4
A Solução Proposta (Shine)

Observando o problema apresentado na seção anterior, o objetivo desta dissertação é aplicar a abordagem de sistemas multi-agentes neste domínio de manipulação de dados gerados a partir da infra-estrutura distribuída de RFID. Para isso, propõe-se prover um *framework* para a instanciação de sistemas capazes de interagir com uma infra-estrutura RFID distribuída, provendo uma arquitetura que ajuda na criação de novas funcionalidades decorrentes do leque de aplicações vislumbradas pelo surgimento de RFID.

Para fins de nomenclatura e identificação desta solução, o nome **Shine** está sendo atribuído ao *framework* proposto.

4.1.
Requisitos de Alto Nível

Alguns requisitos de alto nível podem ser destacados para a solução proposta:

- Devido à natureza distribuída das leitoras RFID e dos *EPC middlewares*, ao grande volume de dados gerados pela infra-estrutura RFID, e ao fato de não se desejar ter um único ponto de falha no sistema (*Single Point of Failure*), o primeiro requisito encontrado é que a solução aqui proposta deve contemplar **distribuição**.
- Devido à possibilidade de existência de diversas implementações do *EPC middleware* e da camada ALE, é importante que a implementação do *framework* seja **independente da implementação da ALE**, dependente apenas da interface especificada no padrão ALE.
- Para facilitar a implementação de novas funcionalidades, a solução deve ter **flexibilidade na forma de adicionar essas novas funcionalidades**, não devendo ser necessária a modificação de classes do *framework*.

PUC-Rio - Certificação Digital Nº 0310882/CB

- **Tarefas** a serem executadas pelas aplicações instanciadas a partir do *framework* devem poder ser **independentes entre si**, podendo ser executadas paralelamente. Ao mesmo tempo, caso seja de interesse do desenvolvedor da aplicação, deve haver uma forma de determinar dependência entre as tarefas.

Com os requisitos acima preenchidos, pretende-se mostrar que este *framework* pode facilitar o desenvolvimento de aplicações dentro do domínio de aplicações RFID, tais como:

- Visibilidade de inventário em tempo real;
- Alertas para baixo nível de inventário;
- Checagem automática de recebimento;
- Geração automática de Notificação de Recebimento;
- Controle de regras de estocagem;
- Rastreabilidade de produtos;
- Checagem automática de despacho;
- Geração automática de ASN (*Advanced Shipping Notice*);

Existem muitos sistemas e *frameworks* multi-agentes [1] [2] [9] [21]. Uma característica que difere o Shine dos demais é a simplicidade na implementação e as abstrações usadas no Shine em comparação com as demais usadas em outros *frameworks*. Tendo o objetivo de avaliar o uso de sistemas multi-agentes no domínio de RFID, buscou-se criar um *framework* simples, que facilitasse a execução de agentes distribuídos com capacidade de comunicação ente si, que por sua vez executassem tarefas a partir dos objetivos representados pela mensagem recebida. Abstrações e conceitos como organizações, papéis, planos, mobilidade, aprendizado, entre outros, que são característicos de outros sistemas e *frameworks* multi-agentes não estão presentes no Shine. Assim, obteve-se uma solução simples, voltada para a instanciação de sistemas multi-agentes simples, em que a **distribuição, autonomia, interação e colaboração são os principais conceitos de sistemas multi-agentes presentes nesta solução**. Com este *framework* simples, foi possível criar uma instância que prova o conceito de aplicação de um sistema multi-agente para o domínio de RFID.

PUC-Rio - Certificação Digital Nº 0310882/CB

4.2. Arquitetura do Domínio

Para facilitar a visualização do contexto no qual esta solução está inserida, a Figura 1 apresenta uma visão de alto nível dos componentes participantes de uma solução genérica do domínio de RFID, na qual o Shine está inserido.

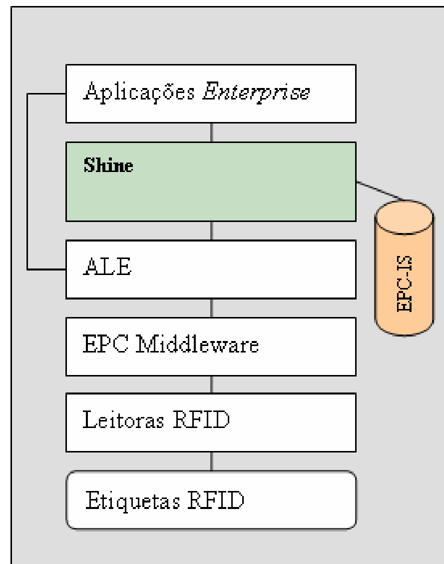

Figura 1: arquitetura do domínio de RFID

Nesta figura, vê-se que a aplicação Shine é responsável por realizar a interface com a camada ALE, apesar de, pela especificação ALE, não haver imposição para que as próprias aplicações Enterprise não possam interagir diretamente com ela – por isso há um relacionamento entre essas aplicações e a camada ALE na arquitetura da Figura 1. Como apresentado na seção 2.3.4, a camada ALE responde a requisições das aplicações cliente transmitindo relatórios contendo dados EPC processados em tempo real. Os relatórios gerados por ela não possuem nenhuma semântica de negócios, possuem apenas dados filtrados e consolidados de fontes de dados EPC de baixo nível, como as leitoras RFID.

Portanto, as aplicações criadas a partir da instanciação do *framework* Shine devem realizar estas requisições de relatórios e devem processar esses relatórios. A inferência da existência de eventos de negócio, tais como, recebimento de carregamentos, checagem do carregamento, ausência de produto em estoque, detecção de furto, etc., é responsabilidade das aplicações, e não do *framework* Shine.

PUC-Rio - Certificação Digital Nº 0310882/CB

Em seu mais alto nível, a instância do Shine poderia interagir com aplicações *enterprise* ou implementar, ela mesma, regras de negócios específicas, como as citadas acima.

4.3.
Metodologia Proposta

Um armazém utilizando a tecnologia RFID/EPC, por exemplo, pode ter diversas leitoras espalhadas, além de poder ter diversas instâncias de *EPC middleware* e ALE sendo executadas. Com o objetivo de obter escalabilidade, desempenho na comunicação com a camada ALE, é importante que o Shine seja uma aplicação distribuída, de preferência tendo cada instância da camada ALE um nó associado da aplicação Shine instalado no mesmo hardware (Figura 2). Esta premissa de instalação num mesmo hardware da instância de ALE e do nó Shine associado se deve ao fato de desempenho de comunicação entre esses dois softwares, além do fato de, inicialmente, facilitar a implementação da comunicação entre essas duas aplicações. Mas nada impede que posteriormente, elas estejam em máquinas diferentes se comunicando através de algum protocolo remoto.

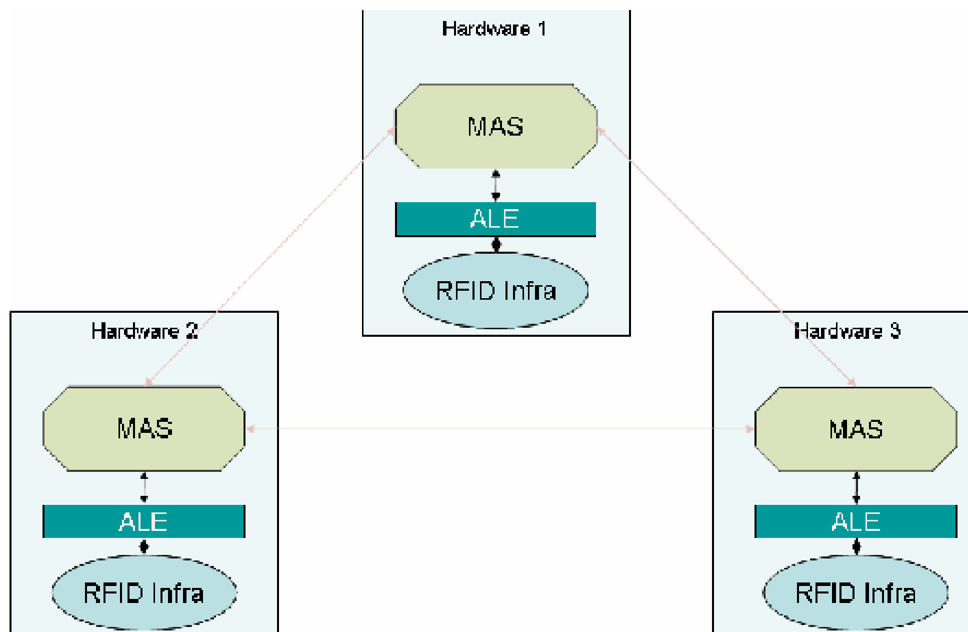

Figura 2: diagrama da metodologia proposta

PUC-Rio - Certificação Digital Nº 0310882/CB

O Shine recebe como entrada relatórios provindos da ALE, a partir destes relatórios, é preciso realizar algum processamento de acordo com as regras associadas às leitoras geradoras dos relatórios. Portanto, é preciso configurar no Shine as leitoras existentes. Na verdade, como citado na seção 2.3.4, é preciso configurar as leitoras lógicas apenas, deixando a responsabilidade para a camada ALE de saber mapear uma leitora lógica em uma ou mais leitoras físicas.

É preciso ainda determinar os diversos nós do Shine existentes que fazem a interface com as instâncias do ALE e coordená-los para que façam as requisições à camada ALE nos momentos devidos. Estes, recebendo a "ordem" de requisitar relatórios, devem fazê-lo e processar o relatório respectivo de acordo com as regras associadas às leitoras presentes nos relatórios. Podem existir casos em que, para se completar o objetivo final do processamento de tal regra, seja preciso que esses diferentes nós do Shine se comuniquem entre si a fim de, por exemplo, se ter a informação precisa da localização de determinado objeto (etiqueta RFID/EPC), ou de se gerar um relatório de inventário em tempo real. Esta comunicação é simbolizada no relacionamento entre os diferentes nós do Shine na Figura 2.

PUC-Rio - Certificação Digital Nº 0310882/CB

4.4. Requisitos do *Framework* Shine

Esta seção apresenta os requisitos do Shine em forma de casos de uso. Ao iniciar o projeto das duas aplicações de estudo de caso, detalhadas no capítulo 5, percebeu-se que elas possuíam dois conceitos importantes em comum:

- elas eram sistemas multi-agentes distribuídos;
- e elas interagiam com a infra-estrutura RFID através da implementação da especificação ALE.

