UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

LICENCIATURA EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Ramo de Inteligência Artificial

Trabalho Final de Curso

Plataforma para Desenvolvimento de Aplicações baseadas em Agentes

Relatório Final

Autores:
Nuno Alexandre Meira - 40199
Ivo Conde e Silva – 36480

Orientador:
Professor Alberto Silva

Índice

Índice de Figuras

Página deixada propositadamente em branco

1 Sumário

Pretende-se com este trabalho desenvolver uma plataforma para desenvolvimento de aplicações baseadas em agentes de software. Partindo de uma concepção de agente entendido como um objecto rico (numa definição simplificada), o desenvolvimento de aplicações baseadas em agentes prende-se a um novo paradigma de programação. Este paradigma, em que o agente é o conceito nuclear pode ser visto como uma evolução natural do paradigma de programação orientado a objectos.

As aplicações baseadas em agentes correm por cima de uma máquina virtual de suporte de agentes. A plataforma desenvolvida neste trabalho tem como base o sistema de suporte de agentes AgentSpace, e constitui uma extensão a este na medida em que fornece um conjunto de mecanismos adicionais que facilitam a construção de aplicações baseadas em agentes.

O desenvolvimento da plataforma implicou, numa fase inicial, a identificação, dentro das características que um agente pode ter, aquelas que são mais prementes dadas as próprias características do tipo de aplicação que esta plataforma pretende proporcionar, em que a Internet é um contexto essencial a ter em conta e pretendendo-se consequentemente para os agentes de software qualidades como a mobilidade, a autonomia e a comunicação. Neste sentido foi feito um trabalho inicial sobre padrões de software focalizados para agentes, que culminou com a elaboração do artigo “A Set of Agent Patterns for a More Expressive Approach”, escrito em colaboração com o Professor orientador Alberto Silva e apresentado por este na conferência EuroPlop 2000 (European Conference on Pattern Languages of Programming).

Para a plataforma foram então implementados alguns dos padrões estudados, tais como o padrão Itinerary, Courier, Receptionist, Secretary e Session. Como consequência da implementação destes padrões, foi ainda implementado a classe de agentes AgentSapiens, uma versão enriquecida da classe Agent já disponível na plataforma AgentSpace, e ainda alguns agentes que derivam deste, como o agente Receptionist, agente Courier, agente Secretary. A plataforma apresenta assim uma hierarquia de agentes úteis para simplificar o trabalho dos programadores, já que conjuntamente com os padrões, estes agentes permitem gerar soluções a problemas recorrentes no desenvolvimento deste tipo de aplicações.

Além dos padrões foram implementados outros mecanismos úteis para o desenvolvimento de aplicações baseadas em agentes, nomeadamente:

- Três agentes genéricos de sistema que providenciam funcionalidades genéricas: o MailAgent, para envio de email, o ReminderAgent, para calendarização de mensagens e o ReceptionistAgent, como um agente facilitador que permite que agentes encontrem outros agentes.

- Mecanismos para o desenvolvimento de interfaces html para controlo dos agentes, que permite assim bem como uma biblioteca para desenvolvimento de páginas html em java que facilita os mecanismos necessários para o contacto com os agentes.

Por fim foram elaboradas duas aplicações que demonstram as funcionalidades oferecidas pela plataforma, a aplicação myGlobalNews e a aplicação AgentSpace PhotoPaint. A primeira é uma aplicação de recolha e fornecimento de notícias personalizadas aos seus utilizadores. A segunda é uma aplicação que ilustra a procura e compra de imagens na internet usando agentes de software.

Página deixada propositadamente em branco

2 Introdução

Existe nos dias de hoje uma necessidade crescente de se desenvolver software cada vez maior e mais complexo. Essa necessidade verifica-se nos mais diferentes domínios, desde a necessidade das empresas ligarem os seus sistemas aos dos seus fornecedores e clientes, aos sistemas distribuídos que operam na Internet e intranets, à globalização dos sistemas de software de grandes empresas, entre outros.

As aplicações baseadas em agentes constituem um novo paradigma de programação para uma grande maioria dos programadores e engenheiros de software [18]. Apesar dos programadores estarem familiarizados com ferramentas de abstracção como funções, procedimentos, métodos ou objectos o conceito de agente de software é uma poderosa abstracção para visualizar e estruturar software cada vez mais complexo. É nossa convicção de que este paradigma ainda não foi devidamente explorado e que as suas vantagens ainda não foram totalmente aproveitadas.

Por ser ainda hoje matéria de investigação a muitos níveis da teoria de agentes não existem por isso muitos ambientes de desenvolvimento de aplicações baseadas em agentes (ABA). Apesar da plataforma de desenvolvimento que nos propomos fazer não ser "a solução" (nenhuma plataforma o é), ela é sem dúvida um contributo para facilitar o desenvolvimento de ABA aos programadores e engenheiros de software. A necessidade de plataformas, essa existe e existirá enquanto houver a necessidade de construir software cada vez mais complexo e consequentemente a necessidade de recorrer a paradigmas de programação que facilitem esse desenvolvimento.

