3
Framework CubiMed

Neste capítulo, descrevemos o *framewok* “CubiMed”, que tem como propósito a criação de aplicações no contexto de assistência medica ubíqua (u-Healthcare), baseadas nos princípios de SMA e TCAC. Primeiramente, apresenta-se uma visão geral do framework, seguida de uma explicação sobre sua arquitetura. Finalmente, apresentam-se seus pontos fixos e flexíveis.

3.1.
Domínio do framework

Conforme descrito no trabalho (MARKIEWICZ & DE LUCENA, 2001), os frameworks são geradores de aplicações que estão diretamente relacionadas a um domínio específico, ou seja, a uma família de problemas relacionados. Isto acontece porque um framework possui código fonte e arquitetura que podem ser reutilizados pelos desenvolvedores, o que agiliza o trabalho e reduz o esforço no momento do desenvolvimento. Com base nisto, explicaremos a seguir o contexto especifico para o qual está direcionado o framework desenvolvido neste trabalho.

Na subseção 1.1, foi apresentado o ciclo base de atendimento ao paciente, no qual pode-se observar que o paciente e o médico apenas possuem um contato direto no momento da consulta. Depois disso, a interação é esporádica, o que não permite que o médico possa acompanhar adequadamente o paciente. A mesma situação ocorre com os membros da equipe médica, já que eles só conhecem o estado de saúde do paciente quando o médico os informa. Desta forma, torna-se difícil fornecer um serviço integral que auxilie o paciente em sua recuperação imediata. Neste sentido, em (CHAN et al., 2008) é apresentado um cenário de interação entre o paciente e a equipe médica, que mostra como normalmente ocorre o fluxo de informação, permitindo entender melhor o contexto em que o framework CubiMed foi desenvolvido. Na figura 7 apresenta-se este cenário de interação e depois são descritos os passos do cenário.

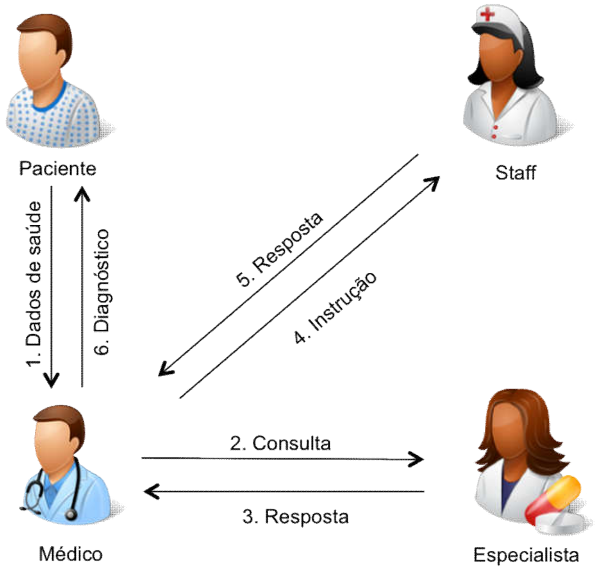

Figura 7: Cenário de interação Médico-Paciente

1. O paciente fornece os dados sobre sua saúde para o médico.
2. Se for preciso, o médico consulta um especialista para ter condições de identificar o problema do paciente, de forma direta ou solicitando que o paciente consulte um especialista.
3. O especialista fornece uma resposta sobre a consulta, que pode ser um diagnóstico e uma possível solução.
4. De posse de um diagnóstico, o médico também pode interagir com algum membro do staff médico, como uma enfermeira, a fim de instruí-la sobre algum tipo de atendimento que o paciente deve receber.
5. O membro do staff cumpre a instrução e fornece uma resposta para o médico.
6. Finalmente, o médico fornece um diagnóstico ao paciente, além de instruções que ele deve seguir.

Considerando o ciclo básico de atendimento médico apresentado no primeiro capítulo, bem como o cenário de interação da equipe médica com o paciente apresentado acima, pode-se verificar que o procedimento de atendimento ao paciente e a interação entre os membros da equipe médica possuem um fluxo básico genérico, independentemente do tipo de paciente com o qual estão lidando. Nesse contexto, este trabalho está focado especificamente em projetar e desenvolver um framework que permita aos desenvolvedores

fazerem uso de conceitos de computação ubíqua, TCAC e SMA, na área da assistência médica.

Dessa forma, os desenvolvedores poderão criar aplicações de atendimento remoto para o paciente, não importando o lugar onde este se encontre, mas considerando o conceito de coordenação entre todos os membros da equipe médica. O foco deste framework consiste em prover ao desenvolvedor ferramentas já desenvolvidas, que possam ser reutilizadas e que permitam a comunicação entre as diferentes pessoas envolvidas no atendimento médico. O framework também mantém continuamente a interação entre os envolvidos, por meio de agentes que representem cada um dos participantes e que possam automatizar alguns dos processos já conhecidos.

A modelagem e desenvolvimento do framework são apresentados na seção seguinte.

3.2. Arquitetura

Conforme explicado na seção 2.6, já foram propostos trabalhos que pretendem relacionar a área de TCAC ao paradigma SMA, onde a comunicação dos diferentes participantes que fazem uso de aplicações colaborativas é permitida através das já utilizadas arquiteturas de três camadas e de agente representante, por meio de agentes.

Neste trabalho, utiliza-se uma arquitetura híbrida que reúne as duas arquiteturas, com o propósito de:

- Usar a arquitetura de três camadas para estabelecer a estrutura de comunicação do framework, a qual será usada em todas as aplicações desenvolvidas utilizando-se o CubiMed;
- Usar a arquitetura de agente representante para permitir que todos os usuários de aplicações desenvolvidas a partir deste framework tenham a possibilidade de serem assistidos por um agente instanciado em uma aplicação web, *desktop* ou *mobile*, no momento em que realizam suas tarefas.

As duas arquiteturas mencionadas estão estritamente relacionadas, já que na camada de colaboração do aplicativo, na arquitetura de três camadas, será aplicada a arquitetura de agente representante.

Também precisamos destacar que o modelo de consciência proposto pela TCAC e descrito na seção 2.6.1 aplica-se a toda arquitetura do framework. Isso permitirá que todos os participantes das aplicações desenvolvidas com o framework tenham consciência do que está sendo realizado.

Como o framework proposto permite criar aplicações colaborativas, estamos assumindo que todo aplicativo criado a partir deste framework terá pelo menos dois participantes que irão colaborar entre si, o que equivale a um mínimo de dois agentes. Dessa forma, todas as aplicações criadas com o CubiMed serão consideradas um SMA. Como afirmam (INGLADA & NAVARRO, 2002), o SMA tem que dispor de uma infraestrutura que inclua todos os aspectos relacionados aos processos de comunicação, ou seja, com o envio e recebimento de mensagens. Já que a criação dessa infraestrutura é muito complexa e demandaria muito tempo para a sua construção, o framework proposto é criado sobre a plataforma JADE e JADE-LEAP, as quais são descritas nas seções 2.4 e 2.5, respectivamente. Ambas fornecem o suporte necessário para trabalhar com agentes, de acordo com as especificações FIPA.

3.2.1. Especificação da arquitetura

Como já foi mencionado, o framework proposto trabalha usando uma arquitetura hibrida entre a arquitetura de três camadas e a arquitetura de agente representante. Nesse sentido, o framework permitirá criar dois tipos de agentes: os que proveem os serviços de colaboração e os que serão instanciados nos aplicativos de colaboração. Estes últimos irão permitir aplicar a arquitetura de agente representante, já que será criado um destes agentes para cada participante do sistema. A seguir, são descritas as três camadas do framework, onde são enfatizados aplicativos de colaboração, para explicar como é utilizada a arquitetura de agente representante.

3.2.1.1. Camada de comunicação

Conforme mencionado anteriormente, o framework CubiMed está criado com base na plataforma JADE. Por este motivo, a camada de comunicação está em conformidade com a plataforma interna de transporte de mensagens do JADE, que já contém uma infraestrutura que permite a troca de mensagens FIPA-ACL entre os diferentes agentes, que poderão ser instanciados quando o

framework CubiMed for utilizado. Para que esta comunicação possa existir no framework e todos os agentes instanciados possam entenderem-se uns aos outros, foi criada uma ontologia, a qual é especificada na seção 3.3.1.