Percebendo que outras aplicações com a mesma natureza também teriam esses dois fatos em comum, buscou-se criar um *framework* simples para sistemas multi-agentes distribuídos que interagisse com a infra-estrutura RFID.

Observando que o resultado final das aplicações apresentava implementações em comum e outras diferenciadas, específicas de cada caso, e ainda observando que parte do que era comum era voltado para aplicações de sistemas multi-agentes distribuídos, de maneira abrangente (não apenas para sistemas com interação com a infra-estrutura RFID), decidiu-se isolar o que era

framework para sistemas multi-agentes da parte que implementa a interação com a infra-estrutura RFID.

Assim, o *framework* Shine final é composto por dois módulos:

- Módulo ShineFSMAD (Shine - *Framework* de Sistemas Multi-Agentes Distribuídos): este módulo é responsável pela infra-estrutura de sistemas multi-agentes. Ele é um *framework* por si só e provê a possibilidade de geração de sistemas multi-agentes distribuídos.
- Módulo ShineALE: este módulo estende o módulo descrito acima, adicionando agentes que possuem a capacidade de interagir com a infra-estrutura RFID através da interface ALE.

Por conta desta separação em dois módulos, a partir deste momento os módulos sempre serão referenciados, desde os requisitos até a solução de implementação, que são separadas em arquivos do tipo JAR [22].

A Figura 3 mostra o *framework* Shine composto por seus dois módulos. Pode-se perceber pela figura que o módulo ShineALE estende o módulo ShineFSMAD. Isso é feito através da extensão de classes abstratas presentes no módulo ShineFSMAD.

O módulo ShineFSMAD foi projetado para poder ser utilizado separadamente do restante da solução geral do Shine. Ele pode ser usado para a geração de sistemas multi-agentes distribuídos. Estes sistemas serão simples no sentido dos agentes possuírem um número limitado de características de agentes. Apenas autonomia, distribuição, interação e colaboração estão presentes na solução deste módulo. Características como objetivos, planos, organizações, conhecimento, mobilidade, entre outras, não foram implementadas. Porém, o módulo se tornou extremamente simples de ser instanciado e tem grande utilidade para a prova de conceitos de aplicações multi-agentes.

Já o módulo ShineALE estende explicitamente o módulo ShineFSMAD, consequentemente depende inteiramente do mesmo, e não pode ser usado separadamente.

PUC-Rio - Certificação Digital Nº 0310882/CB

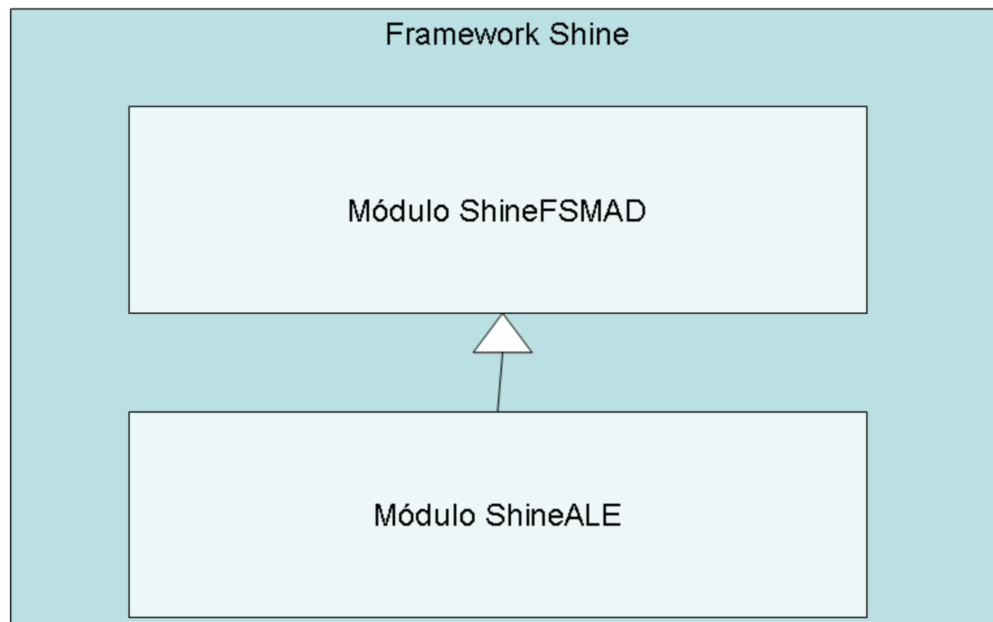

Figura 3: módulos do *framework* Shine

PUC-Rio - Certificação Digital Nº 0310882/CB

4.4.1. Casos de Uso do módulo ShineFSMAD

A Figura 4 mostra o diagrama dos principais casos de uso para agentes deste módulo.

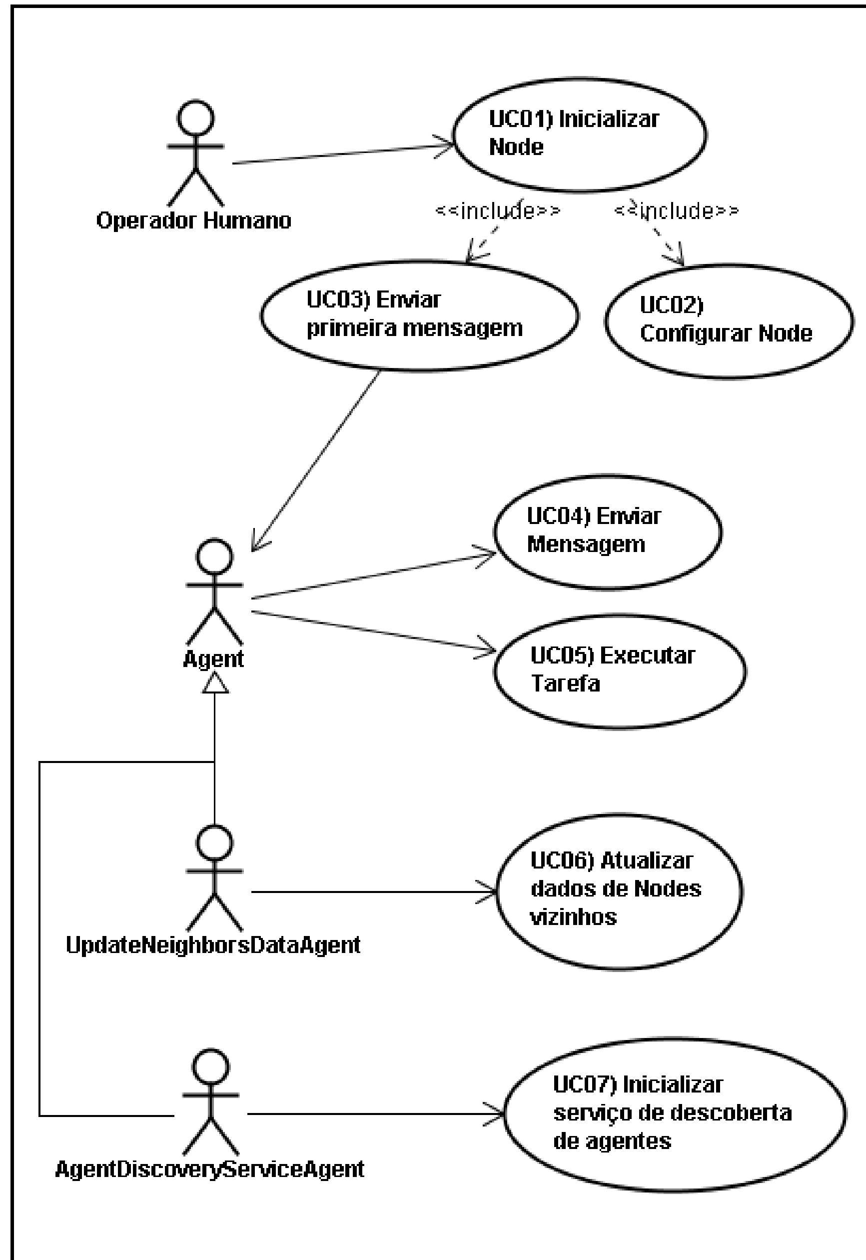

Figura 4: casos de uso dos agentes do módulo ShineFSMAD

PUC-Rio - Certificação Digital Nº 0310882/CB

Descrição dos Casos de Uso:

UC01) Inicializar `Node`

- Descrição: um operador humano (nomeou-se operador humano apenas para se diferenciar atores humanos de atores agentes de software) deve inicializar o sistema através do executável que inicializa um `Node`. Este operador deve determinar qual `Node` está

sendo inicializado, para que, assim, o Shine possa utilizar o arquivo de configuração específico do `Node` em questão.

- Pré-condições: o usuário deve selecionar o arquivo XML de configuração correto do `Node` em questão.
- Fluxo de Eventos:
  - Usuário executa o executável da aplicação do Shine.
  - Ele seleciona o arquivo XML específico do `Node` em questão.
  - O Shine executa o caso de uso 2: Configurar `Node`.
  - O Shine executa o caso de uso 3: Enviar primeira mensagem.

UC02) Configurar `Node`

- Descrição: a primeira ação a ser tomada por um `Node` de uma aplicação Shine é instanciar os objetos de seus agentes corretamente, atribuindo-lhes uma `Thread` independente e fornecendo-lhes as tarefas (`Task`s) que eles poderão executar a partir de mensagens recebidas.
- Pré-condições: não há.
- Fluxo de Eventos:
  - O Shine executa a configuração do Node, instanciando os objetos de agentes e fornecendo-lhes uma `Thread` independente para suas execuções.
  - Com as descrições das tarefas que cada agente pode executar presentes no arquivo de configuração, o Shine instancia estas ações, agrupa-as em uma coleção de ações do agente e atribui esta coleção à coleção de tarefas do agente.

UC03) Enviar primeira mensagem

- Descrição: para que o sistema possa ser inicializado de fato, os agentes precisam iniciar as execuções de suas tarefas. Alguns executam tarefas após receberem mensagens de outros agentes. Porém, algum deve iniciar suas atividades assim que o sistema inicializa. O mecanismo criado para que estes agentes iniciem suas atividades é fazer com que o Node, após configurado e inicializado

PUC-Rio - Certificação Digital Nº 0310882/CB

corretamente, envie uma primeira mensagem para todos os seus agentes. Aqueles que devem iniciar suas atividades neste momento podem fazê-lo. Os demais simplesmente ignoram esta primeira mensagem.