2.1 Conceitos Básicos

Intuitivamente, um agente pode ser entendido como uma entidade a quem o utilizador desse agente delega uma tarefa para este realizar. Embora existam múltiplas definições de diferentes autores do que um agente deve ser, uma das definições mais compreensíveis é a apresentada por Wooldrige e Jennings e retractada sumariamente em [3] (págs. 70-73). Resumidamente, um agente deverá possuir, numa noção fraca de agente, características básicas como a de autonomia, sociabilidade, reactividade, pró-actividade e persistência. Outras características são igualmente desejáveis, no sentido em que fortificam a noção de agente, tornando-o mais inteligente. Exemplos de tais características são a mobilidade, a intencionalidade, a aprendizagem, a veracidade, a racionalidade etc.

Em regra geral um agente não existe por si só. Encontra-se integrado num sistema ou numa aplicação. Um sistema de suporte de agentes (SSA) é uma plataforma (que pode ser construída sobre o hardware, um sistema operativo ou ainda uma máquina virtual) que providencia um ambiente computacional de execução/interpretação de agentes [3] (pág. 97). Além destes outros mecanismos podem ser providenciados, tais como a comunicação, persistência, navegação, segurança e outros.

O terceiro e principal conceito com que este projecto lida é o da definição do que deve ser uma aplicação baseada em agentes (ABA). A definição aqui seguida é aquela apresentada em [4] e que aqui reiteramos: Uma ABA é uma aplicação dinâmica, potencialmente distribuída em larga escala num contexto aberto e heterogéneo, onde a unidade conceptual de desenho e desenvolvimento das mesmas é o agente de software. Existem um conjunto de características e requisitos que definem uma ABA:

- *Autonomia* – Cada utilizador cria e mantém os seus agentes usando recursos próprios e/ou de outros.

- *Heterogenia* – Diversidade de arquitecturas, interfaces, linguagens de programação, pacotes de comunicação etc.

- *Abertura* – Agentes deverão ser capazes de interagir com entidades externas como por exemplo bases de dados e outros sistemas de informação.

- *Dinamismo* – Os agentes poderão ser adicionados, removidos e actualizados a qualquer altura. Os agentes deverão ter mecanismos que lhes permitam lidar com situações novas e/ou outras características variáveis.
- *Robustez* - Agentes terão que ser capazes de lidar com falhas nas máquinas, problemas de comunicação, etc.

- *Segurança* – O SSA deverá suportar diferentes perfis de segurança aos agentes e estes devem actuar em conformidade com estes mecanismos.

2.2 Objectivos do trabalho

A construção de uma aplicação baseada em agentes não é simples. Requer o conhecimento das mais variadas áreas tecnológicas: protocolos e métodos de comunicação em rede, representação do conhecimento, aprendizagem, planeamento, raciocínio, e muitas outras. Apesar dos SSA providenciarem já alguns mecanismos que permitem que o programador se abstraia de alguns detalhes existem ainda numerosas funcionalidades e situações recorrentes que os programadores desejariam ver a sua implementação facilitada e que lhes permitisse lidar com os seus agentes segundo uma noção cada vez mais forte bem como dotar as ABA das características atrás mencionadas.

Este projecto lida por isso com o problema de definir uma plataforma de desenvolvimento de aplicações baseadas em agentes que permita facilitar o desenvolvimento das mesmas por parte dos programadores. A plataforma deverá recorrer por isso a um SSA conhecido e estendê-lo por via da definição e implementação de padrões de agentes, providenciando assim ferramentas poderosas para o programador desenvolver a sua aplicação [7],[22],[27].

2.3 Contexto do trabalho

A implementação da plataforma de desenvolvimento de ABA do qual este projecto é objectivo baseia o seu trabalho no Sistema de Agentes AgentSpace [12] desenvolvido pelo Professor Alberto Silva [3],[4]. O diagrama em baixo ilustra as várias componentes envolvidas no ambiente de desenvolvimento das aplicações baseadas em agentes que propomos implementar. As componentes de Gestão , Suporte e AS-API definem a infra-estrutura AgentSpace como ela está correntemente implementada.