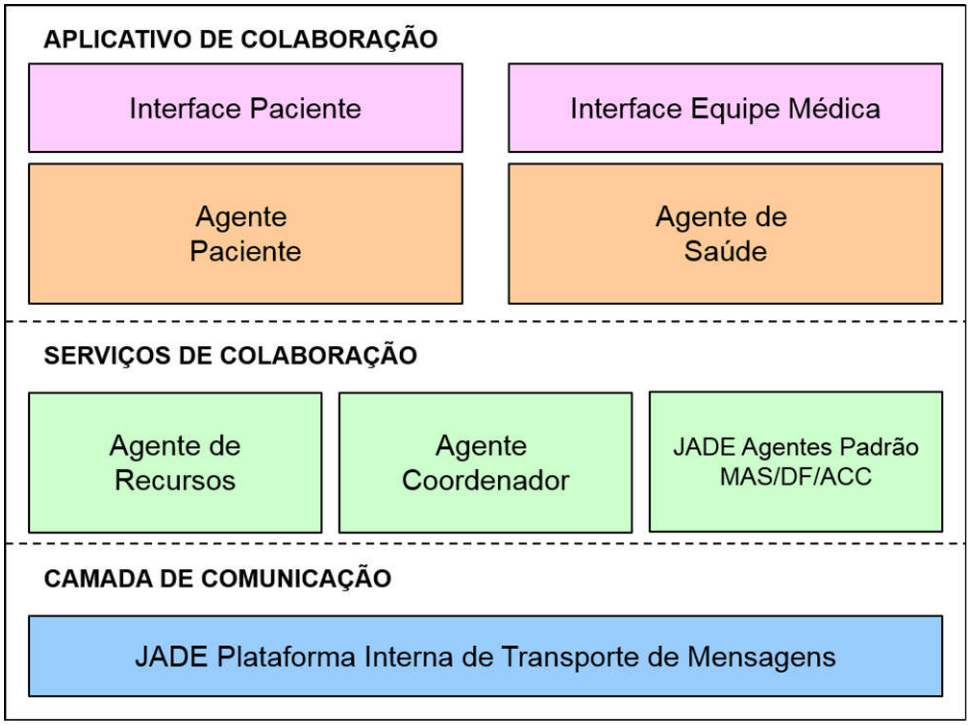

Figura 8: Modelo de três camadas do Framework CubiMed.

3.2.1.2. Camada de serviços de colaboração

A camada de serviços de colaboração é a camada que fornecerá aos agentes participantes do sistema a informação necessária para que eles possam se entender e colaborar entre si. Para isto, e baseados na arquitetura do JADE, esta camada estará representada pelo container principal da plataforma, onde são instanciados os agentes AMS, DF e ACC, quando do início da plataforma. Contudo, além destes agentes já fornecidos pela plataforma JADE, são criados também o Agente Coordenador e o Agente de Recursos, os quais são descritos a seguir.

Agente Coordenador: O agente coordenador, conforme seu nome sugere, tem a função de coordenar todas as interações que existem entre os

agentes do aplicativo desenvolvido. Por conseguinte, este agente possui as seguintes tarefas:

- Verifica e valida o ingresso de um novo agente no sistema;
- Comunica para os outros agentes quem entrou no sistema;
- Comunica para os outros agentes quem saiu do sistema;
- Permite saber com quais agentes um determinado agente pode interagir;

Com o cumprimento destas tarefas, o agente coordenador permite cobrir as categorias “Quem” e “Onde” do modelo de consciência, relacionados ao momento atual, os quais são mostrados na Tabela 1 da seção 2.6.1.

Agente de Recursos: Ao criar um sistema, é preciso considerar o banco de dados onde toda a informação referente ao funcionamento do sistema estará armazenada. Além disso, todos os participantes necessitam ter acesso à informação, para que possam cumprir seus objetivos. O agente de recursos é responsável por gerenciar e centralizar o acesso ao banco de dados. Nesse sentido, este agente poderá cumprir as seguintes tarefas:

- Consulta de informações no banco de dados;
- Registro de informações sobre as atividades dos agentes;
- Atualização de informações no banco de dados;

Já que ter um banco de dados implica em ter o registro de todas as informações relevantes sobre o comportamento dos participantes no sistema, este agente permitirá cobrir todos os pontos do modelo de consciência relacionados ao passado, os quais são apresentados na Tabela 2. Além disso, também cobrirá a categoria “O que” do modelo de consciência, relacionado ao presente, o qual é apresentado na Tabela 1 da seção 2.6.1. Isso será possível sempre que o desenvolvedor criar a estrutura necessária para salvar todas as informações relacionadas a estes pontos.

3.2.1.3. Camada de aplicativos de colaboração

Dentro desta camada, é feita a integração da arquitetura de três camadas com a arquitetura de agente representante. Isto acontece porque cada aplicativo

criado a partir do framework instanciará um novo agente, para cada participante que intervenha na aplicação.

Para não confrontar com a terminologia que está sendo utilizada, consideramos que uma aplicação criada com o framework pode ter muitos aplicativos de colaboração associados à aplicação geral. Além destes aplicativos, encontra-se também o servidor principal, onde é criado o container principal. Neste contexto, aplicativos de colaboração referem-se às aplicações finais com as quais o usuário irá interagir para realizar a colaboração. Os aplicativos de colaboração poderão ser criados como aplicações móveis executando sob Android, ou como aplicações *desktop* e web, executando sob Java. Para cada aplicativo de colaboração, será criado um agente que interage com o usuário e permite que este se comunique com o resto da aplicação.

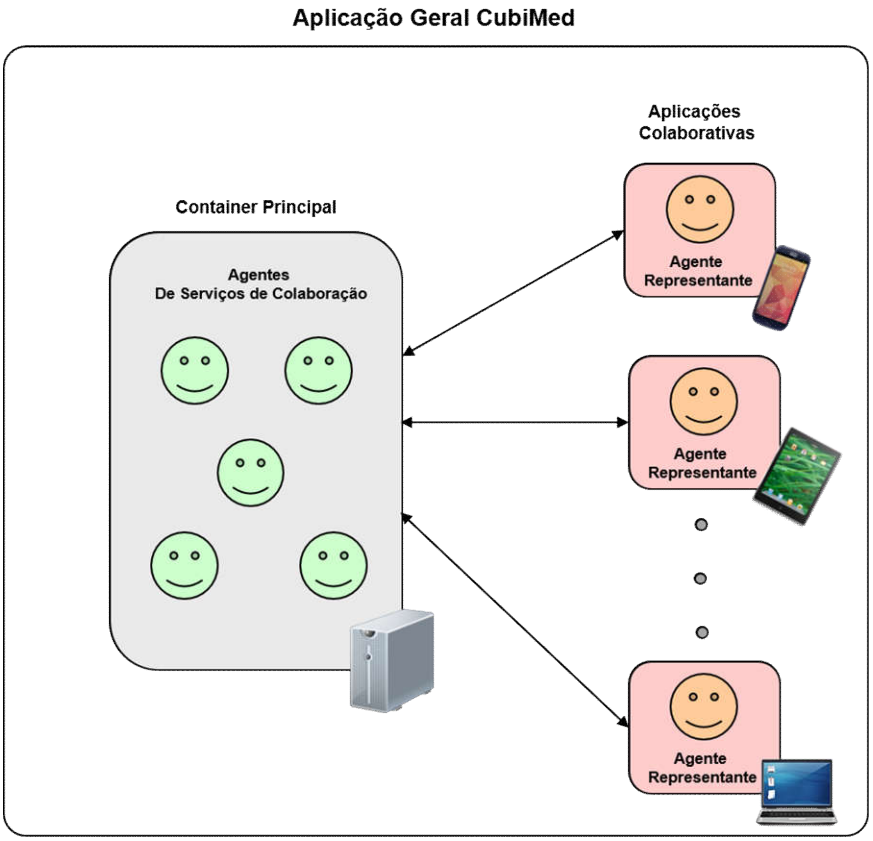

Figura 9: Ilustração geral de uma aplicação criada com o framework CubiMed

A seguir, são descritos os tipos de agentes que poderão ser instanciados em cada aplicativo de colaboração criado.