- Pré-condições: não há.
- Fluxo de Eventos:
  - Ao término da configuração do `Node`, este envia mensagem para todos os seus agentes. Esta mensagem deve ter a performativa "`firstMsg`".

UC04) Enviar mensagem

- Descrição: os agentes podem interagir entre si. Para isso, o agente deve enviar uma mensagem a outro(s) agente(s).
- Pré-condições: no momento de se enviar uma mensagem o agente deve conhecer o identificador (id) dos agentes destinatários da mensagem.
- Fluxo de Eventos:
  - O agente cria uma nova mensagem, e a configura com sua performativa, destinatários e conteúdo.
  - Envia mensagem.

UC05) Executar tarefa

- Descrição: no módulo ShineFSMAD, o principal objetivo de um agente é executar tarefas a partir do recebimento de mensagens. Quando se recebe uma mensagem, a mensagem contém uma performativa. A partir desta performativa, o agente deve executar tarefas específicas para a mesma.
- Pré-condições: recebimento de uma mensagem com uma performativa. A relação entre a performativa da mensagem e a(s) tarefa(s) a ser(em) executada(s) é determinada no arquivo de configuração do Node (Tabela 1: arquivo XML de configuração de Node).
- Fluxo de Eventos:
  - Agente recebe mensagem.

PUC-Rio - Certificação Digital Nº 0310882/CB

  - Agente obtém performativa da mensagem e recupera em sua lista de tarefas aquelas relacionadas com a performativa em questão.
  - Agente executa as tarefas recuperadas, caso alguma tenha sido encontrada.

UC06) Atualizar dados de `Node`s vizinhos

- Descrição: Quando um `Node` é inicializado, sua configuração XML determinada se ele possui vizinhos ou não. Caso possua, este `Node` deve tentar enviar mensagem para o `Node` vizinho para que este atualize sua lista de vizinhos, acrescentando-o.
- Pré-condições: para que este caso de uso seja executado é preciso que a configuração XML possua a configuração necessária para definição de vizinho(s). Além disso, para que se concretize, é preciso que o `Node` consiga entrar em contato com seu vizinho.
- Fluxo de Eventos:
  - O Node é inicializado.
  - Durante a inicialização, o configurador (`ConfigManager`) verifica a existência de configuração de vizinhos.

UC07) Inicializar serviço de descoberta de agentes

- Descrição: ao inicializar a aplicação, é preciso inicializar o serviço de descoberta de agentes. Este serviço será usado para encontrar agentes pertencentes a outros Nodes.
- Pré-condições: não há.
- Fluxo de Eventos:
  - O Node é inicializado.
  - Durante a inicialização o serviço de descoberta de agentes é inicializado.

4.4.2. Casos de Uso do Módulo ShineALE

A Figure 5 apresenta o digrama de casos de uso deste módulo.

PUC-Rio - Certificação Digital Nº 0310882/CB

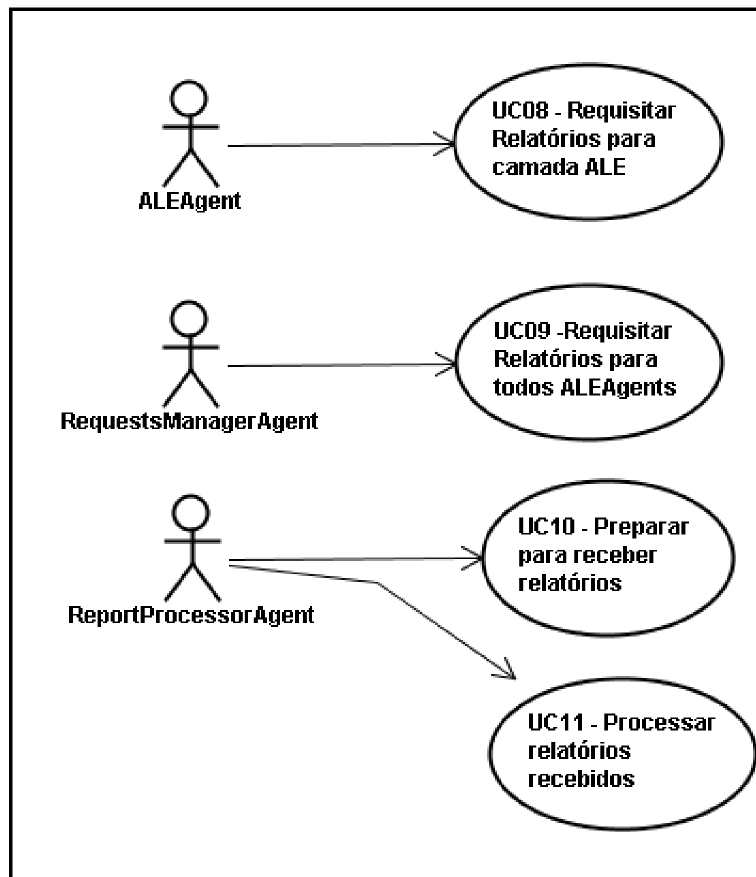

Figure 5: diagrama de casos de uso do módulo ShineALE

PUC-Rio - Certificação Digital Nº 0310882/CB

Descrição dos Casos de Uso:

UC08) Requisitar relatórios para camada ALE

- Descrição: este caso de uso tem o objetivo de requisitar relatórios de ciclos de evento para a devida implementação da especificação ALE.
- Precondições: para a requisição de relatórios acontecer, é preciso receber mensagem cuja performativa está associada à tarefa de requisição de relatórios.
- Fluxo de Eventos:
    - Recebimento de mensagem para requisição de relatórios;
    - Execução da requisição de relatório e recebimento do mesmo.
    - Envio do relatório para o devido destinatário.

UC09) Requisitar relatórios para todos ALEAgents

- Descrição: os agentes que requisitam relatórios para as implementações da especificação ALE são os chamados ALEAgents. Para que eles possam requisitar os relatórios para seus devidos ALEs, como descrito no caso de uso acima, eles precisam receber mensagem ordenando tal ação. Este caso de uso tem a responsabilidade de enviar esta mensagem de ordem: requisitar a todos os ALEAgents para estes requisitem relatórios aos seus respectivos ALEs.
- Precondições: não há.
- Fluxo de Eventos:
    - As aplicações instâncias deste *framework* em algum momento precisarão de relatórios. Quando isso acontecer, elas devem enviar mensagem para o agente responsável pela execução deste caso de uso.
    - O agente responsável pela execução deste caso de uso envia mensagem para todos os ALEAgents para que estes possam requisitar os relatórios para seus respectivos ALEs.
    - Este mesmo agente adiciona o agente responsável pelo processamento dos relatórios como destinatário da mensagem citada acima.

UC10) Preparar para receber relatórios

- Descrição: o agente responsável por processar relatórios provindos de ALEAgents precisa ser notificado com a informação de que ALEAgents estão sendo requisitados para gerarem relatórios, como descrito nos casos de uso anteriores. Com esta informação, o agente processador de relatórios saberá exatamente quais os ALEAgents deverão enviar-lhe relatórios, e pode, assim, esperar por cada um desses relatórios a fim de processá-los.
- Precondições: receber mensagem contendo informação de quais ALEAgents estão sendo requisitados para gerar relatórios.
- Fluxo de Eventos:
    - Recebimento de mensagem contendo informação de quais ALEAgents estão sendo requisitados para gerar relatório.
    - Armazenar essa informação para posteriormente poder saber se todos os relatórios esperados já foram recebidos ou não.

PUC-Rio - Certificação Digital Nº 0310882/CB

UC11) Processar relatórios recebidos

- Descrição: após receber cada um dos relatórios esperados, como descrito no caso de uso acima, o agente processador de relatórios deve processá-los segundo regras da aplicação em questão.
- Precondições: receber todos os relatórios esperados.
- Fluxo de Eventos:
    - Recebimento das mensagens de cada um dos ALEAgents.
    - Após receber todos os relatórios esperados, deve-se processar os relatórios segundo regras específicas da aplicação em questão.

A seção 4.5 seguinte apresenta a arquitetura criada para contemplar os requisitos apresentados acima.

4.5.
Arquitetura do *Framework* Shine

Esta seção apresenta a arquitetura do *framework* Shine, mostrando as diferentes camadas e suas responsabilidades. A Figura 6 apresenta uma visão desta arquitetura.

PUC-Rio - Certificação Digital Nº 0310882/CB

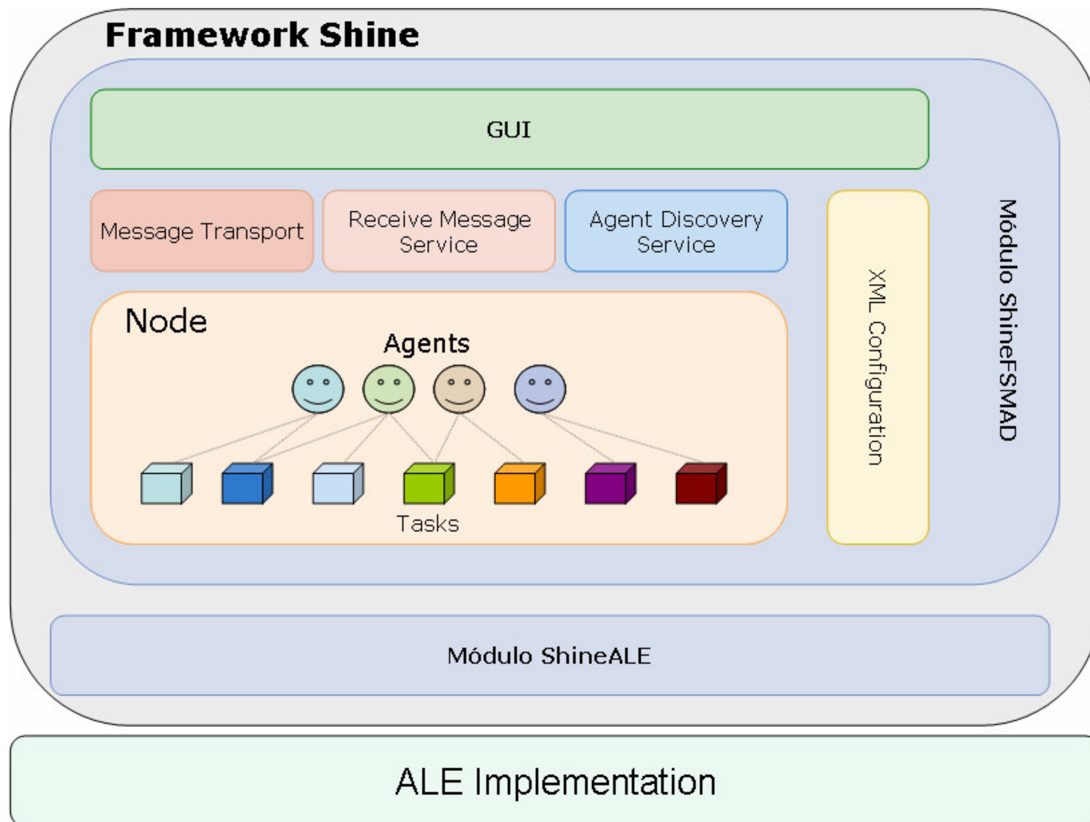

Figura 6: arquitetura do *framework* Shine

As subseções seguintes apresentam com mais detalhes cada camada da arquitetura apresentada na Figura 6.