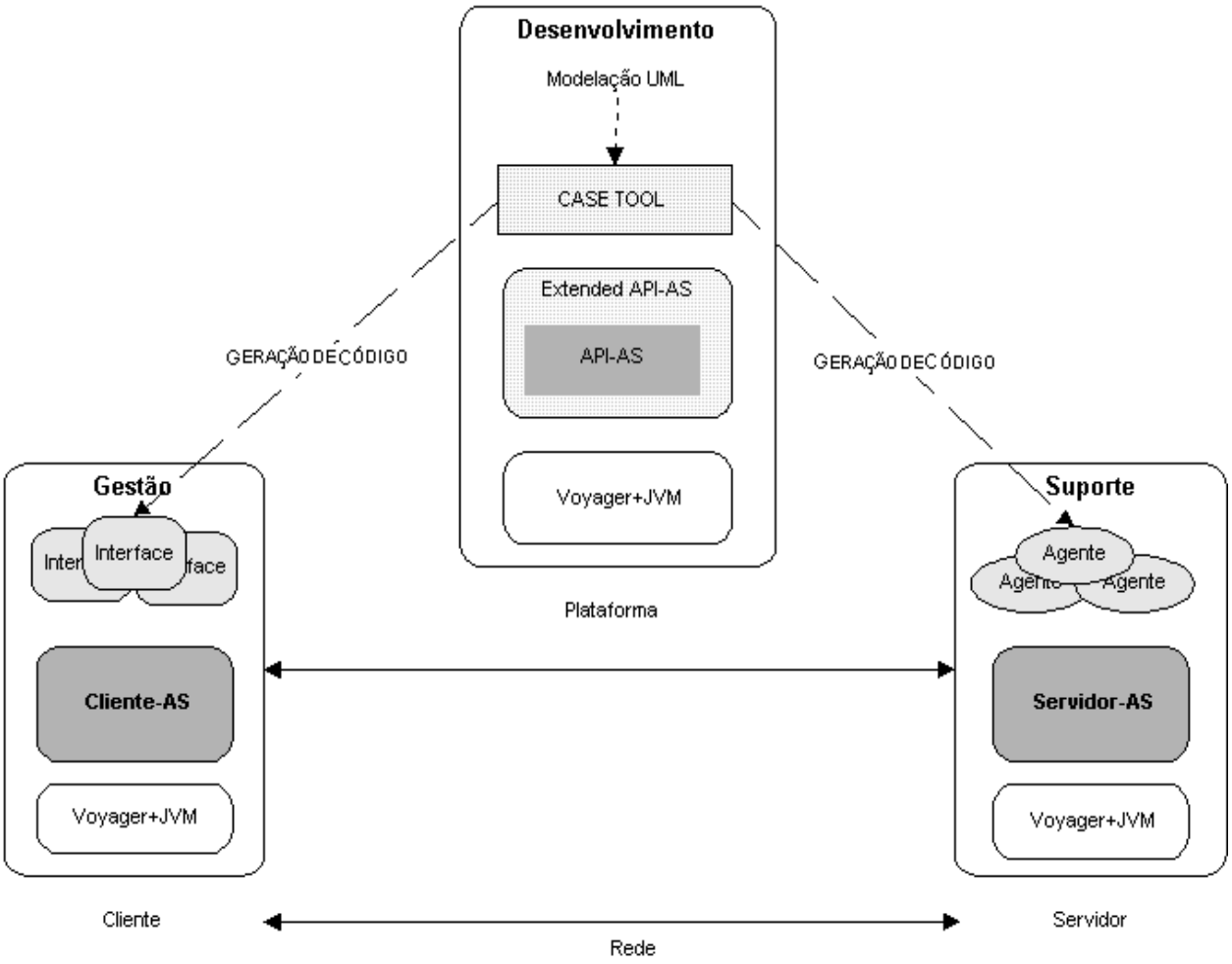

Figura 1 - Arquitectura AgentSpace

O servidor AgentSpace (Servidor-AS) consiste num processo Java, de múltiplas actividades, no qual os agentes são executados. O Servidor- AS providencia vários serviços, designadamente o de criação de agentes e locais de execução, execução de agentes, persistência, controlo de acesso, suporte à mobilidade e comunicação de agentes, , geração de identificadores , e uma shell de monitorização e gestão.

A API-AS consiste em uma biblioteca de classes e interfaces que definem regras de como se desenvolvem (classes de) agentes, em particular, e aplicações baseadas em agentes em geral. A API suporta o programador na criação de classes de agente que são criadas e armazenadas no Servidor-AS para posterior utilização e de classes de applets específicos de forma a providenciar uma interface de utilizador, para gestão remota de agentes.

O cliente AgentSpace (Cliente-AS) consiste numa aplicação java que permite a gestão e monitorização de agentes e outros recursos existentes, de forma integrada com a Web/Internet. Oferece a qualquer utilizador a possibilidade de gerir facilmente os seus próprios recursos mantidos em um ou mais Servidor-AS.

As componentes cliente e servidor AgentSpace são executadas sobre a máquina virtual Java e utilizam funcionalidades providenciadas pelo Voyager [19]. O Voyager é um ORB (Object Request Broker) Java que permite o desenvolvimento genérico de aplicações baseadas no

modelo de objectos distribuídos. Ambas as componentes podem ser executadas na mesma máquina ou em máquinas diferentes.