Agente Paciente: O Agente Paciente é o agente que tem a tarefa de representar o paciente dentro dos aplicativos criados, o qual cumpre as seguintes tarefas:

- Identificar o paciente;
- Monitorar o contexto do paciente, a partir de dispositivos externos como sensores, e enviar a informação para a equipe médica;
- Recuperar as informações da interface criada para o paciente e enviá-las para a equipe médica;
- Consultar a equipe médica sobre possíveis problemas que o paciente possa apresentar;
- Receber instruções da equipe médica e repassá-las ao paciente.

Agente de Saúde: Entidade capaz de representar qualquer membro da equipe médica, podendo representar, desde um especialista a uma enfermeira ou secretaria, dependendo do tipo de funções que o usuário final da aplicação tenha. É importante mencionar que, se o usuário final tiver duas funcionalidades, como, por exemplo, especialista cardiologista e especialista dermatologista, o agente poderá também ser identificado no sistema com essas duas funcionalidades, conseguindo cumprir os dois papéis. Tal agente desempenha as tarefas de:

- Identificar o membro da equipe médica;
- Receber consultas por parte do paciente;
- Receber consultas, instruções e algum tipo de informação por parte de qualquer outro participante do sistema;
- Responder a todas as solicitações, tanto do paciente como de outros membros da equipe médica.

3.2.2. Especificação de aplicações

O framework Cubimed permitirá desenvolver aplicações colaborativas, onde o paciente possa interagir de forma contínua com todos os membros da equipe médica. Para isto, todas as aplicações que venham a ser desenvolvidas com o framework apresentarão uma arquitetura distribuída, na qual existirão várias aplicações colaborativas contendo uma instância de um agente, e, dessa forma, irão representar cada participante. Todas essas aplicações irão se

conectar a um servidor central que coordenará todas a tarefas e irá permitir a colaboração de todos (Figura 10).

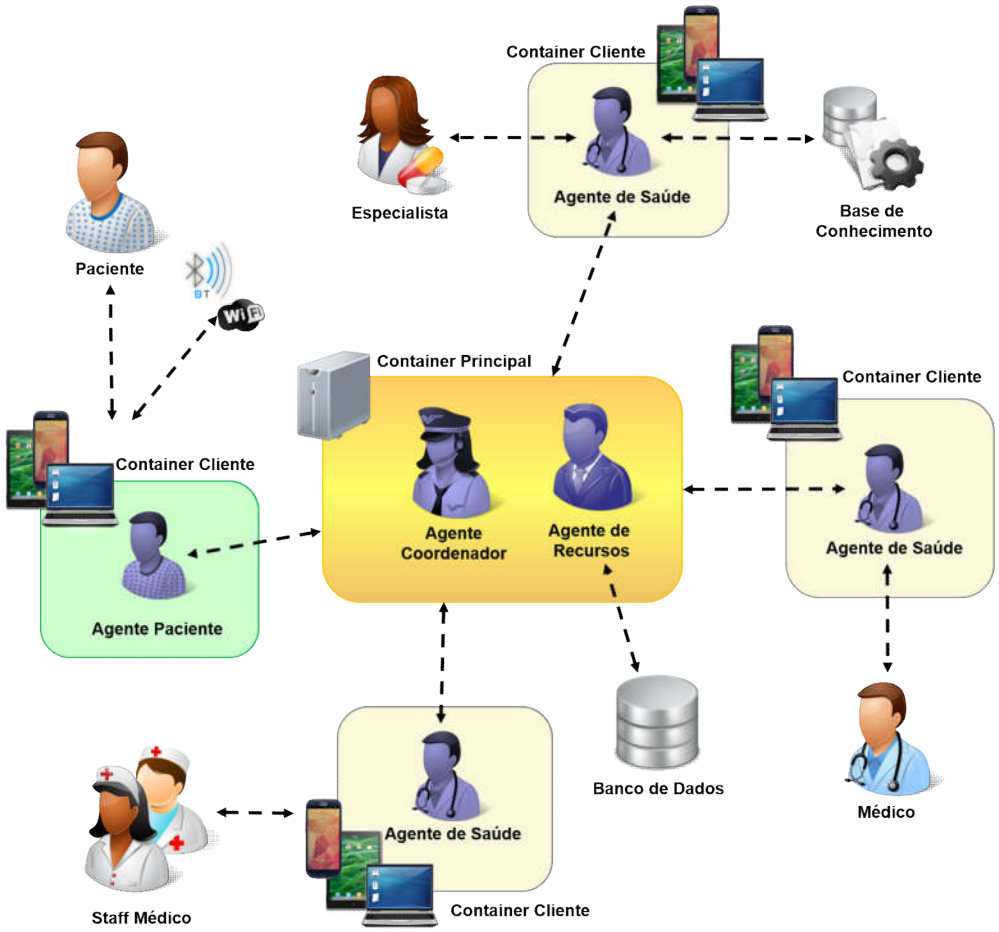

Figura 10: Esquema geral de instanciação do framework CubiMed.

Conforme apresentado na Figura 10, para cada aplicação criada com o framework, deve existir um container principal que estará alocado a um servidor. Neste container, serão instanciados o agente coordenador e o agente de recursos. O agente de recursos permitirá o acesso a um banco de dados, que pode ser um banco de dados relacional, na nuvem ou mesmo um serviço web, cuja escolha será realizada pelo usuário da aplicação.

Para o caso do paciente, poderá ser criada uma aplicação colaborativa na qual será instanciado um agente do tipo paciente, na qual poderão ser coletadas, processadas e enviadas para o servidor informações sobre o paciente, a partir de dispositivos *wi-fi* ou *bluetooth*. Neste caso, o servidor irá direcionar as mensagens aos membros da equipe médica.

Este framework tem foco na criação de aplicações ubíquas para acompanhamento contínuo do paciente, permitindo que tenham um serviço de atenção médica integral. Neste caso, as aplicações colaborativas que o paciente irá utilizar sempre terão a possibilidade de serem criadas em dispositivos móveis. Contudo, caso o desenvolvedor ou responsável pela criação da aplicação considere importante, também poderá criar aplicações desktop ou web, conforme julgar necessário.

O framework também permite a criação de uma aplicação colaborativa para cada membro da equipe médica, independentemente do tipo de responsabilidade que cada um deles possa ter. Dentro de cada aplicação, será instanciado um agente de saúde que pode assumir o papel de qualquer pessoa do staff, cabendo ao desenvolvedor a definição das tarefas que o agente irá realizar. Desta forma, o agente pode ser apenas um mero representante membro do staff, encarregando-se de receber ou enviar mensagens. Também pode-se manter uma base de conhecimento que permita que o agente realize processos automáticos em nome do membro do staff. Da mesma forma que nas aplicações destinadas ao paciente, as aplicações para os médicos também poderão ser criadas em ambientes diferentes, ou seja, móvel, desktop e web. Desta forma, quando se fizer necessário que agentes de tipo médico realizem grandes processamentos, estes não enfrentarão problemas de comunicação com o restante dos agentes.

3.3. Implementação

Nesta seção será explicada de forma detalhada, como o framework proposto neste trabalho foi implementado.

A implementação do CubiMed foi realizada em duas partes, já que o framework precisava permitir a criação de aplicações de forma distribuída, suportando a instanciação de aplicações tanto em ambientes móveis como em ambientes estáticos, como um servidor ou um computador pessoal. A primeira parte consistiu na implementação normal do CubiMed que está baseado no core do JADE e poderá ser executada em qualquer computador que suporte JAVA; A segunda parte, por sua vez, consiste em uma implementação feita especificamente para rodar em dispositivos móveis que suportem Android, na qual foi utilizado o JADE-LEAP. Por esse motivo, a segunda parte do framework será referenciada como CubiMed-LEAP. Dentro das duas implementações,

existem algumas variações para que elas possam se adequar ao entorno onde serão executadas. Entretanto, a parte da comunicação onde é estabelecida a ontologia que será usada pelos agentes é mantida nas duas implementações para permitir que a troca de informação seja realizada de forma satisfatória (Figura 11).