4.5.1. ALE Implementation

Como descrito na seção 2.3.4, a camada ALE é uma especificação da EPCglocal Inc. Para atingir o objetivo desta dissertação de mestrado e avaliar a aplicação de sistemas multi-agentes no domínio de RFID foi preciso implementar uma versão da especificação. Esta implementação segue rigorosamente a especificação, e, conseqüentemente, pode ser trocada por implementações de terceiros, que realmente interagem com *EPC middlewares*. Para efeitos de avaliação desta dissertação, a implementação do ALE criada simula a existência da infra-estrutura RFID (*EPC middleware*, leitoras e etiquetas RFID). Pode-se ver na Figura 7 a representação da classe `ALEShineImpl` que implementa a interface `ALE` definida pela especificação da EPCglobal Inc.. Detalhes do que

PUC-Rio - Certificação Digital Nº 0310882/CB

cada operação desta interface deve fazer podem ser vistos na própria especificação [13].

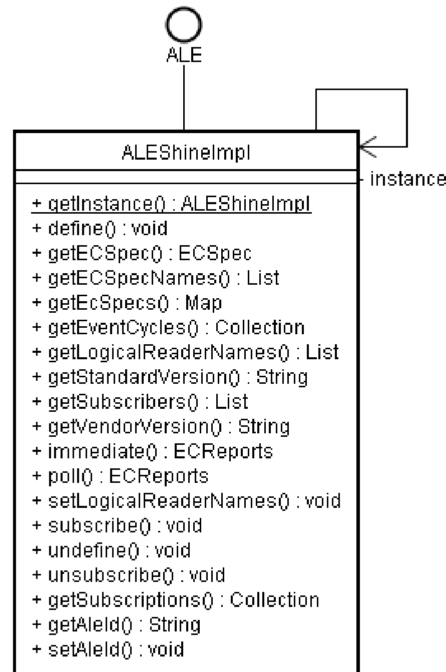

Figura 7: classe que implementa a interface ALE especificada pela EPCglobal Inc.

PUC-Rio - Certificação Digital Nº 0310882/CB

4.5.2.
Módulo ShineALE

A interação com a implementação do ALE se dá através do módulo ShineALE. Esta camada é responsável por conhecer a implementação do ALE e por requisitar a ela relatórios de leituras. Esta requisição é feita através de determinação de qual ECSpec deve ser executado. Com esta requisição e o correspondente ECSpec, a implementação do ALE executa um ciclo de evento, como apresentado na seção 2.3.4. A partir deste ciclo de evento, é gerado um relatório no padrão ECReport, e este é repassado para a ALE Interface, que por sua vez entrega-o para quem fez a requisição do relatório. Maiores detalhes de projeto e implementação, inclusive com diagramas, serão apresentados na seção 4.6.

4.5.3. Módulo ShineFSMAD

Este módulo é responsável por implementar o *framework* de sistemas multi-agentes. Ele pode ser usado independentemente do domínio de RFID, já que é disponibilizado em um arquivo JAR independente. As camadas deste módulo são descritas nas seções a seguir.

4.5.3.1. Node

Sendo uma instância do Shine uma aplicação distribuída com partes compondo o sistema como um todo, pode-se imaginar o sistema como sendo um grafo composto por seus diversos nós. Por conta disso, cada parte do Shine é representada por um nó, chamado aqui de *Node*.

Cada *Node* pode agregar um ou mais agentes, fornecendo-os serviços básicos para sua execução, como serviços para envio e recebimento de mensagens, e serviço de descoberta de agentes, além de ser responsável pelo ciclo de vida de cada agente. Veremos na seção a seguir detalhes destes serviços.

PUC-Rio - Certificação Digital Nº 0310882/CB

4.5.3.2. Serviços do Node

Abaixo, são descritos os serviços desta camada da solução.

1. **Message Transport**

Este é o serviço responsável pela entrega da mensagem entre dois agentes. A solução criada para o Shine tem dependência com uma interface Java que define este serviço. Conseqüentemente tornou-se simples a troca de uma implementação por outra. A implementação padrão do Shine realiza o transporte da mensagem através de chamadas de métodos remotas usando o protocolo RMI (*Remote Method Invocation*) nativo de Java. Porém, caso seja desejável realizar este envio de mensagem via *Web Services*, por exemplo, deve-se implementar a interface do serviço e alterar a configuração do Shine para que use esta

nova implementação. Esta troca de implementação é realizável facilmente por conta do uso de interfaces, e não classes concretas.

2. **Receive Message Service**

Análogo ao *Message Transport*, este serviço de recebimento de mensagem é responsável por receber uma mensagem endereçada a um ou mais agentes e entregá-la aos mesmos. Este também é definido por interfaces, sendo flexível à troca de implementação.

3. **Agent Discovery Service**

Este serviço é fornecido aos agentes de um Node para que estes possam realizar uma busca por determinados agentes participantes do sistema. Havendo necessidade de um agente enviar mensagem para outro(s), o agente pode usar este serviço para descobrir em que Node o destinatário da mensagem se encontra. De maneira geral, este serviço é usado internamente pelo Node para que os serviços de entrega e recebimento de mensagem possam entregar e receber as mensagens para os agentes corretos, em qualquer Node que estejam.

PUC-Rio - Certificação Digital Nº 0310882/CB

4.5.3.3. Configuração XML

Tendo o objetivo de facilitar a instanciação do *framework* Shine, flexibilizar a troca das implementações dos serviços existentes por novas implementações (caso venham a ser implementadas) e ainda facilitar a visualização dos agentes e respectivas tarefas pertencentes a cada Node, foi criada uma configuração de cada Node do Shine via arquivo XML. As Tabela 1 e Tabela 2, presentes na seção 4.6.1.3, apresentam esta configuração com mais detalhes.

4.5.3.4. GUI

Esta camada fornece uma interface gráfica (GUI – Graphical User Interface) para visualização dos agentes e nós (Nodes) da aplicação criada instanciando o Shine. Novas janelas específicas da aplicação podem ser adicionadas facilmente implementando um dos *hot spots* do *framework* para esta camada de interface gráfica.

4.6. Projeto e Implementação do *Framework* Shine

Esta seção apresenta o projeto e detalhes da implementação do *framework* Shine. Procurar-se-á sempre apresentar os detalhes separados pelos dois módulos do *framework*.

A Figura 8 apresenta o diagrama de classes do *framework* Shine. Apenas as principais classes estão representadas para facilitar a visualização e seu entendimento (a camada de interface (GUI) não está representada). As entidades destacadas com sombreado são os *hot spots* do *framework*. Tanto os *hot spots* quanto os *frozen spots* serão detalhados nas seções 4.6.1.1 e 4.6.1.2.

PUC-Rio - Certificação Digital Nº 0310882/CB

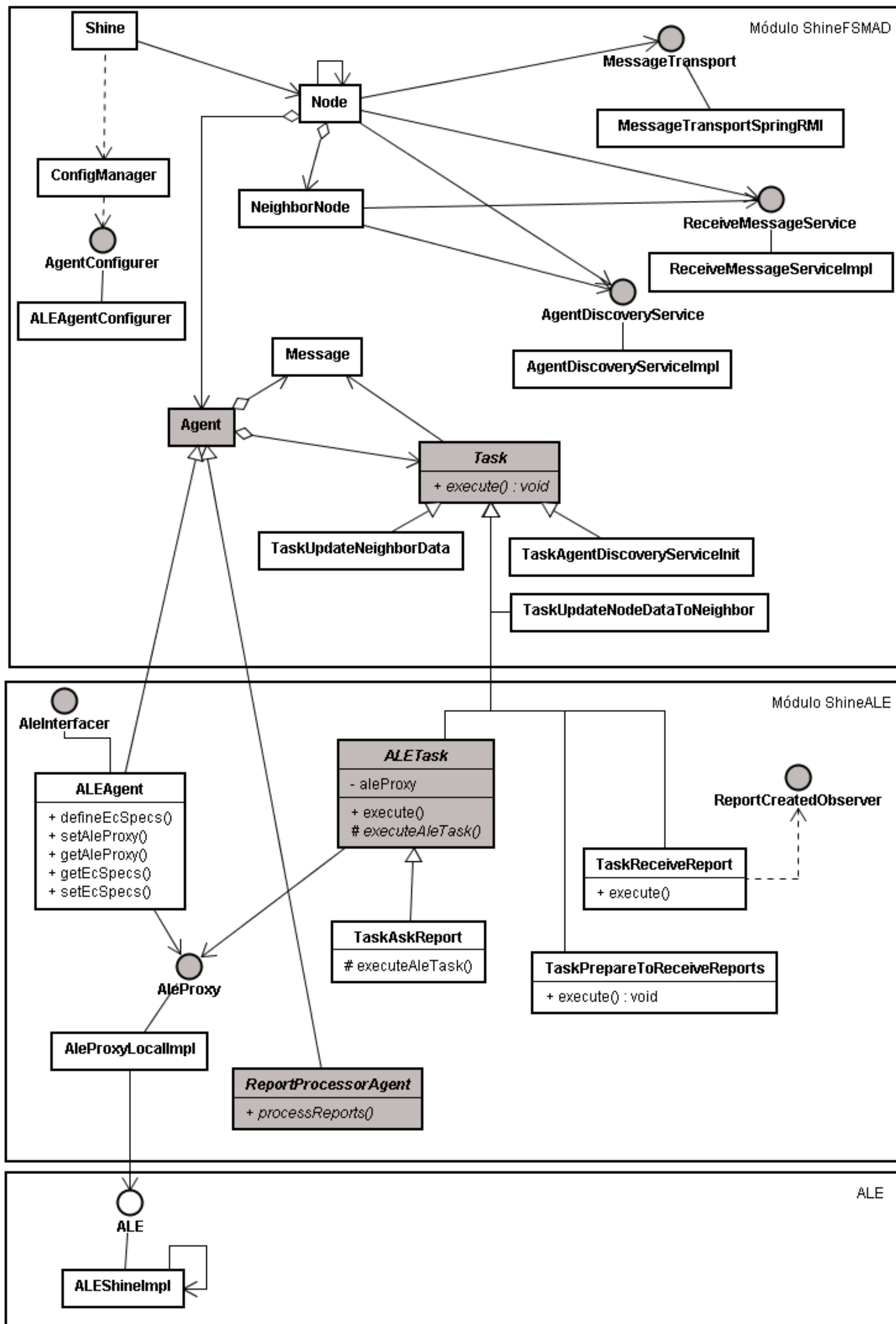

Figura 8: diagrama de classes do *framework* (principais classes)

PUC-Rio - Certificação Digital Nº 0310882/CB

PUC-Rio - Certificação Digital Nº 0310882/CB

4.6.1.
Módulo ShineFSMAD (Shine - *Framework* de Sistemas Multi-Agentes Distribuídos)

Este módulo tem como objetivo ser um *framework* por si só, capaz de gerar aplicações de sistemas multi-agentes distribuídos.