Os agentes são executados sempre no contexto de um Servidor-AS. Por outro lado estes interagem com os utilizadores através de applets (específicos ou genéricos) ou frames executados no contexto de clientes Web.

2.4 Organização do documento

Feita uma introdução aos conceitos básicos, objectivos e contexto do trabalho, os capítulos que se seguem descrevem o trabalho realizado propriamente dito.

Desta forma, no capítulo 3 é apresentada uma visão geral da plataforma de desenvolvimento e feita a descrição de como esta está organizada.

No capítulo 4 são introduzidos e descritos os padrões de software para agentes que foram objecto de estudo e realização no trabalho. É apresentado neste capítulo uma definição de padrões de software, apresentada a motivação para o seu uso na plataforma e feita uma classificação destes padrões.

No capítulo 5 são apresentados os agentes desenvolvidos. Começa-se por destacar a classe de agentes AgentSapiens como a entidade agregadora das diferentes funcionalidades oferecidas pela plataforma e ponto central de desenvolvimento das aplicações baseadas em agentes. São ainda descritos os agentes de sistema MailAgent, ReminderAgent e ReceptionistAgent, como agentes genéricos prontos-a-usar que facilitam o desenvolvimento de aplicações dadas as funcionalidades que oferecem.

No capítulo 6 é apresentada a AgentSpace Servlet, que possibilita o desenvolvimento de interfaces html e interacção dos utilizadores com os agentes através de um web browser. São ainda apresentados mecanismos de apoio ao desenvolvimento de interfaces html que facilitam a componente de comunicação com os agentes, bem como classes de apoio ao desenvolvimento de interfaces java para os agentes.

No capítulo 7 são referenciadas duas aplicações desenvolvidas que demonstram as funcionalidades oferecidas pela plataforma descritas nos capítulos anteriores.

Por fim é apresentado um capítulo de conclusões sobre o trabalho realizado bem como sugestões para o desenvolvimento de trabalho futuro que enriqueceria ainda mais a plataforma aqui apresentada.

3 DevAPI: Visão Geral

3.1 Introdução

A plataforma para desenvolvimento de aplicações baseadas em agentes, que aqui denominamos de DevAPI, constitui uma extensão aos mecanismos de desenvolvimento já existentes na plataforma de agentes AgentSpace. Os "blocos" constituintes da plataforma desenvolvida podem ser caracterizados segundo duas grandes vertentes de desenvolvimento. Por um lado, um conjunto de mecanismos que geram soluções, concretizado no estudo e desenvolvimento de padrões de software para agentes e por outro o desenvolvimento de alguma soluções finais, isto é , que não carecem de desenvolvimento por parte do programador, de que são exemplo os agentes de sistema MailAgent, ReminderAgent e ReceptionistAgent e a AgentSpace Servlet.

A hierarquia de packages descrita na secção seguinte dá a entender as áreas de desenvolvimento que estão abrangidas pela plataforma:

Definição de uma hierarquia de agentes genéricos, cujas classes se encontram no package *agentsapiens* (e seus sub-packages) e onde se destacam a classe mais genérica AgentSapiens e outras classes de agentes que oferecem funcionalidades mais específicas, como Secretary e Receptionist na área dos agentes estáticos e Courier e SecretaryCourier na área dos agentes móveis. Uma descrição mais detalhada destas e outras classes referentes aos agentes citados pode ser encontrada no capítulo 4.

Mecanismos de comunicação, cujas classes de desenvolvimento se encontram no package *sessions,* onde se destacam classes para as sessões de comunicação entre agentes genéricas - Session e KQMLSession (ver capítulo 4, secção 3) - e ainda algumas sessões mais específicas que implementam a comunicação com os agentes de sistema desenvolvidos - MailSession, ReminderSession e RecepionistSession (ver capítulo 5, secções 2, 3 e 4) - ou ainda com a AgentSpace Servlet, como a classe ServletSession (ver capítulo 6, secção 3).

Mecanismos de gestão e execução de tarefas, cujas classes de implementação se encontram no package *tasks* (ver capítulo 4 secções 1 e 3).

No package *faces* encontram-se classes para o desenvolvimento de interfaces html e java (ver capítulo 6, secções 3 e 4).

No package *system* podem-se encontrar os agentes de sistema MailAgent, ReminderAgent e ReceptionistAgent (ver capítulo 5, secções 2, 3 e 4).

Por fim, encontram-se na package *Object* algumas classes "avulsas" na sua maioria, representando objectos que são passados na comunicação entre agentes, objectos que facilitam a interacção dos agentes com bases de dados, entre outras.

A documentação mais exaustiva de todas as classes da DevAPI pode ser encontrada no apêndice de Documentação Java, em anexo a este relatório.