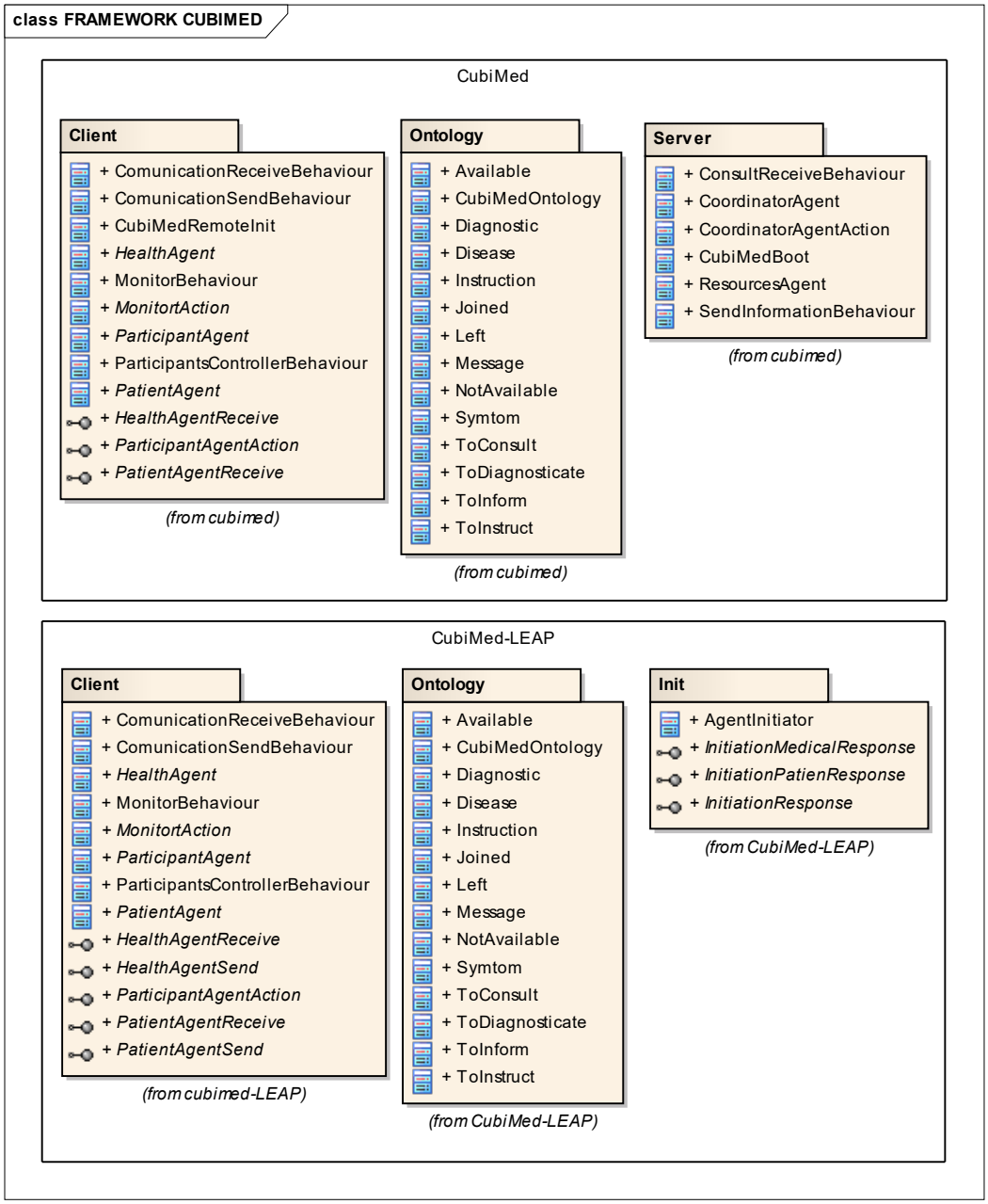

Figura 11: Organização de pacotes no framework CubiMed

A seguir, explicamos como foi implementado todo o framework. Primeiro, apresenta-se uma visão detalhada da parte da comunicação, focando na

ontologia que é utilizada tanto pelo CubiMed quanto pelo CubiMed-LEAP. Em seguida, descreve-se as duas implementações, mostrando suas diferenças e explicando o funcionamento de cada uma delas.

3.3.1. Comunicação

Para que os agentes criados com o framework CubiMed possam comunicar entre si, eles precisam compartilhar o mesmo idioma, vocabulário e protocolos. Conforme mencionado anteriormente, o framework CubiMed está baseado na plataforma JADE, que têm um certo grau de consciência sobre os pontos necessários para a comunicação, uma vez que usa as especificações FIPA e a linguagem de conteúdo. Contudo, é preciso definir ontologias especificas, com um vocabulário próprio e semântica de conteúdo, para que os agentes possam interpretar a atividade médica de colaboração e atendimento ao paciente.

Conforme descrito na subseção 2.4.1, o JADE permite definir ontologias com base em três tipos de objetos: conceitos, predicados e ações dos agentes. Nesse sentido, foram definidos os seguintes objetos para a ontologia do framework, os quais estão baseados no ciclo básico de atendimento a um paciente (Figura 12).

Conceitos:

- Mensagem: Corresponde ao conceito mais simples e serve para possibilitar o envio de qualquer tipo de informação entre os participantes das aplicações. Pode ser utilizada, por exemplo, para informar se uma tarefa foi realizada, caso o médico precise informar que não poderá atender uma solicitação. Este conceito também é usado pelo agente de recursos para responder a consultas feitas por outros agentes, quando estes precisam de alguma informação do banco de dados.
- Sintoma: Conceito que serve para informar os membros da equipe médica sobre o estado de saúde do paciente. Os sintomas podem ser usados para enviar os dados recolhidos pelos sensores ou simplesmente os dados introduzidos pela interface da aplicação colaborativa, com o objetivo de manter o monitoramento do paciente.

- Doença: Conceito que os membros da equipe médica podem usar para informar sobre uma descoberta no estado de saúde do paciente. Este conceito normalmente será usado depois de analisar os sintomas do paciente, como uma alerta para que a equipe médica possa realizar uma ação ou para informar o paciente que ele precisa tomar alguns cuidados.
- Instrução: O conceito de instrução é usado para que um determinado agente da aplicação possa fornecer um conjunto de instruções a outro agente. Estas instruções podem ser, por exemplo, uma prescrição médica, uma ordem para que alguma pessoa do staff atenda o paciente ou até mesmo uma instrução de qualquer um dos participantes para o agente de recursos, a fim de que este possa obter alguma informação do banco de dados.

Ações dos Agentes:

- Informar: Esta é a ação que será utilizada quando um dos participantes desejar enviar uma informação para outro participante. Sempre que esta ação for executada, será enviado para o participante destinatário um conceito do tipo mensagem.
- Consultar: Quando um paciente precisar fazer uma consulta para um membro da equipe médica ou simplesmente desejar informar sobre o estado de saúde do paciente, esta ação será executada, tendo sempre como conteúdo um conceito do tipo sintoma.
- Diagnosticar: Uma vez analisada a informação obtida do paciente, esta ação poderá ser executada e terá como conteúdo um conceito do tipo doença, com o qual será possível alertar os outros participantes.
- Instruir: Ação executada para enviar uma ordem para algum participante do sistema. Sempre irá apresentar como conteúdo um objeto do tipo instrução.

Predicados:

- Disponível: Este predicado será utilizado por um agente para informar aos outros agentes que ele está ativo no sistema e que qualquer um que precise dos seus serviços poderá comunicar-se com ele para obter sua ajuda.