Sistemas multi-agentes, como descrito na seção 2.2, possuem alguns conceitos ou abstrações que são características destes sistemas. Para esta dissertação, procurou-se criar um *framework* multi-agentes simples, que fosse instanciado rapidamente, sem grande estudo ou investigação de como o mesmo funciona. Para isso, buscou-se implementar o *framework* apenas com as seguintes abstrações de sistemas multi-agentes distribuídos: (i) Autonomia; (ii) distribuição; (iii) comunicação entre agentes (interação) e (iv) colaboração.

Sendo assim, a implementação de sistemas multi-agentes distribuídos ficou bastante simplificada com o módulo ShineFSMAD, que por si só é um *framework*. As instâncias deste *framework* (módulo) têm em comum o fato de terem agentes autônomos distribuídos em diferentes nós (Nodes), que têm capacidade de interação entre si e executam tarefas pré-determinadas a partir das mensagens recebidas.

As próximas duas seções apresentam os *frozen* e os *hot spots* do módulo ShineFSMAD.

4.6.1.1.
Frozen Spots

Para um melhor entendimento das partes fixas e flexíveis deste módulo, é interessante pensar na seqüência de execução da inicialização do sistema. Para visualizar melhor esta seqüência, a Figura 9 apresenta o diagrama de seqüências desta execução. Este diagrama é importante, pois já apresenta, além da seqüência de inicialização, a seqüência para o envio de uma mensagem, porém, sem detalhes. Detalhes da seqüência da execução do envio de mensagem podem ser vistos na Figura 11.

A inicialização de um Node do Shine se dá através, inicialmente, da escolha do arquivo de configuração que se quer inicializar (arquivo de configuração do Node). Detalhes do arquivo podem ser vistos na Tabela 1 da seção 4.6.1.3.

Tendo o XML definido, o sistema lê este arquivo e obtém as informações das classes que implementam as interfaces de serviço (MessageTransport,

ReceiveMessageService e AgentDiscoveryService), dos agentes que pertencem ao Node em questão, das tarefas que podem ser executadas pelos agentes e dos Nodes vizinhos. Com estas informações, o sistema instancia essas classes e repassa seus objetos aos devidos métodos das classes do Shine, do Node e dos Agentes. Toda esta configuração é feita pelo ConfigManager (ver Figura 9).

Após configurar o Node, uma primeira mensagem é enviada para todos os agentes do Node. Aqueles que tiverem sido configurados para executar alguma tarefa (Task), ao receber uma mensagem com a performativa desta primeira mensagem, terão suas tarefas executadas.

Podemos citar, então, os seguintes frozen spots do *framework:*

- Configuração de cada *Node* do sistema multi-agente através de arquivos XML, representada pela classe ConfigManager.
- Execução autônoma de cada agente presente em um *Node*. O *Node* cria uma Thread para cada agente, fazendo com que cada um deles se torne independente de outros e possa executar suas tarefas independente de ação de algum operador humano, dependendo apenas das mensagens que o mesmo recebe.
- Envio e recebimento de mensagens entre agentes. Os agentes podem estar no mesmo *Node* ou em *Nodes* diferentes. O *Node* utiliza os serviços de ReceiveMessageService e MessageTransport para receber e enviar mensagens, respectivamente. Apesar dos serviços serem *hot spots*, a forma como eles são utilizados pelo *Node* é fixa.
- Descoberta de agentes a partir de suas identificações (id do agente).
- Execução de tarefas, por parte dos agentes, a partir de performativas das mensagens recebidas.
- Execução das tarefas de forma dependente ou independente umas das outras. As tarefas podem ser configuradas para dependerem de outras tarefas, ou seja, uma tarefa só é executada após o termino de outra. Toda esta configuração é feita nos arquivos XML. Tarefas só podem depender de outras tarefas do mesmo agente. Não pode haver dependência de tarefas que pertencem a agentes diferentes;
- Execução de configuradores específicos de agentes. A classe ConfigManager faz a configuração do *Node* que se está

PUC-Rio - Certificação Digital Nº 0310882/CB

inicializando e executa os configuradores que forem criados para agentes específicos, caso sejam criados. Esta execução se dá utilizando a interface AgentConfigurer, que é um *hot spot*, como descrito abaixo.

PUC-Rio - Certificação Digital Nº 0310882/CB

PUC-Rio - Certificação Digital Nº 0310882/CB

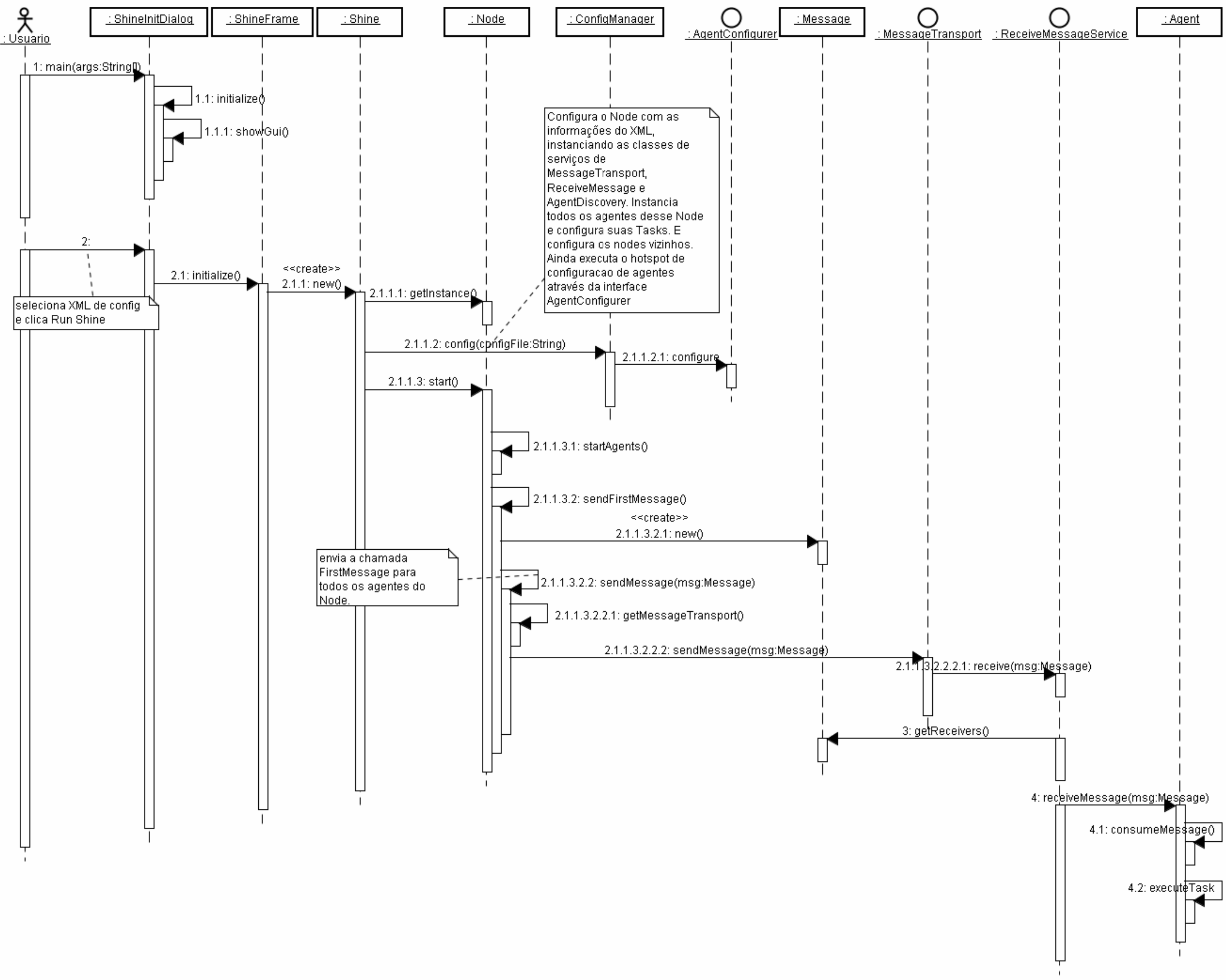

Figura 9: diagrama de seqüências da inicialização do Shine

4.6.1.2. Hot Spots

No mesmo diagrama de seqüências apresentado na Figura 9, pode-se ver algumas interfaces participando da execução da inicialização e troca de mensagem. Estas interfaces são *hot spots* e devem ser implementadas por aplicações que estendem o Shine. Porém, implementações padrão já são fornecidas com o próprio *framework* e podem ser usadas. Além das interfaces de serviços, há mais um *hot spot* neste diagrama que é a classe `Agent`. O agente em si é um *hot spot* e pode ser estendido para prover mais funcionalidades além das já fornecidas pela classe `Agent` do *framework*.

