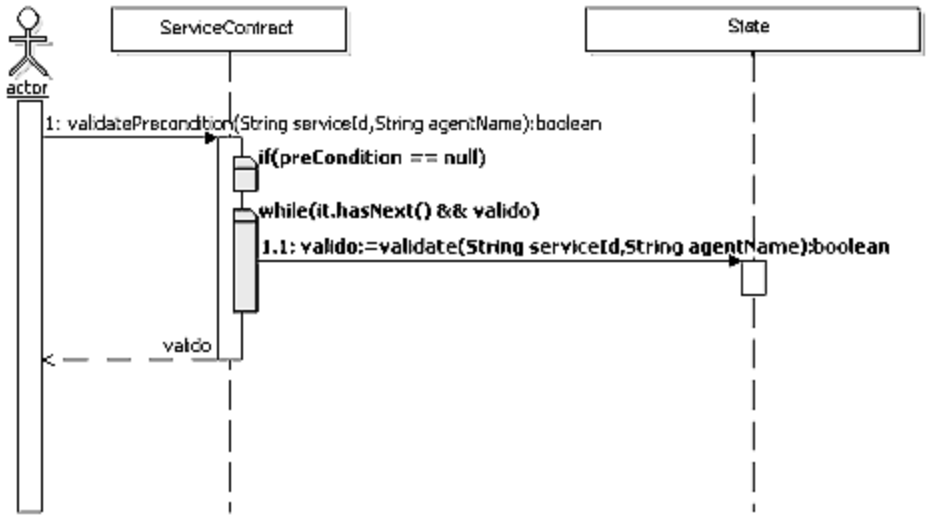

Figura 26: Diagrama de seqüência do contrato de serviço.

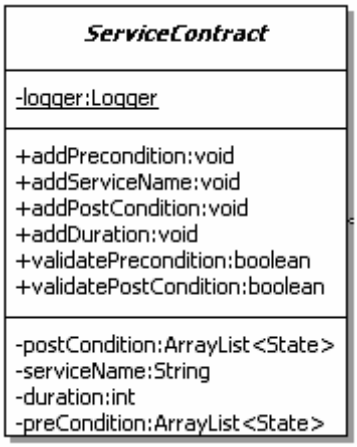

Figura 27: Contrato de Serviço.

4.1.4.2 Contrato de Componente

Contrato de componentes descreve as individualizações dos agentes, definindo os serviços oferecidos pelos agentes ao entrarem em uma organização. A classe abstrata *ComponetContract* possui uma lista de contratos de serviços, representando os serviços assumidos pelos agentes:

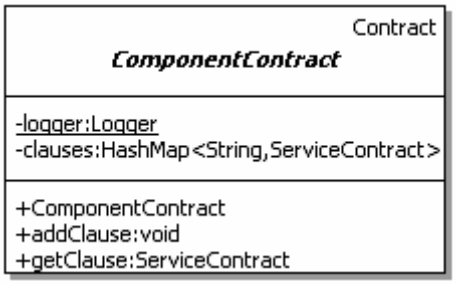

Figura 28: Contrato de Componente.

4.1.4.3 Contrato de Assembly

O contrato de *assembly* especifica parcerias entre agentes, conectando agentes requisitantes e provedores de serviços. A classe *Partner* especifica a ligação entre agentes parceiros na execução de um serviço, através dos atributos *executor*, *client* e *serviceName.* A classe abstrata *AssemblyContract* representa o contrato de ligação, contendo como atributo uma lista de parceiros *(partners).*

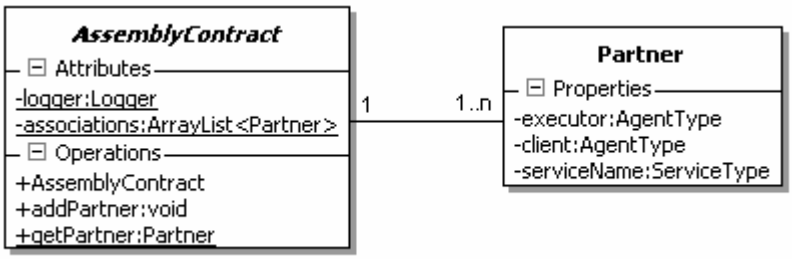

Figura 29: Contrato de Assembly.