3.2 Organização (Packages)

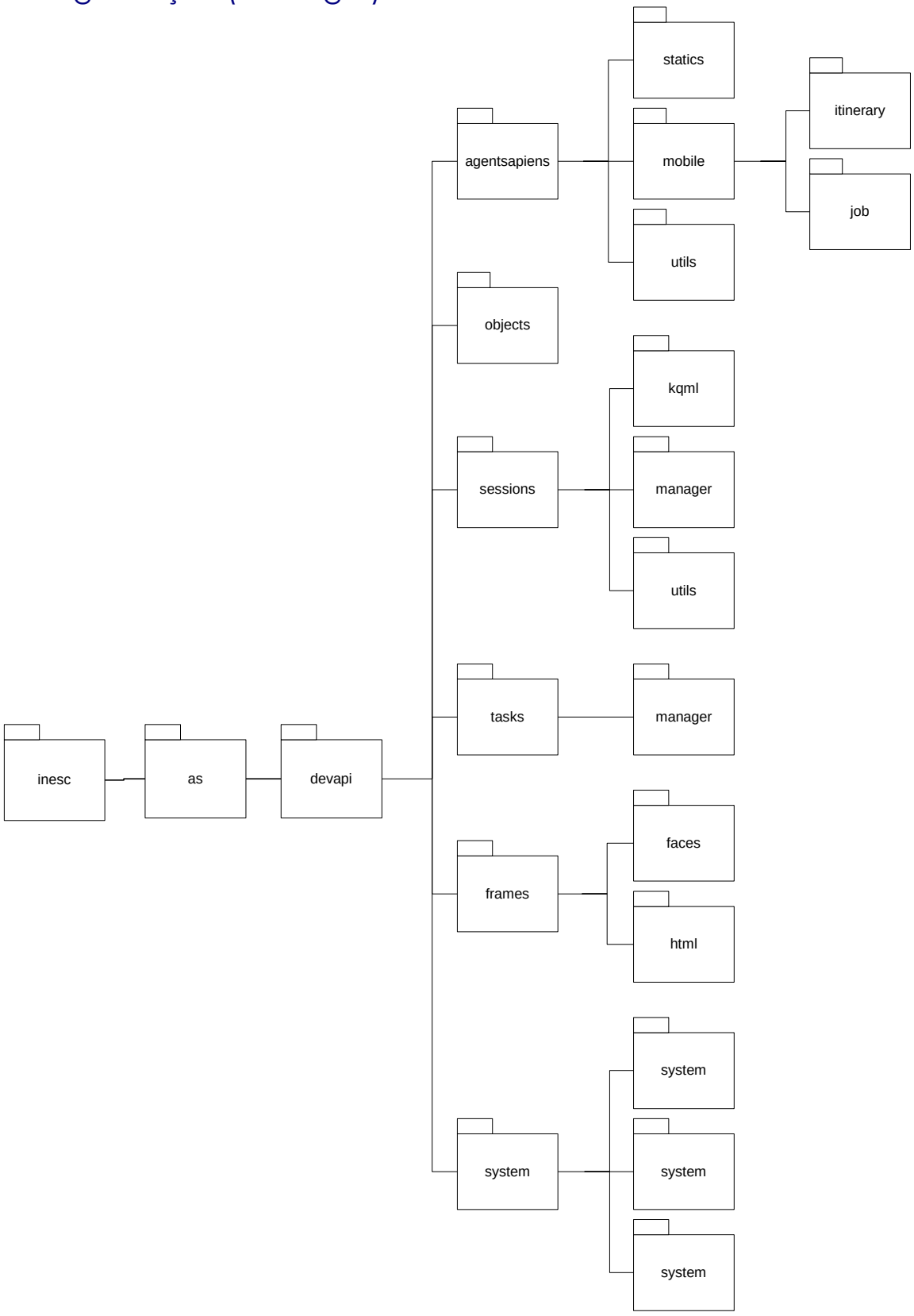

Figura 2 - Packages DevAPI

4 DevAPI: Padrões de Software

4.1 Introdução

4.1.1 Definição

Apesar de haver variadas definições, optamos pela do Alexander´s book, " The Timeless Way of Building" [22], que é a seguinte:

- Um padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um problema e uma solução.
- Cada padrão é uma relação entre um certo contexto, um certo sistema de forças que ocorrem repetidamente nesse contexto, e uma certa configuração espacial que permite que estas forças se resolvam por si só.
- Um padrão é uma instrução, que mostra como a configuração espacial pode ser utilizada, repetitivamente, para resolver o dado sistema de forças, onde quer que seja que o contexto o faça relevante.
- O padrão é, de uma maneira resumida, ao mesmo tempo uma coisa, que ocorre no mundo, e a regra que nos diz como criar essa coisa, e que quando deve ser criada. É tanto um processo como um coisa; tanto uma descrição de uma coisa que está viva, e uma descrição de um processo que irá gerar essa coisa.