Podemos citar os seguintes *hot spots*:

- Serviço de descoberta de agentes (`AgentDiscoveryService`);
- Serviço de transporte de mensagens (`MessageTransport`);
- Serviço de recebimento de mensagens (`ReceiveMessageService`);
- O agente (`Agent`). A classe `Agent` é um *hot spot*, pois pode e deve ser estendido caso se deseje criar um agente com características mais específicas do que as já presentes na implementação fornecida pelo *framework*. Para casos simples, em que a execução de tarefas e troca de mensagens é tudo que se deseja do agente, a classe `Agent` fornecida com o *framework* é suficiente.
- As tarefas, representadas e especificadas pela classe abstrata `Task`.
- Serviço de configuração de agente (`AgentConfigurer`). Implementações deste *hot spot* têm o objetivo de configurar agentes específicos de aplicações que instanciam o *framework*. Quando a aplicação é inicializada, o configurador da aplicação (`ConfigManager`) é executado e verifica se a configuração XML de cada agente possui uma implementação desta interface. Caso possua, executa a mesma.

PUC-Rio - Certificação Digital Nº 0310882/CB

Para os *hot spots* acima, fora as tarefas (`Task`s), foram criadas implementações padrão, para que os desenvolvedores de aplicações pudessem instanciar o *framework* com foco apenas no comportamento de suas aplicações, não se preocupando com implementação dos serviços básicos. Apesar disso, caso desejem, podem trocar a forma como esses serviços são implementados, que é extremamente simples, sendo necessária apenas a criação da nova classe que implementará o serviço em questão e a configuração correta no arquivo XML das classes que implementam cada serviço. A Figura 10 apresenta o diagrama de classes das implementações padrão do *framework*.

PUC-Rio - Certificação Digital Nº 0310882/CB

PUC-Rio - Certificação Digital Nº 0310882/CB

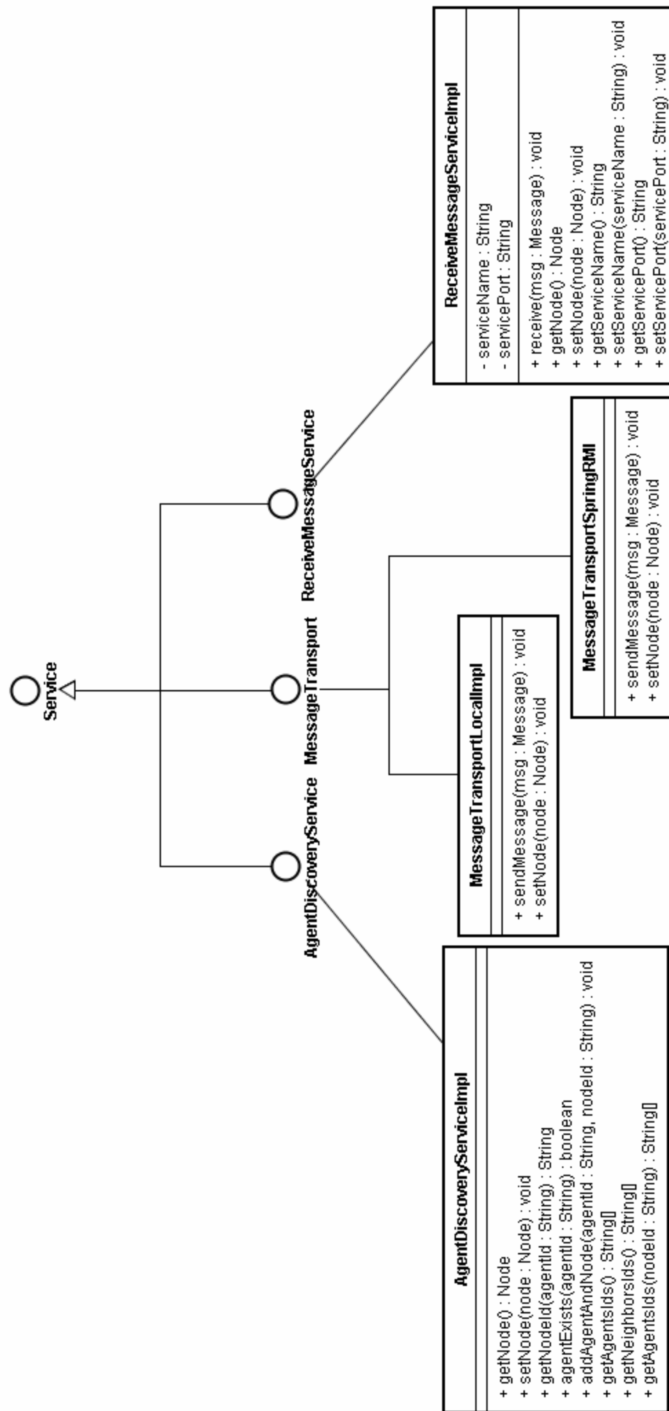

Figura 10: diagrama de classes das implementações padrão dos serviços

Procurou-se implementar este módulo orientado sempre a interfaces, orientada a objetos, facilitando a troca das implementações dessas interfaces.

PUC-Rio - Certificação Digital Nº 0310882/CB

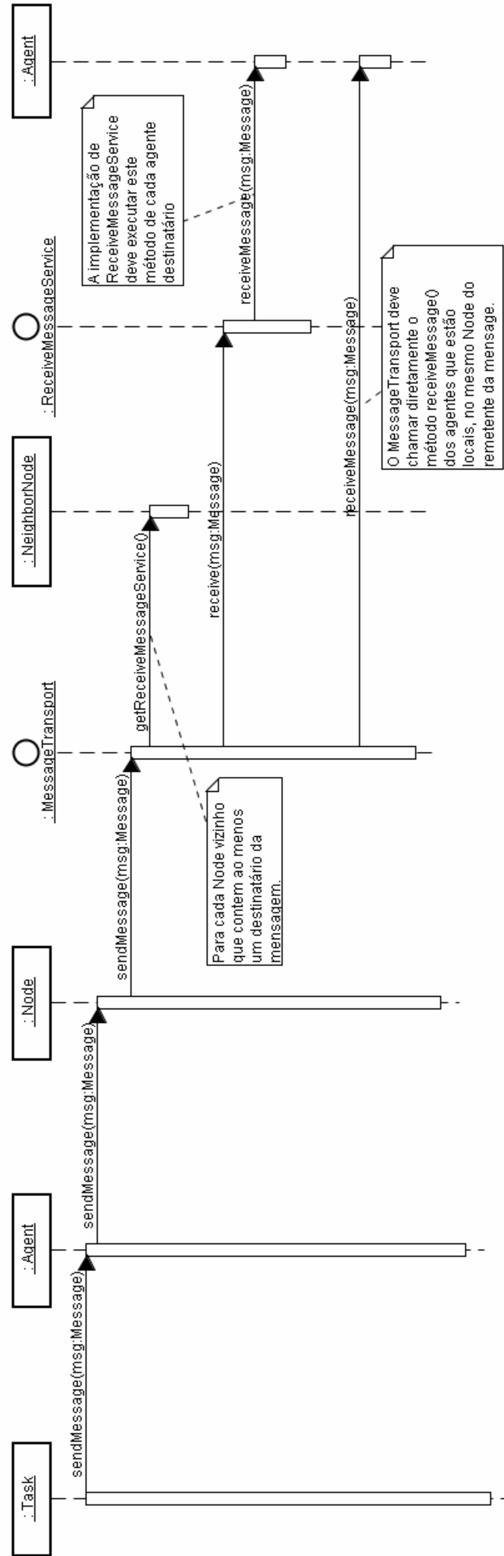

Figura 11: diagrama de seqüências da troca de mensagem entre agentes

No diagrama da Figura 11, omitiu-se alguns detalhes para fins de clareza do diagrama, como por exemplo, a criação do objeto de mensagem (`Message`). Como não há ator neste diagrama (ator externo a aplicação) é provável que surja a dúvida de como esta seqüência se inicia. Porém, esta dúvida pode ser esclarecida no diagrama da Figura 10. Quando o sistema é inicializado, o `Node` envia uma mensagem para todos os agentes pertencentes a ele. Com esta primeira mensagem (possui performativa “firstMsg”), os agentes podem iniciar a execução de suas tarefas. Claramente nem todos os agentes deverão possuir alguma tarefa relacionada com esta primeira mensagem, porém algum agente de algum `Node` da aplicação deve ter pelo menos uma tarefa relacionada com esta mensagem inicial, para assim executar, no mínimo, alguma tarefa que seja objetivo do sistema. Na execução desta primeira tarefa, o agente pode então enviar uma mensagem para outros agentes, como representado no diagrama da Figura 11. E assim o sistema inicia sua execução e pode dar continuidade à execução de tarefas através da troca de mensagens entre agentes.

A Figura 12 apresenta o diagrama de seqüências do caso de uso de execução de tarefas (UC05).

PUC-Rio - Certificação Digital Nº 0310882/CB

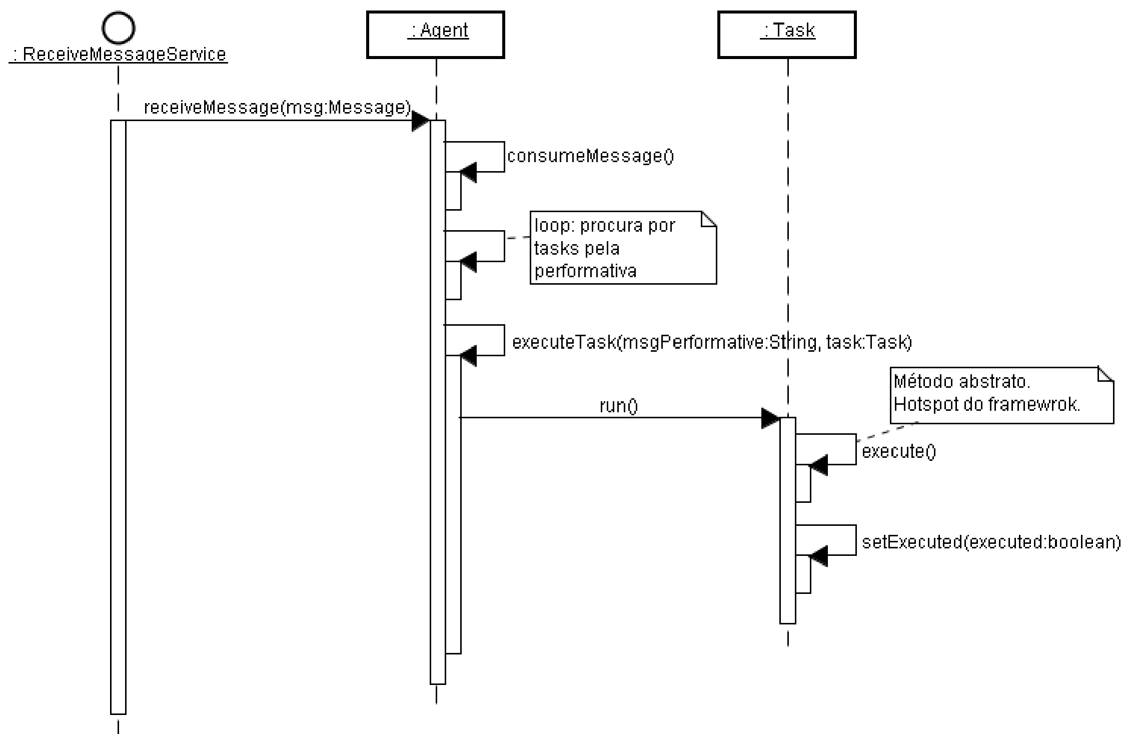

Figura 12: diagrama de seqüências de execução de tarefas

Mais uma vez, neste diagrama pode-se ter dúvidas de como a seqüência se inicia. Observando-se o diagrama da Figura 11, percebe-se que este diagrama da Figura 12 é, na verdade, uma continuação do primeiro. Portanto, a descrição de como a seqüência do diagrama anterior é iniciada é válida para este também.