4.1.4.4 Contrato de Composição

Contratos de composição encapsulam um grupo de agentes e, similarmente a um agente individual, podem prover e requerer serviços. A composição permite definir internamente um contrato de ligação entre seus agentes e o *singleton ContractController* registra todas as composições que compõem o sistema.

A classe *ExportedService* representa os serviços exportados por uma composição, conectando o tipo do serviço a ser exportado ao agente responsável por sua execução. Seus atributos são:

- *AgentType:* objeto que descreve o tipo de um agente. *AgentType* possui operações que permitem recuperar o agente, seu nome local e a composição a qual pertencem.
- *ServiceType:* objeto que descreve o tipo de um serviço. *ServiceType* possui operações que permitem recuperar a implementação do contrato de serviço, e fazer comparações entre serviços, ou seja, verificar se um serviço é uma especialização ou não de outro serviço.

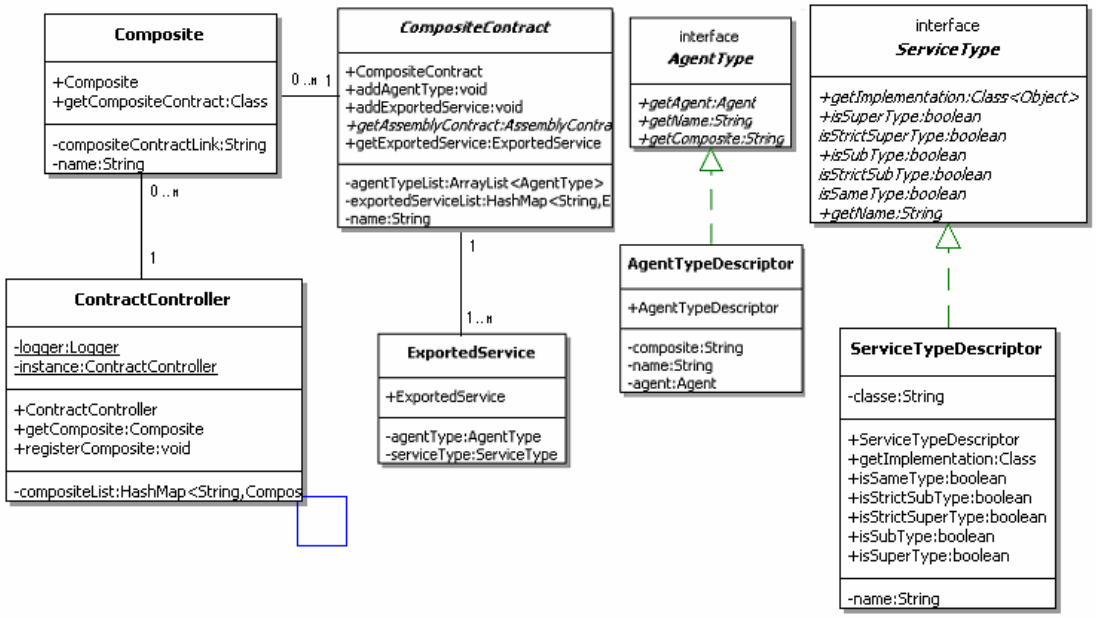

Figura 30: Esquema de contrato de composição.

4.1.4.5 Mediator Agent

Uma das informações acessíveis sobre os agentes é o seu comportamento observável através da troca de mensagens. Assim, o gerenciamento de contratos deve interceptar estas mensagens, analisá-las e verificar se os agentes estão cumprindo o contrato assumido dentro da organização. O mecanismo de contratos necessita, portanto, de um mediador entre os agentes, capaz de interceptar todas as mensagens antes de direcioná-las para os agentes destinatários da mensagem.