4.1.2 Motivação

É ponto assente para os engenheiros de software que existem no desenho de sistemas de software múltiplos problemas e respectivas soluções que são recorrentes. Estes não estão interessados em reinventar constantemente a roda, rescrevendo por vezes soluções para os problemas que estão longe das ideais. Os padrões além da sua capacidade generativa [1] também permitem encapsular detalhes , porque estes ou não interessam ao seu utilizador ou então porque se encontram num domínio que vai para lá do seu conhecimento. Num contexto como o das aplicações baseadas em agentes esta encapsulação tem uma relevância ainda maior, já que estas aplicações podem lidar com múltiplas áreas da ciência da computação simultaneamente (comunicação, navegação, gestão de tarefas, raciocínio, etc.).

Os padrões de agentes permitem resolver um conjunto de problemas existentes no paradigma de programação de agentes, entre os quais:

- contribuem para a definição de um agente, minimizam o esforço dos programadores.
- facilitam a identificação e especificação de abstracções acima do nível de agentes simples.

- providenciam um vocabulário comum, reduzem a complexidade dos problemas já que se pode pensar os problemas em termos dos objectivos e soluções para os mesmos sem ter que se preocupar de como o fazer (os padrões encapsulam essa informação).
- contribuem para a modularidade das aplicações, facilitando o seu desenvolvimento e manutenção.
- podem ser utilizados sem requerem conhecimento específico, truques ou domínio da linguagem de programação especiais.

4.1.3 Classificação

Os diferentes padrões podem ser estruturados segundo a seguinte classificação:

- *Agentes* – padrões que lidam com a arquitectura de agentes e ABA. Exemplos de padrões deste género podem-se encontrar em [3] e [9].
- *Comunicação* – lidam com a forma como os agentes comunicam entre si.
- *Navegação* – lidam com aspectos de gestão da navegação dos agentes e qualidade dos serviços.
- *Tarefa* – lidam com a divisão e delegação de tarefas pelos agentes.
- *Interacção* – lidam com a localização de agentes e facilitação da sua interacção.
- *Coordenação* – lidam com a gestão de dependências entre as actividades dos agentes.

De notar que um padrão de software não tem que necessariamente ser classificado por um e só um nível. De facto, existem padrões que podem ser classificados segundo mais do que um nível. Um padrão pode ser considerado por exemplo como sendo de comunicação e de interacção.

4.1.4 Descrição do Formato de Documentação

O formato que iremos utilizar, é um formato mínimo proposto por D. Deugo [1] composto pelos seguintes itens:

- **Nome** - Identificação do padrão segundo um nome representativo.
- **Problema** - Definição do problema a ser resolvido pelo padrão.
- **Contexto** - Descrição de uma ou mais situações em que o padrão é aplicável.

- **Forças** - Descrição dos factores que podem influenciar a decisão de quando o padrão deve ser aplicado.

- **Solução** - Descrição da solução que o padrão vai gerar.

Após alguma pesquisa, em que encontrámos alguns padrões descritos e interessantes, que apesar de conterem descrições razoavelmente detalhadas e seguirem formatos standard de desenho de padrões, concluímos que lhes faltam alguma expressividade. Afinal há padrões, para os quais existem boas analogias e metáforas do mundo real que podem ajudar a dar uma forma mais definida e compreensível a esses padrões. Introduzimos por isso mais quatro itens que entendemos convenientes:

- **Introdução** – Descrição introdutória ao padrão com recurso a metáforas e/ou analogias.

- **Classificação** – Classificação do padrão.

- **Implementação** - Descrição da implementação do padrão com exemplo de utilização.

- **Referências** - Indicação das referências biográficas e/ou padrões que inspiraram o padrão definido.

4.2 Padrão Itinerary/Courier agent

- **Introdução**

  Quando viajamos, quer em trabalho ou lazer, viajamos segundo um determinado itinerário, que está á partida pré-definido, ou que vamos definindo á medida que vamos viajando. Neste itinerário temos lugares, uns a que temos mesmo de ir, outros não, alguns segundo uma certa ordem e outros nem por isso.
  De qualquer maneira, independentemente dos lugares aonde vamos e o que lá vamos fazer, ou seja podemos olhar o itinerário e a maneira como o vamos percorrer como uma situação recorrente, ou seja como um padrão. (ex.: ver se posso e quando posso fazer uma viagem para um determinado lugar, comprar os bilhetes viajar, fazer reservas, decidir qual a próxima viagem ,etc.)

  De certa forma este padrão também poderá estar associado a uma componente de planeamento.

- **Classificação**

  Navegação, Tarefa, Agente.

- **Nome:**

  Itinerary/Courier agent

- **Problema:**

  Como tratar um percurso que se tem de percorrer de modo a que se possam realizar determinadas tarefas.