Até o momento foram apresentados os principais casos de uso deste módulo, assim como o diagrama de classes das principais classes responsáveis por estes casos de uso, assim como seus diagramas de seqüência. Detalhes de implementação foram omitidos com o intuito de clareza dos diagramas e do entendimento do módulo como um todo. Porém, vale ressaltar outros *hot spots* importantes para desenvolvedores que forem utilizar este módulo como um *framework* para seus sistemas multi-agentes.

Utilizando o padrão de projeto Observer [18], este módulo disponibiliza os seguintes observadores como *hot spots,* apresentados na Figura 13. Desenvolvedores que desejarem criar objetos que precisem ser notificados quanto da atualização de estado de um dos objetos observáveis devem implementar as interfaces destacadas na figura e em algum momento adicionar o objeto em questão como observador do objeto observável. O próprio módulo ShineFSMAD implementa um dos *hot spots*, como pode ser visto na figura.

O *hot spot* sendo implementado é referente à interface NeighborAddedObserver. Classes que implementem esta interface serão notificadas quando um novo Node vizinho é adicionado à lista de vizinhos do Node em questão. A classe `TaskUpdateNodeDataToNeighbor`, ao ser notificada da adição de um novo Node na lista de vizinhos, é responsável por enviar mensagem para este novo Node da lista, informando as informações pertinentes deste Node. Por exemplo, digamos que o sistema inicie com a execução de um Node chamado "Node A". Em seguida, um novo Node é adicionado ao sistema, o "Node B". No momento em que o Node B é inicializado, ele enviará mensagem para o Node A informando que ele é um vizinho seu. Neste instante, o Node A adicionará o Node B em sua lista de vizinhos e notificará todas as classes que implementam a interface `NeighborAddedObserver`, inclusive a tarefa `TaskUpdateNodeDataToNeighbor`. Esta, por sua vez, sendo executada por um agente do Node A, enviará mensagem para um agente do Node B informando detalhes do Node A, como por exemplo, informando todos os agentes pertencentes ao Node A. Assim, o Node B saberá encontrar esses agentes no Node correto, o Node A, caso precise enviar mensagens para eles.

PUC-Rio - Certificação Digital Nº 0310882/CB

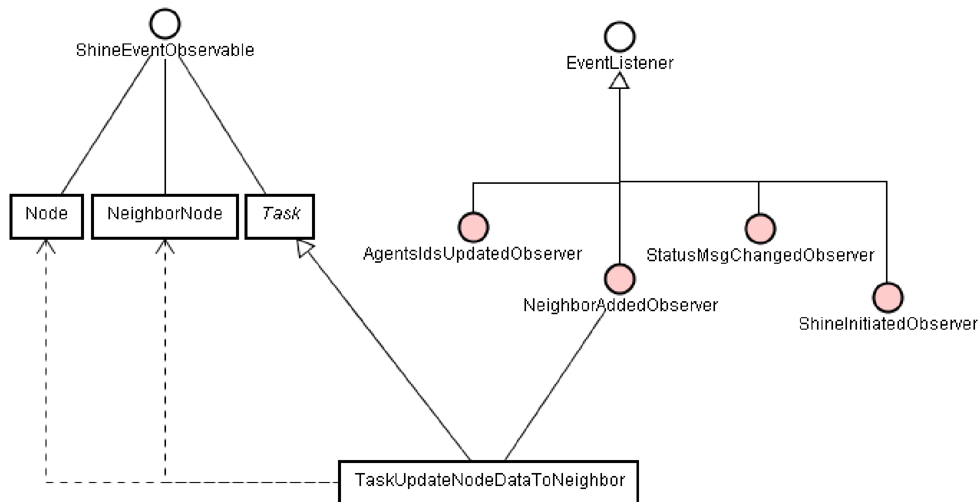

Figura 13: diagrama de classes dos *hot spots* do padrão Observer

PUC-Rio - Certificação Digital Nº 0310882/CB

4.6.1.3. Configuração XML

Tendo o objetivo de facilitar a instanciação do *framework* Shine, flexibilizar a troca das implementações dos serviços existentes por novas implementações (caso venham a ser implementadas) e ainda facilitar a visualização dos agentes e respectivas tarefas pertencentes a cada `Node`, foi criada uma configuração de cada um via arquivo XML.

A Tabela 1 apresenta um exemplo da parte do arquivo XML que define a identificação do `Node` e os serviços citados nas seções anteriores. É possível observar que para cada um dos serviços, deve-se especificar a classe que implementa as interfaces de cada um deles. Desta forma, caso haja a necessidade de alterar a forma de implementação destes serviços, basta criar uma classe que implemente a interface do serviço em questão, e alterar a configuração do XML. Nenhuma alteração de código já existente é necessária, não sendo necessário, portanto, recompilação do *framework*. A especificação da estrutura deste arquivo XML, feita através de XML Schema, se encontra no ANEXO I.

```
<node>
   <id>node2</id>
   <ip>139.82.24.252</ip>
   <messageTransport>
      br.pucrio.les.shine.agent.msg.MessageTransportSpringRMI
   </messageTransport>
   <receiveMessageService>
      <port>1199</port>
      <serviceName>ReceiveMessageService</serviceName>
      <serviceClass>
         br.pucrio.les.shine.service.ReceiveMessageServiceImpl
      </serviceClass>
   </receiveMessageService>
   <agentDiscoveryService>
      <port>1199</port>
      <serviceName>AgentDiscoveryService</serviceName>
      <serviceClass>
         br.pucrio.les.shine.service.AgentDiscoveryServiceImpl
      </serviceClass>
   </agentDiscoveryService>
   ...
</node>
```

Tabela 1: arquivo XML de configuração de Node (Serviços)

PUC-Rio - Certificação Digital Nº 0310882/CB

A Tabela 2 mostra a parte da configuração XML que determina os agentes que participarão do `Node` em questão e suas respectivas tarefas. Observando a Tabela 2, percebe-se a configuração da identificação do agente (pela tag `<id>`), assim como a classe que implementa este agente. Esta classe pode ser a própria classe de agente do *framework* (`br.pucrio.les.shine.agent.Agent`), que fornece toda a capacidade de executar tarefas com autonomia e de interagir com outros agentes através de troca de mensagens. Ou, caso haja necessidade de especializar o agente, pode-se estender esta classe, que é um *hot spot* do *framework*, e implementar as características específicas necessárias. Observando ainda a Tabela 2, pode-se perceber que as tarefas estão associadas a performativas. Estas performativas são determinadas no momento de envio de uma mensagem. O agente, ao receber a mensagem, executa as tarefas referentes a performativa em questão. Pode-se perceber ainda em destaque nesta tabela, que é possível criar uma dependência entre tarefas, para que uma seja executada apenas quando a outra é finalizada.

```
<agents>
   <agent>
      <id>ALEAgent</id>
      <class>br.pucrio.les.shine.agent.ALEAgent</class>
      <tasks>
         <performative>
            <value>askReport</value>
            <taskClass id="AskReport">
               br.pucrio.les.shine.agent.task.TaskAskReport
            </taskClass>
         </performative>
      </tasks>
      <aleProxy>br.pucrio.les.shine.AleProxyLocalImpl</aleProxy>
      <ecSpecs>
         <ecSpec>ecspec-Node2Reader.xml</ecSpec>
         <ecSpec>ecspec-Inventory.xml</ecSpec>
      </ecSpecs>
      <configurer>
         br.pucrio.les.shine.config.ALEAgentConfigurer
      </configurer>
  </agent>
  <agent>
    <id>AleReportsProcessorAgent</id>
    <class>br.pucrio.les.shine.agent.Agent</class>
      <tasks>
        <performative>
           <value>receiveReport</value>
           <taskClass id="RecRep" dependsOn="PrepareToReceiveReports">
               br.pucrio.les.shine.agent.task.TaskReceiveReport
            </taskClass>
         </performative>
         <performative>
            <value>askReport</value>
            <taskClass id="PrepareToReceiveReports">
               br.pucrio.les.shine.agent.task.TaskPrepareToReceiveReports
            </taskClass>
         </performative>
      </tasks>
   </agent>
</agents>
```

Tabela 2: arquivo XML de configuração de Node (agentes e tarefas)

PUC-Rio - Certificação Digital Nº 0310882/CB

4.6.2. Módulo ShineALE

O módulo ShineALE tem o objetivo de complementar o módulo ShineFSMAD alcançando as funcionalidades de interação com a implementação da especificação ALE, interagindo, portanto, com a infra-estrutura RFID.

Este módulo estende o módulo ShineFSMAD apresentado na seção 4.6.1, criando o *framework* Shine como um todo. Este módulo implementa alguns *hot spots* do módulo ShineFSMAD que podem ser observados na Figura 14.