- Indisponível: Ao contrário do predicado anterior, este predicado será utilizado para informar os outros agentes que o participante encontra-se ativo na aplicação, mas está realizando alguma outra tarefa, não podendo colaborar no momento.
- Ingresso: Este predicado permitirá indicar a todos os agentes da aplicação que um novo agente ingressou no sistema e que podem comunicar-se com ele, caso precisem de algum serviço que ele ofereça.
- Saída: Contrariamente ao predicado anterior, este predicado será usado para informar aos outros agentes que um determinado agente saiu do sistema, não sendo mais possível contar com sua colaboração.

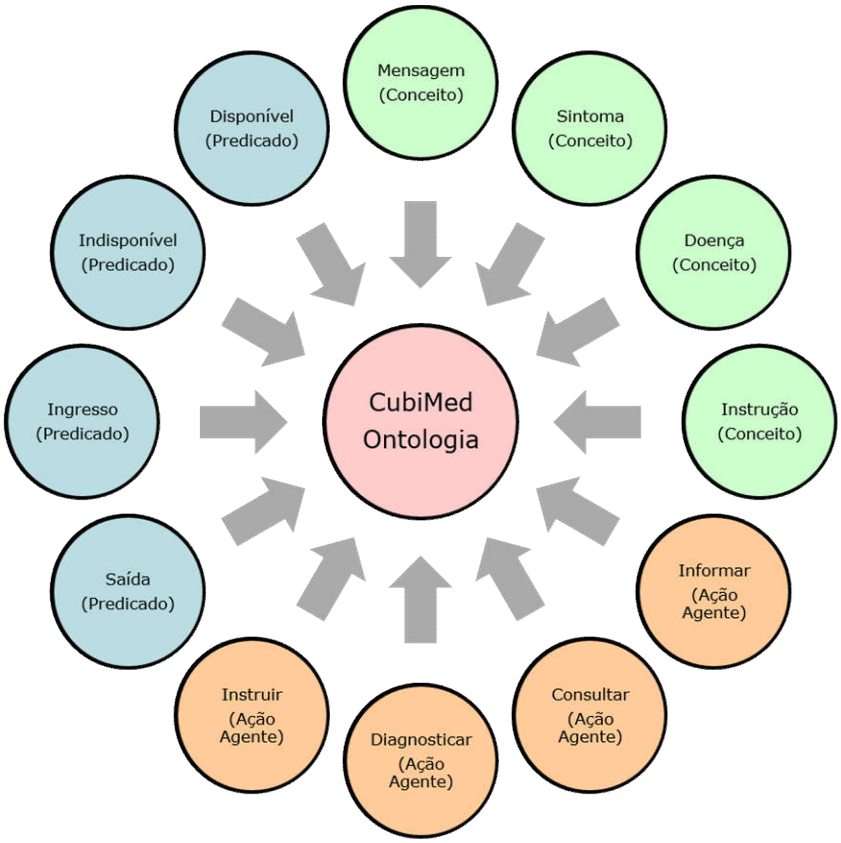

Figura 12: Ontologia Framework CubiMed

Depois de ter estabelecido a ontologia que vai ser utilizada no framework, foi usada a ferramenta PROTEGE (PROTÉGÉ, 2015). PROTEGE é uma aplicação de código aberto desenvolvida na universidade de Stanford, que permite criar ontologias, através de uma interface gráfica, que podem ser exportadas para código Java (NOY, 2001) e interpretadas pelo JADE, bem como

pelo framework CubiMed. A seguir na Figura 13, apresenta-se uma captura de tela que permite visualizar como foi criada a ontologia na ferramenta mencionada.

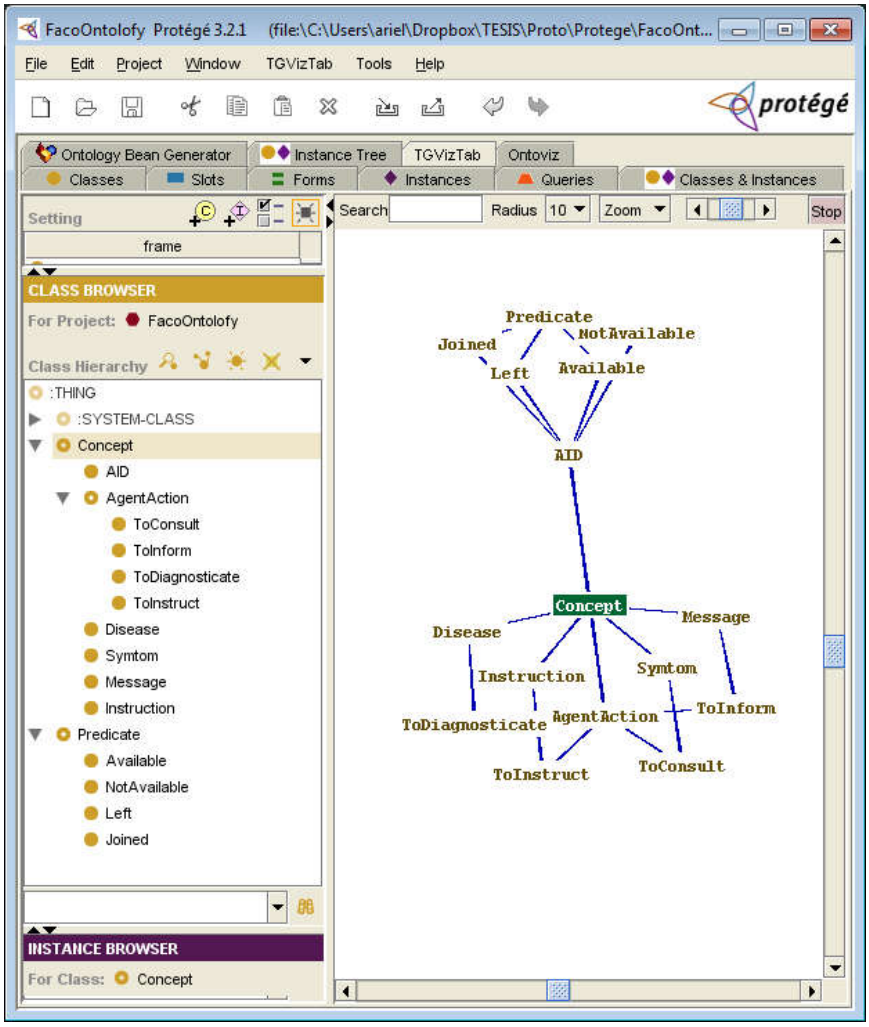

Figura 13: Criação da ontologia com a ferramenta PROTEGE

Uma vez projetada a ontologia com a ferramenta PROTEGE, foi possível usar o complemento *OntologyBeanGenerator* (VAN AART, 2015), que tornou possível a geração de classes Java, posteriormente adaptadas para funcionar de forma compatível com o framework.

A seguir, é apresentada a estrutura final da ontologia, representada em forma de um diagrama de classes, no qual pode-se apreciar as relações finais que serão usadas no framework para permitir a comunicação entre todos os agentes instanciados.