- **Contexto:**

  Está-se a desenvolver um sistema de agentes móveis, e como tal existem agentes com tarefas que terão ou poderão ter de ser realizadas em lugares diferentes. O cenário tipo é um agente que tem de ir a vários lugares e realizar em cada um determinadas tarefas e como tal existe um percurso associado.

- **Forças:**

  - Os agentes móveis mudam de lugares.
  - As tarefas a realizar num determinado lugar devem ser definidas e realizadas independentemente do lugar de onde o agente veio e para onde vai.
  - Tratar a complexidade do percurso separadamente da complexidade das tarefas.
  - Agentes moveis lidam sistematicamente com percursos (onde ir a seguir, o que fazer se um determinado destino não existe ...).
  - O numero de tarefas não deve introduzir complexidade no código.

- **Solução:**

  Introduzir o conceito de “itinerário” (Itinerary). Através deste conceito poderemos tratar o problema de viajar ao longo de múltiplos destinos de uma maneira separada das tarefas, sendo depois só necessário estabelecer uma relação entre as tarefas e o lugar. Introduzir o conceito de “estafeta” (Courier) que consiste num agente que sabe iterar sobre esse Itinerário. Agentes que herdem do agente Courier poderão utilizar o objecto Itinerary e definir as tarefas que necessitam de ser executadas (associadas a um lugar), de uma maneira simples e legível, invocar um método do agente e este realizará todas a viagens necessárias e executará todas as tarefas.

- **Referencias:**

  Itinerary Pattern [27], inspirado no padrão Itinerary definido em [7].

- **Implementação:**

  Para implementar este padrão, foram então implementadas as classes Itinerary e Courier.

  O objecto Itinerary permite definir listas de tarefas, associadas ao lugar em que têm de ser executadas. Permite também definir listas de tarefas que tem de ser executadas:

  - quando começa a viajar.
  - antes de se mover para outro lugar.
  - depois de se mover para outro lugar.
  - quando acaba de viajar.

  Também permite especificar em que lugar deve terminar a viagem.

  O objecto (agente) Courier disponibiliza métodos para a adição dessas tarefas, um método para iniciar o objecto Itinerary, *newItinerary*, e um método para começar a iterar esse objecto , *goTravel*, ou seja começar a viajar.

  Este padrão implicou também a implementação dos seguintes objectos:

  DestinationPlace - que não é mais do que um Place associado a um Ticket. Quando se viaja para um determinado lugar o acesso á realização de determinadas tarefas é limitada pelo Ticket que se utilizou para ir para esse Place. Assim diferentes tarefas a serem realizadas num mesmo lugar, podem necessitar de diferentes Tickets. Este objecto estende assim o conceito de Place de modo a facilitar a iteração do objecto Itinerary, e a definição de uma tarefa.

- AgentTask - que representa o conceito de tarefa associado a um DestinationPlace.
- CallbackAgenttask - que é uma AgentTask genérica baseada no "reflected method invocation".
- TaskManager - responsável pela execução das tarefas.

De seguida é apresentado um diagrama que ilustra a relação entre as classes mencionadas.