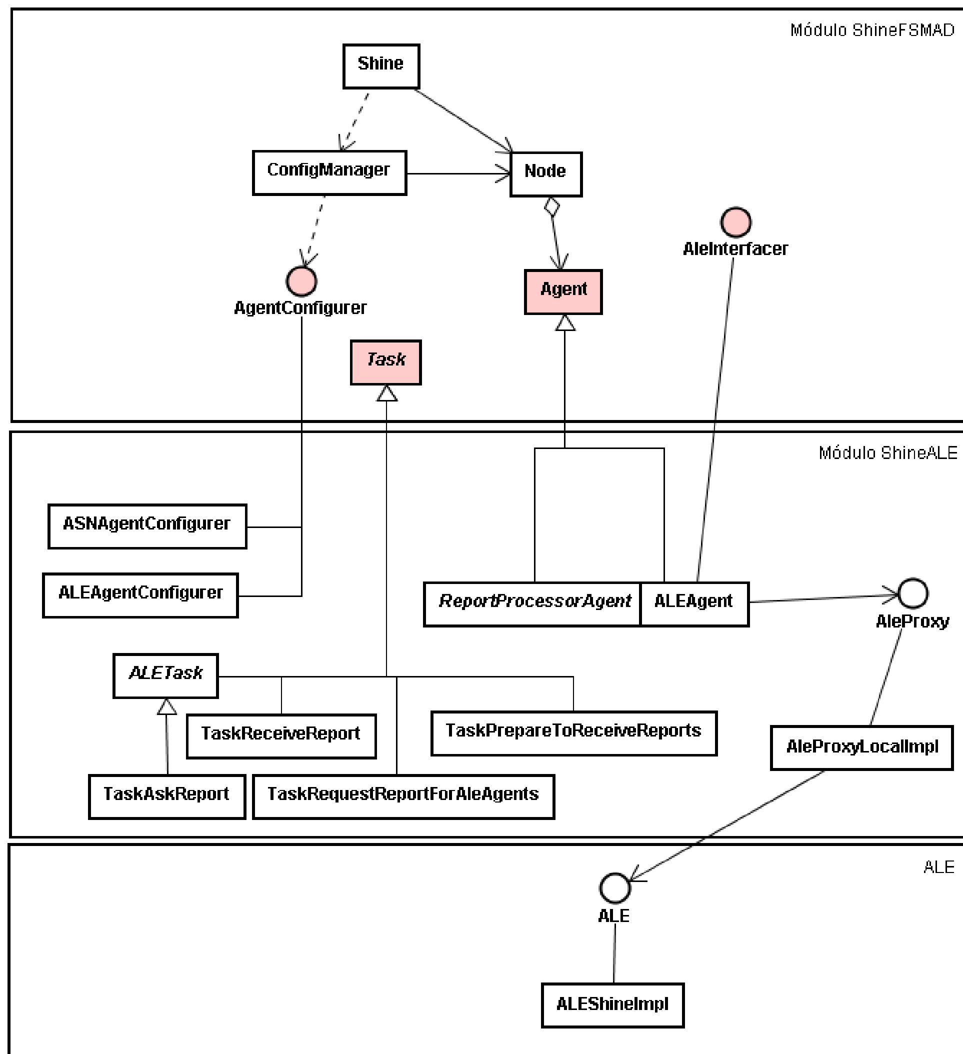

Figura 14: diagrama de classes que implementam os *hot spots* do módulo ShineFSMAD

A Figura 14 mostra as classes implementadas pelo módulo ShineALE que instanciam os *hot spots* do módulo ShineFSMAD, destacados na figura (em cinza). Abaixo, são apresentados detalhes desta implementação.

<u>Detalhes do *Hot spot* “Agent”:</u>

- `ALEAgent`: este agente é responsável por fazer a interação com a implementação da camada ALE (ver seção 2.3.4). Ele tem uma associação com `AleProxy` (implementação do padrão de projeto Proxy [18]), cuja implementação (`AleProxy` é uma interface) é responsável por interagir com a devida implementação da especificação ALE.

PUC-Rio - Certificação Digital Nº 0310882/CB

PUC-Rio - Certificação Digital Nº 0310882/CB

- `ReportProcessorAgent`: este agente é responsável por processar relatórios de ciclos de eventos (*Event Cycles,* ver seção 2.3.4) provindos de um ou mais `ALEAgent`s. Ao mesmo tempo em que este agente é uma extensão de um *hot spot* do módulo ShineFSMAD, ele também é um *hot spot* do *framework* Shine, pois, embora estenda a classe Agent, ainda define um método abstrato que deve ser implementado pela classe da aplicação específica. A Figura 15 mostra a classe que representa o agente. Nela, pode-se ver o método processReport(), que é um método abstrato que concretiza o *hot spot*. Este é o método que deve ser implementado por instâncias do *framework* para realizar o processamento de relatórios da maneira que melhor convier.

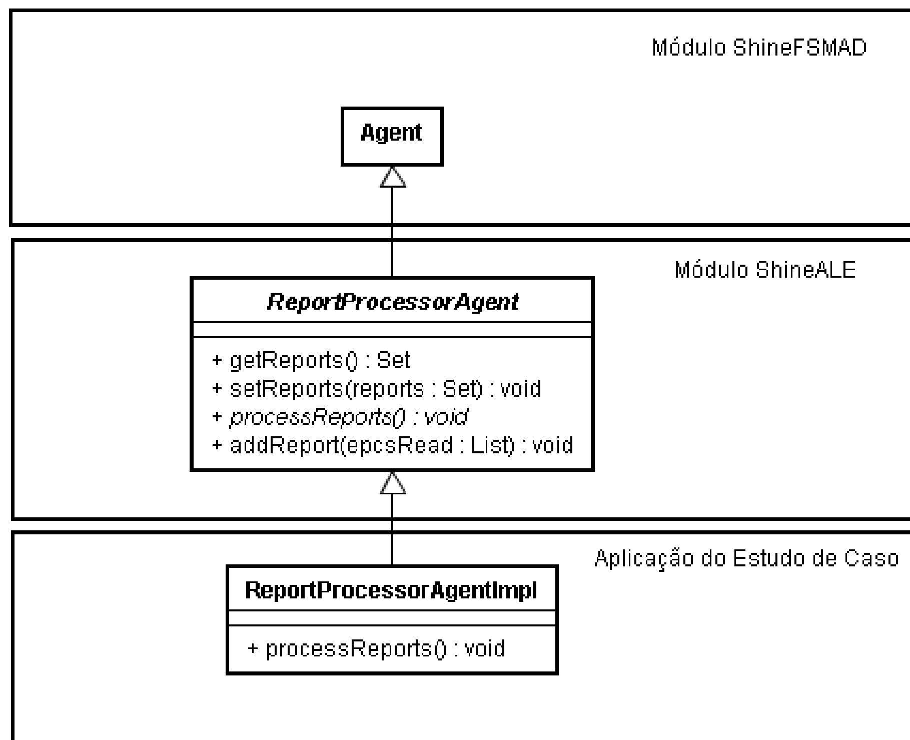

Figura 15: diagrama de classes do *hot spot* ReportProcessorAgent

- Existem outros agentes que são objetos da classe `Agent`, e por isso não aparecem tão explicitamente no diagrama de classes da Figura 14. Porém, são agentes que agregam tarefas (`Task`s) importantes para o funcionamento deste módulo. Um exemplo é o agente denominado `RequestsManagerAgent`, configurado no

arquivo XML de configuração. Este agente executa a tarefa de requisitar relatórios de ciclos de eventos a todos os `ALEAgent`s de todos os Nodes. Detalhes destas tarefas serão apresentados a seguir.

Detalhes da instanciação do *hot spot* "Task":

- `ALETask`: esta tarefa, além de ser uma implementação de um *hot spot* do módulo ShineFSMAD, ela é um *hot spot* do *framework* Shine. Desenvolvedores que desejarem interagir com a implementação da especificação ALE devem criar tarefas que estendam este *hot spot*, pois a `ALETask` possui uma associação com `AleProxy`, responsável por interagir com a camada ALE.
- `TaskAskReport`: como pode ser visto na Figura 14, esta tarefa implementa o *hot spot* `ALETask` detalhado acima. Esta tarefa é responsável por requisitar relatórios de ciclos de evento para a camada ALE, e enviá-los para o agente responsável por receber e processar relatórios.
- `TaskReceiveReport`: esta é a tarefa responsável por receber relatórios enviados pelos diversos `ALEAgent`s. Caso o agente que esteja executando esta tarefa seja uma instância de `ReportProcessorAgent`, então esta tarefa executa o método `processReports()`, que é um *hot spot* do Shine.
- `TaskPrepareToReceiveReports`: esta tarefa tem o intuito de avisar o agente processador de relatório em relação a relatórios provenientes de `ALEAgent`s que ele irá receber. Sendo assim, o agente processador de relatórios sabe exatamente quais `ALEAgent`s devem enviar relatórios, podendo assim esperar por cada um deles e só processá-los depois de receber todos eles, ou tomar alguma ação caso um deles não seja recebido em um determinado período de espera.
- `TaskRequestReportsForAleAgents`: esta é a tarefa responsável por requisitar relatórios de ciclos de evento a cada um dos `ALEAgent`s existentes na aplicação, distribuídos em diferentes `Node`s. Ela também adiciona o agente processador de relatórios como destinatário da mensagem para os `ALEAgent`s, para que ele possa executar a tarefa de preparação descrita acima.

PUC-Rio - Certificação Digital Nº 0310882/CB

<u>Detalhes da instanciação do *hot spot* “AgentConfigurer”:</u>

- `ALEAgentConfigurer`: Esta classe é uma implementação da interface `AgentConfigurer`, e tem o objetivo de configurar o `ALEAgent`. A configuração do XML do `ALEAgent` possui outros elementos além dos elementos padrões do módulo ShineFSMAD, e precisa de um configurador específico que conheça esse novos elementos, já que o `ConfigManager` conhece apenas os elementos padrões.

Através da implementação das classes detalhadas acima, conseguiu-se, portanto, implementar um *framework* com a capacidade de ser estendido para criar aplicações de sistemas multi-agentes distribuídos que têm a capacidade de interagir com uma infra-estrutura RFID que expõe sua interface pública através da interface ALE definida pela especificação de mesmo nome da EPCglobal Inc. Estas aplicações podem simplesmente se concentrar em implementar tarefas que tenham regras de negócios que processem os dados de relatórios obtidos de ciclos de eventos, sem se preocupar em como obter estes dados, e em como se comunicar com agentes distribuídos. Esta afirmação é demonstrada nos estudos de casos descritos a seguir.

PUC-Rio - Certificação Digital Nº 0310882/CB