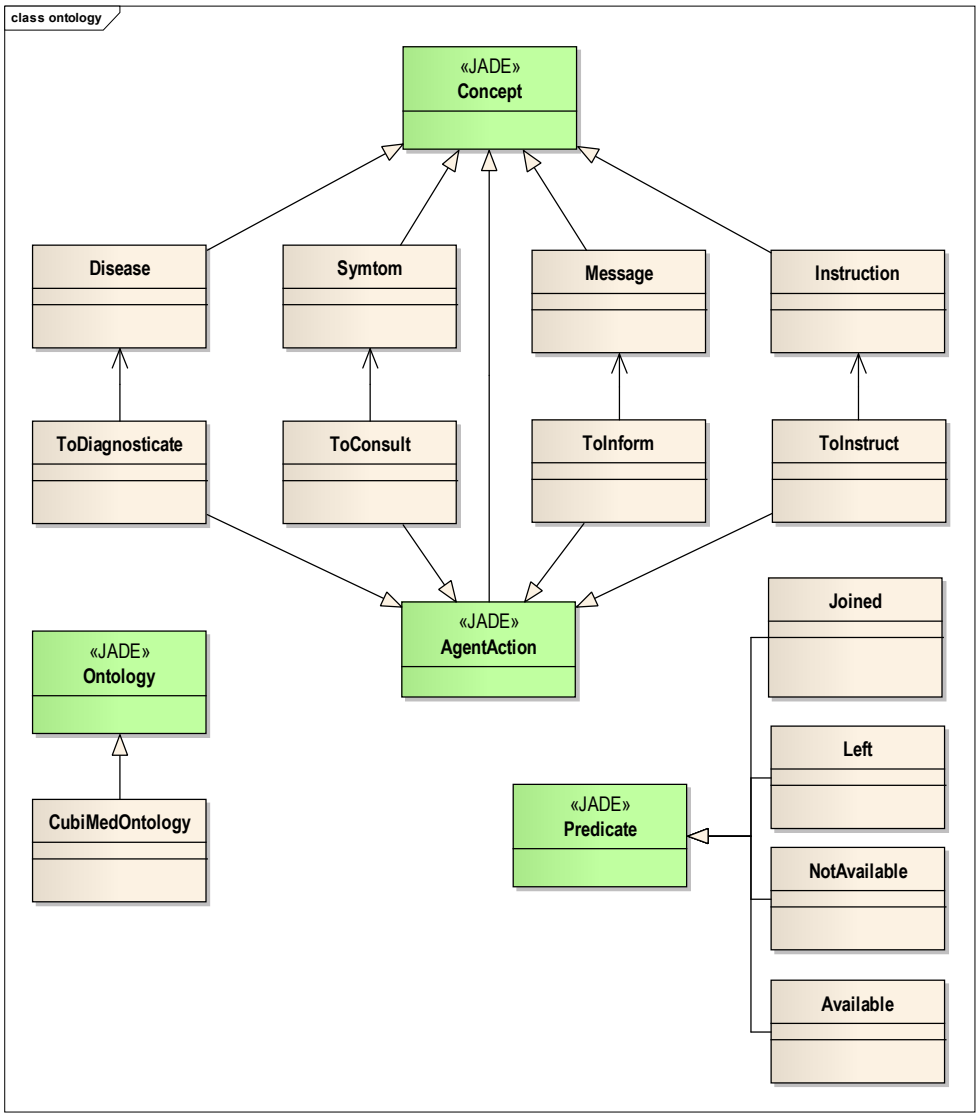

Figura 14: Diagrama de classes do pacote Ontology

Como pode-se observar na Figura 14, as classes *Disease, Symtom, Message* e *Insctruction* são os conceitos da ontologia. Elas são herdadas da classe *Concept* do JADE, as quais podem ser estendidas pelos desenvolvedores para personalizar as informações que serão enviadas. Tais classes tem uma relação direcional com *ToDiagnosticate, ToConsult, ToInform* e *ToInstruct* respectivamente, que, por sua vez, herdam propriedades da classe *AgentAction,* também pertencente ao JADE. As classes *Joined, Left, NotAvailable, Availble* herdam propriedades do *Predicate*. Finalmente, pode-se verificar que existe uma classe denominada *CubiMedOntology* que tem uma relação de generalização com a classe *Ontology* do JADE. Esta classe permite estabelecer a organização

da ontologia, estabelecendo as relações existentes para que possa ser processada pela plataforma JADE.

Todas estas classes formam o pacote *Ontology* do framework, que é usado tanto pela implementação do CubiMed como pela implementação do CubiMed-LEAP, sem que seja necessário realizar quaisquer mudanças.

3.3.2. Implementação do CubiMed

Como pode-se observar na Figura 11, a implementação do CubiMed está em conformidade com os pacotes *Ontology, Server e Agent.* Nesta seção, serão apresentadas as implementações dos pacotes *Server* e *Client,* visto que o pacote *Ontology* já foi implementado.

Server: Este pacote é uma implementação especifica do CubiMed e não possui implementação semelhante no CubiMed-LEAP. As classes criadas neste pacote têm o objetivo de iniciar a plataforma com todos os agentes necessários para que a colaboração possa existir. A seguir, é apresentado o diagrama de classes, juntamente com uma explicação sobre o seu funcionamento.

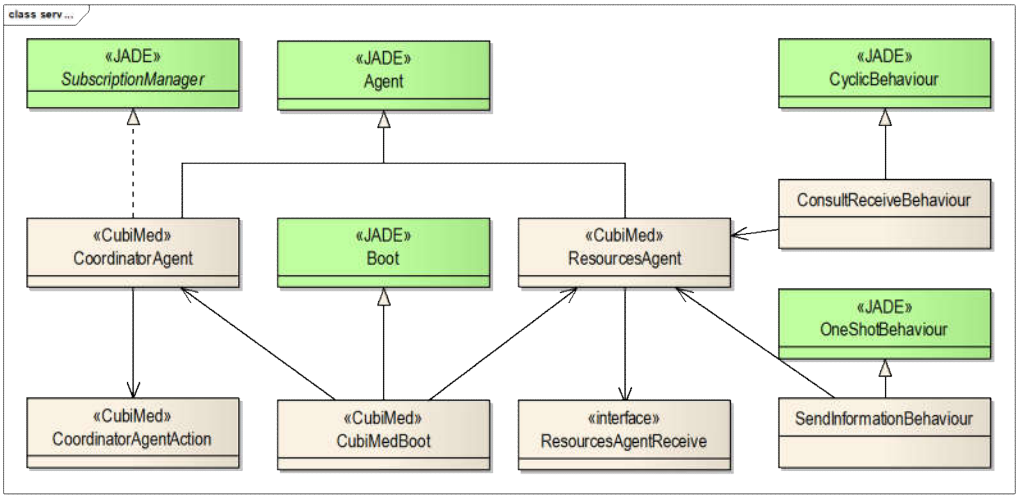

Figura 15: Diagrama de classes do pacote Server, CubiMed

No pacote em questão, pode-se identificar as classes *CoordinatorAgent* e *ResourcesAgent*, as quais estendem a classe Agent do JADE.

O *CoordinatorAgent,* além de estender a classe *Agent,* também implementa a interface *SubscriptionManager.* Desta forma, no momento da sua instanciação, pode-se usar a classe *AMSSubscriber,* também do JADE, para

obter controle sobre as subscrições dos novos agentes na aplicação, podendo assim, serem usados os predicados *Joined e Left* descritos na ontologia, para informar aos agentes que participam da aplicação que um outro agente entrou ou saiu do sistema.

Além destas características, o *CoordinatorAgent* recebe como parâmetro uma instancia da classe *CoordinatorAgentAction,* que consiste em uma classe abstrata com métodos de autenticação e listagem de participantes que funcionam da seguinte forma:

- No método de autenticação, o desenvolvedor poderá receber um objeto serializado onde estarão os dados que sejam precisos para a autenticação. Com este objeto, se conseguirá fazer uma comprovação em memória ou no banco de dados para autenticar ao agente e permitir sua participação na aplicação.
- O método de listagem, tem como parâmetro o identificador do agente que está fazendo a solicitação. Este identificador, pode ser usado para definir a visibilidade dos agentes que o solicitante terá. Ou seja, será o desenvolvedor ao implementar este método quem definirá como estará conformada a lista de participantes que o agente que está fazendo a solicitação poder ver.

A classe *ResourcesAgent,* tem o propósito de permitir e gerenciar o acesso aos recursos do banco de dados. Para isto, ela implementa dois comportamentos que estão representados nas classes *ConsultReceiveBehaviour* e *SendInformationBehaviour,* O primeiro consiste de uma extensão do *CiclicBehaviour,* convertendo-se em um comportamento cíclico que é executado continuamente, à espera de que um dos agentes faça alguma requisição. O segundo é um comportamento que estende o *OneShootBehaviour* e é executado quando se deseja enviar uma mensagem para algum agente.
Finalmente, dentro do pacote existe a classe *CubiMedBoot,* que estende as funcionalidades da classe *Boot* do JADE, sobrescrevendo seu código para inicializar os agentes de coordenação e recursos, em conjunto com a plataforma.