O mecanismo de contratos possui um ciclo de vida constituído por quarto macro-atividades: (i) interceptação; (ii) atualização da máquina de estados do

agente; (iii) redirecionamento; (iv) verificação da máquina de estados do agente em busca de contratos não cumpridos.

Na primeira atividade, o mecanismo intercepta as mensagens trocadas entre os agentes. Na segunda fase, o mecanismo atualiza a máquina de estados do agente, levando em consideração o protocolo do agente e o estado corrente do serviço sendo prestado. Na terceira fase, a mensagem é direcionada para aos destinatários reais. Por fim, é necessário que se verifique a máquina de estados dos agentes, listando os serviços em estado corrente e verificando se estão de acordo com o contrato assumido pelo agente.

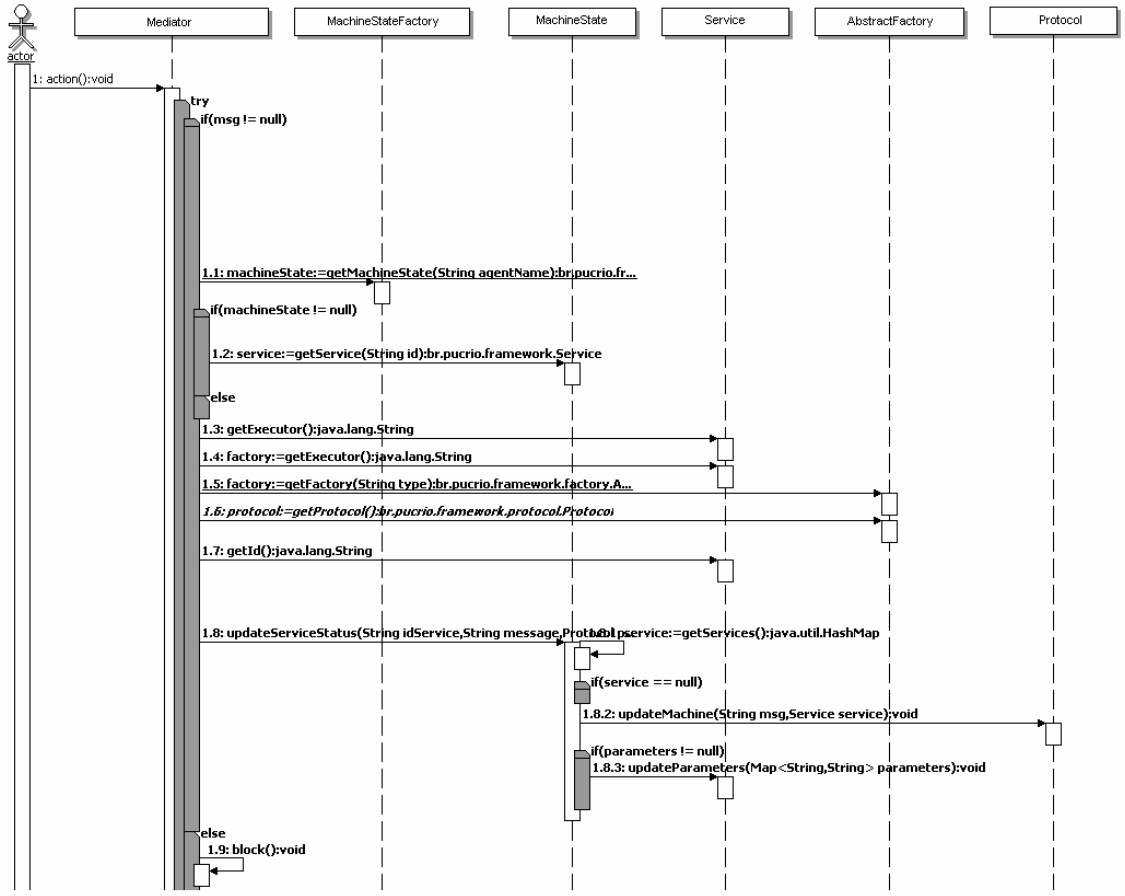

Figura 31: Diagrama de seqüência do agente mediador.