Client: O Pacote *Client* é criado tanto na implementação do CubiMed quanto na do Cubimed-LEAP, com algumas pequenas variações para adequar-se ao ambiente onde vai ser executado. Este pacote tem o objetivo de prover os

recursos necessários para que o desenvolvedor possa criar instâncias de agentes para cada um dos aplicativos colaborativos que pretende desenvolver.

A seguir, são apresentados o diagrama de classes e a descrição do funcionamento deste pacote.

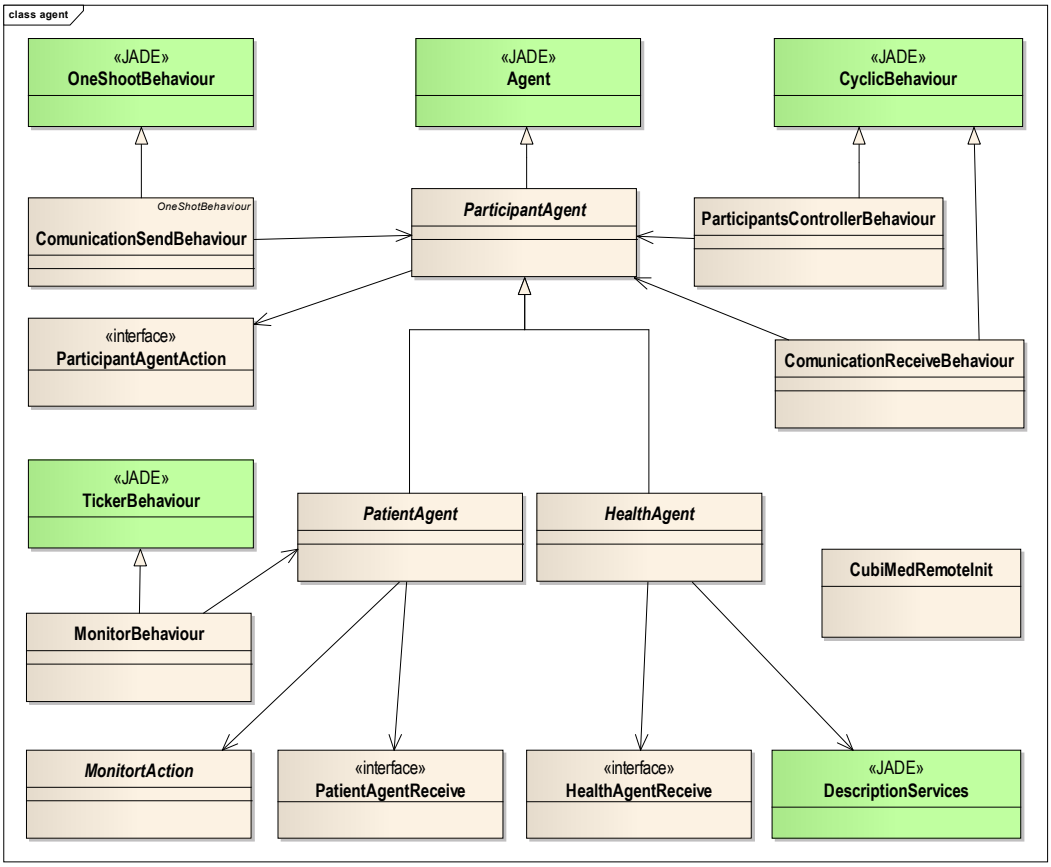

Figura 16: Diagrama de classes do pacote Client, CubiMed

Este pacote contém a classe *ParticipantAgent,* que consiste em uma classe abstrata que herda as funcionalidades da classe Agent do JADE. Esta classe tem implementada as funcionalidades genéricas que um agente precisa ter para estabelecer uma comunicação com o servidor do CubiMed, o que consiste em: especificação da ontologia e a solicitação de autenticação para participar da aplicação. Nesta classe, também são especificados os comportamentos que um participante precisa ter e são representados com as classes: *ParticipantControllerBehaviour*, *ComunicationReceiveBehaviour* e *ComunicationSendbehaviour*. O *ParticipantControllerBehaviour* é um comportamento cíclico que espera continuamente por mensagens do agente coordenador para ser informado sobre o ingresso ou saída de um agente da aplicação. Uma vez que algum desses eventos ocorre, ele chama os métodos

implementados pelo desenvolvedor, usando-se a interface *ParticipantAgentAction,* o qual é enviado como parâmetro no momento da criação de um *ParticipantAgent.* O *ComunicationReceiveBehaviour* também é um comportamento cíclico que espera o recebimento de alguma mensagem de algum outro agente. Por último, o *ComunicationSendBehaviour* é um comportamento simples, usado quando se deseja enviar algum tipo de informação para os outros agentes. Dentro destes comportamentos, são usadas como instâncias as classes criadas no pacote *Ontology,* que servem para definir a ação que está sendo realizada e o conteúdo que se está enviando e recebendo.

Conforme mencionado, a classe *ParticipantAgent,* implementa todas as partes comuns entre os participantes. Contudo, são criados o *PatientAgent* e o *HealthAgent* para implementar características especificas tanto do paciente como dos membros do staff médico. *PatientAgent* é a classe que um desenvolvedor deve estender para criar aplicações colaborativas dirigidas ao paciente. Esta classe tem implementados todos os métodos necessários para executar ações do tipo *ToConsult* e *ToInform,* e fazendo uso do comportamento *ComunicationSendBehaviour* adicionado na classe pai, pode-se enviar informações do tipo *Sintoma* e *Message,* para os outros agentes. Além disso, este agente recebe como parâmetro uma instância do tipo *PatientAgentReceive,* que é uma interface que permite ao desenvolvedor implementar os métodos que serão executados cada vez que se receba uma mensagem a partir do comportamento *ComunicationReceiveBehaviour* que foi adicionado na classe pai. Por último, esta classe adiciona o comportamento mais importante, que é o comportamento *MonitorBehaviour*. Este comportamento está ligado à classe *MonitorAction,* onde o desenvolvedor pode especificar que tipo de monitoramento irá realizar e em quanto tempo. Por último, fazer toda a lógica para pegar as informações a partir de sensores e enviá-las com o formato adequado de mensagens para os agentes da equipe médica.

O *HealthAgent* é o tipo de agente instanciado quando uma aplicação colaborativa é criada, para alguma pessoa da equipe médica. Da mesma forma que o *PatienAgent,* tem toda a estrutura já implantada para enviar as mensagens com o formato adequado, podendo-se usar os quatro tipos de ações: *ToDiagnosticate, ToConsult, ToInform* e *ToInstruct.* Do mesmo modo que no paciente, esta classe receberá como parâmetro de inicialização uma interface, que será do tipo HealthAgentReceive para receber informações a partir do comportamento instanciado no pai. Esta classe terá também como parâmetro

uma lista de *ServiceDescription* do JADE, os quais são usados para especificar todos os serviços que o agente fornece.

Como última classe deste pacote, apresenta-se, de forma isolada, a classe *CubiMedRemoteInit*. Esta é uma classe estática que permite fazer a ligação com o servidor e inicializar o agente. Para isto, é preciso passar as informações de endereço ip e porta do servidor principal com o qual deseja se conectar.

3.3.3. Implementação CubiMed-LEAP

O CubiMed-LEAP é a versão para dispositivos móveis do framework CubiMed apresentada neste trabalho. Está baseado na extensão do framework JADE denominado JADE-LEAP, o qual foi criado com o objetivo de rodar agentes em dispositivos com limitações como *smartphones* e *tablets*. Para sua implementação, foram criados três pacotes que são: *Ontology*, *Init* e *Client*. Dentre estes pacotes, já foi explicado o *Ontology,* o qual não apresenta nenhuma mudança nesta implementação. Por esse motivo, explicaremos os pacotes *Init* e *Client*, lembrando que o último já foi explicado no capítulo anterior e que serão explicadas apenas as variações presentes nesta implementação.

Init: O pacote Init do CubiMed-LEAP tem o objetivo especifico de inicializar agentes no dispositivo móvel e permitir que possam se comunicar com o servidor. É preciso notar que a implementação do framework está usando o modo Split do JADE-LEAP, o qual foi explicado na seção 2.5 e que consiste em criar uma instância do agente no dispositivo com um clone no servidor, permitindo que o maior processamento seja feito no servidor e poupando os recursos do dispositivo. Nesse sentido, seguem abaixo o diagrama de classes do pacote e a explicação do seu funcionamento.

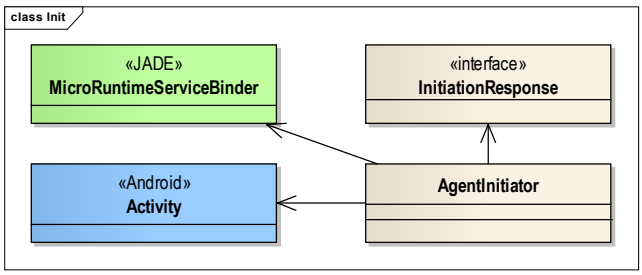

Figura 17: Diagrama de classes do pacote Init, CubiMed-LEAP

Neste pacote, é apresentada a classe *AgentInitiator,* que permite que seja criada uma instância do agente no dispositivo. Esta classe recebe como parâmetro de inicialização uma instância do tipo *InitiationResponse*, que é uma interface com métodos que serão executados quando a criação do agente tenha sido satisfatória ou tenha ocorrido um erro. No diagrama, é possível observar que o *AgentInitiator* possui instâncias do *MicroRuntimeServiceBinder* classe do JADE-LEAP que a partir do endereço ip e da porta de execução do servidor, permite realizar uma conexão. Por último, pode-se ver que existe uma referência para a classe *Activity* pertencente ao Android. Com isto, é possível recuperar o contexto de execução do agente para acessar as permissões de envio e recepção de mensagens via internet.

Client: Como foi mencionado antes, este pacote tem a mesma funcionalidade da implementação CubiMed, mas, para que possa ser utilizado em dispositivos móveis, são implementadas pequenas variações. Para explicar isso, na figura 17, é apresentado o diagrama de classes, juntamente com a descrição do funcionamento.

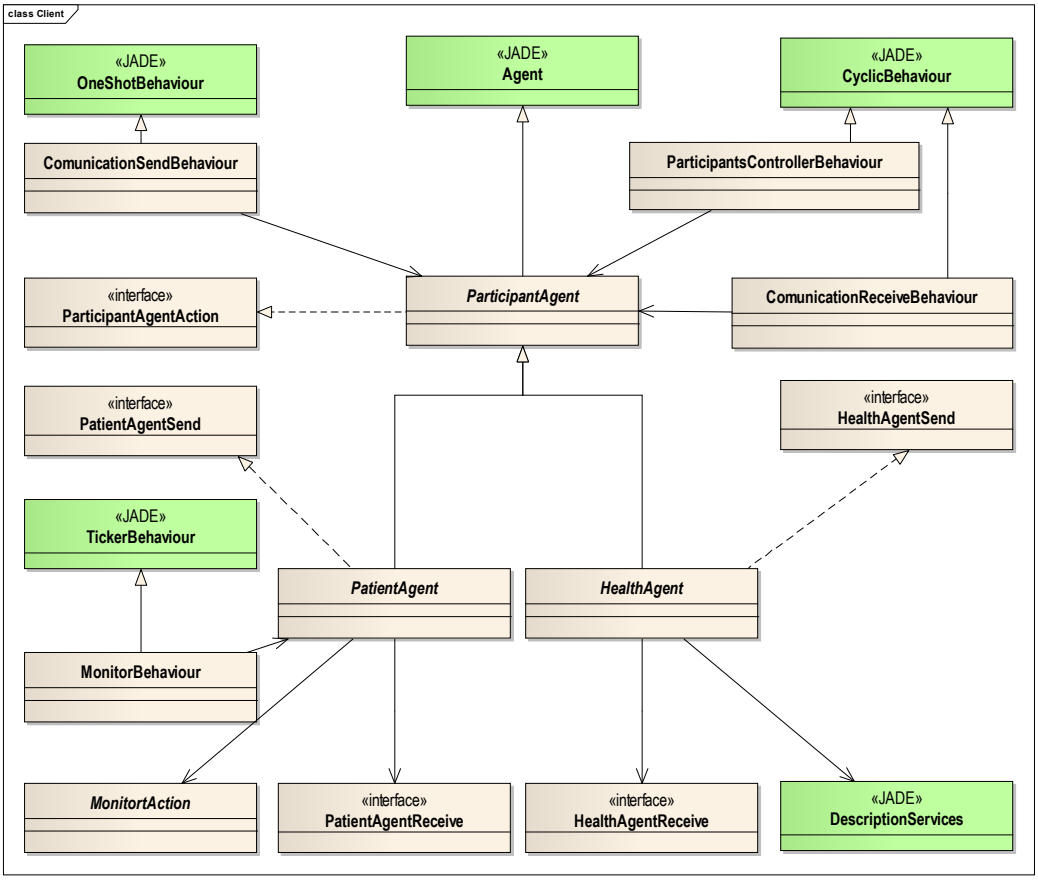

Figura 18: Diagrama de classes do pacote Client, CubiMed-LEAP

A estrutura do *ParticipantAgent* continua sendo a mesma da implementação CubiMed, não apresentando variações. Desse modo, continua implementando os métodos genéricos necesários para conseguir o diálogo com o servidor e os comportamentos que permitem realizar todas as funções anteriormente descritas. O *PatientAgent* e o *HealthAgent* também continuam tendo a mesma estrutura, mas, nesta versão, para que possam executar os métodos de envio de mensagens, precisam implementar as interfaces *PatientAgentSend* e *HealthAgentSend,* respectivamente. Isso é necessário por estarmos usando o modo de execução *Split* do JADE-LEAP, não sendo possível obter uma instancia do agente criado caso não se obtenha uma instância da classe *AgentController.* Por meio do método *getO2AInterface,* a classe AgentController permite recuperar a instância da interface implementada e executar os métodos que foram implementados.

3.4. Pontos Fixos e Flexíveis

No trabalho (MARKIEWICZ & DE LUCENA, 2001), afirma-se que um framework possui pontos fixos e pontos flexíveis. Os pontos fixos, também denominados *"frozenspots",* são pontos imutáveis presentes em um framework que consistem em porções de código já implementadas e que não podem ser alteradas na implementação de uma aplicação. Por outro lado, os pontos flexíveis, também chamados de *"hotspots"*, são descritos como classes ou métodos abstratos que devem ser implementados, podendo ter uma implementação diferente para cada instancia que o desenvolvedor desejar implementar.

Nesse sentido, a seguir são listados os pontos fixos e flexíveis que o framework CubiMed apresenta.

Pontos Fixos:

- Comportamentos do *ParticipantAgent, PatientAgent e HealthAgent.*
- Métodos genéricos para estabelecer comunicação entre as aplicações colaborativas e o servidor principal.
- Métodos genéricos para usar a ontologia criada e estabelecer diálogo entre os agentes.

- Ações dos agentes que estão criadas segundo o ciclo básico de atendimento ao paciente.
- Predicados que representam os estados dos agentes.

Pontos Flexíveis:

- Implementação das ações que um agente deve executar depois de receber uma informação, consulta, instrução ou diagnóstico.
- Implementação das ações que precisam ser realizadas quando for detectado que um agente entrou ou saiu da aplicação.
- Implementação dos métodos de autenticação para que um agente possa ser aceito na aplicação.
- Implementação dos métodos que listam os participantes que estão presentes no sistema. Caso seja necessário, o desenvolvedor poderá restringir o ponto de visibilidade de um determinado agente.
- Implementação dos procedimentos que devem ser executados pelo *MonitorBehaviour,* para obter dados do contexto do paciente.
- Extensão dos conceitos mensagem, instrução, doença e sintoma para enviar a informação que se fizer necessária.