4.1.4.6 Contract Manager Agent

Uma vez atualizada a máquina de estados dos agentes através da interceptação de mensagens pelo mediador, é necessário que se verifique se os serviços estão sendo executados de acordo com os contratos assumidos pelos agentes. Assim, periodicamente, o agente gerenciador de contratos

(*ContractManager)* acessa a máquina de estados dos agentes e percorre sua lista de serviços. Para cada serviço em andamento, o gerenciador recupera o contrato de serviços correspondente e, caso o limite máximo para a execução do serviço tenha expirado, o gerenciador irá verificar se a pós-condição do serviço foi cumprida.

O agente gerenciador de contratos produz um relatório sobre o andamento dos serviços, identificando o agente cliente, o executor, o serviço e se o contrato foi ou não cumprido. Caso não tenha sido, a condição que foi ferida. A seguir é mostrado um exemplo de saída gerada pelo gerenciador de contratos:

```
[contractManager] (CheckContractBehaviour.java:85) - --------------------
[contractManager] (CheckContractBehaviour.java:86) - Agente = montador1
[contractManager] (CheckContractBehaviour.java:88) - Composição = ProductionCenter
[contractManager] (CheckContractBehaviour.java:90) - Servico ID= 1593828667
[contractManager] (CheckContractBehaviour.java:91) - Servico Para = vendedor1
[contractManager] (CheckContractBehaviour.java:92) - Servico de Nome = mount-service
[contractManager] (CheckContractBehaviour.java:93) - Verificando post-condition
[contractManager] (CheckContractBehaviour.java:98) - Agente cumpriu o contrato
[contractManager] (CheckContractBehaviour.java:85) - --------------------
```

Figura 32: Exemplo de saída de um relatório.

4.1.5 Módulo Gerenciamento de Serviços

O Gerenciamento de serviços tem como objetivo a alocação de serviços entre os agentes. Contratos de *assembly*, conforme já descrito, definem parcerias entre agentes que podem ser consultadas ao se alocar um serviço entre um agente requisitante e um agente executor. Embora exista o contrato de *assembly*, a obrigatoriedade de se alocar serviços respeitando este contrato não foi implementada, deixando a estratégia de alocação de serviços como um ponto flexível do *framework*.

4.1.5.1 Service Manager Agent

O agente gerenciador de serviços é responsável pela alocação dos serviços entre os agentes. Este agente intercepta mensagens cuja *conversationId (*maiores informações sobre os padrões de comunicação entre os agentes está descrito no protocolo (Fipa,2003)) seja o evento *"new_service-event".* A estratégia de alocação do serviço, para efeito de demonstração, foi implementada conforme abaixo:

- Interceptar mensagens cuja *conversationId* seja o evento *"new_service-event"*
- Verificar se o agente requisitante do serviço participa de alguma composição

- Caso verdadeiro, o agente só poderá requisitar serviços de dentro da composição
  - Verificar o contrato de *assembly* da composição, buscando por parcerias entre o agente requisitante do serviço e um possível agente executor
  - Encontrado um contrato de *assembly* para o serviço requisitado, alocar o serviço para o agente executor
    - Caso não exista um contrato de *assembly* para o serviço requerido, verificar se algum agente da composição pode suprir a demanda pelo serviço
- Caso negativo, verificar se o agente requisitante do serviço possui algum contrato de *assembly* para o serviço especificado
  - Caso encontre um contrato de *assembly*, alocar o serviço para o agente executor
  - Caso não exista um contrato de *assembly* para o serviço requerido, verificar se existe alguma composição que exporte o serviço
    - Encontrado uma composição que exporte o serviço, alocar o serviço para esta composição
    - Caso não encontre uma composição que exporte o serviço, procurar no pool de agentes do sistema algum que forneça o serviço especificado e alocá-lo como agente executor.