Diagrama de classes para o padrão Itinerary/Courier

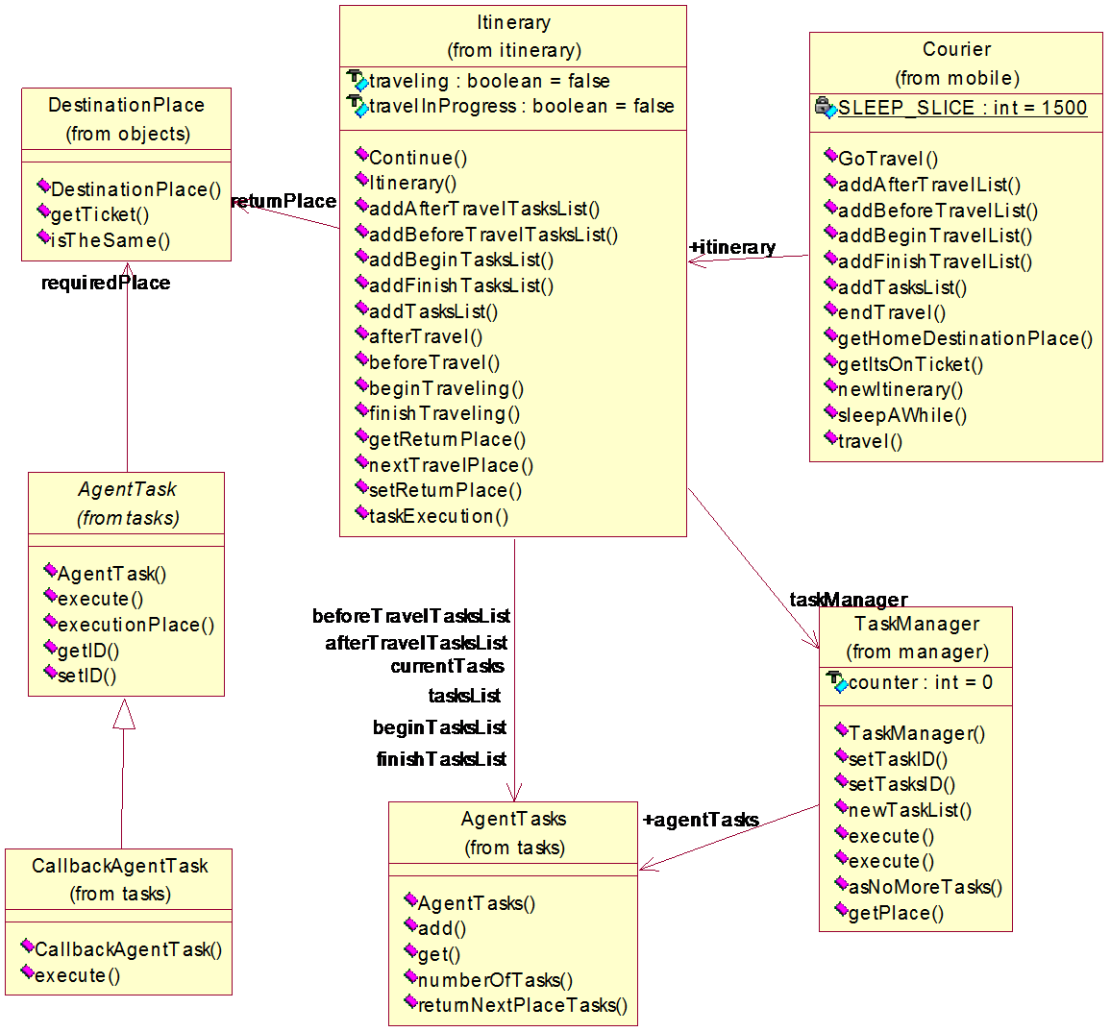

Figura 3 - Padrão Itinerary Courier

Exemplo de utilização do padrão:

Para demonstrar como utilização deste padrão, recorremos ao exemplo AgentSpace PhotoPaint application (ver apêndices em anexo), mais concretamente ao código de um agente de procura.

Para alem de Courier este tipo de agente é também um SecretaryCourier (conceito que será explicado noutro padrão), no entanto serve perfeitamente para perceber como definir um itinerário e como utilizar o padrão.

Exemplo de como definir um itinerário e iniciar a viagem para realizar as tarefas definidas:

```
public void getImagePreviewsJob(){

newItinerary();

addTasksList(new
CallbackAgentTask(photoPaintReceptionistPlace,this,"askForImageStores"));

addTasksList(new
CallbackAgentTask(photoPaintReceptionistPlace,this,"goToAllImageStores"));

addFinishTravelList(new CallbackAgentTask(this,"endJob"));

GoTravel();
}
```

Por outras palavras, o objecto Itinerary do agente Courier é iniciado, de seguida são adicionadas duas tarefas a serem realizadas no Place da Recepcionista do PhotoPaint, e mais uma tarefa a ser realizada quando o agente terminar a execução das tarefas que tem de executar e voltar ao local de onde partiu.

Exemplo de como adicionar mais tarefas durante a viagem:

```
public void goToAllImageStores(){

 for(int i=0; i < (imagesStoresIds.numberOfObjects()); i++){

    PlaceId place = new PlaceId((String) imagesStoresPlaceIds.get(i));

    addTasksList(new CallbackAgentTask(getDestinationPlace(place),

this,"askForImagePreviews",(String) imagesStoresIds.get(i)));
 }
```

Resumindo, para cada loja existente é adicionada uma tarefa a ser executada no lugar dessa loja. Este exemplo é também um exemplo da execução de uma tarefa (CallbackAgentTask). Depois da execução deste método não é necessário realizar mais nada. O agente continuará a viajar e a realizar essas tarefas.

4.3 Padrão Session

- **Introdução:**

  Normalmente quando iniciamos uma conversa com alguém num determinado lugar, ficamos nesse lugar a conversar até a conversa acabar e só depois vamos embora. Se o conteúdo de um determinada conversa for importante, não convém andar a saltitar de um lado para o outro nem fazer outras coisas ao mesmo tempo, pois convém estar com atenção. No entanto, também e possível conseguirmos ter um conversa mais descomprometida e estarmos a deslocar-nos ao mesmo tempo.
  Este tipo de comportamento, pode ser visto como um comportamento padrão (simples) ao nível da comunicação no nosso dia a dia, e pode ser visto como uma "sessão" que estamos a ter com alguém, que tem um principio, um meio e um fim.

- **Classificação**

  Comunicação

- **Nome:**

  Session

- **Problema:**

  Como é que agentes moveis ou serviços conseguem ter uma comunicação complexa entre si, de uma maneira fácil e eficiente.

- **Contexto:**

  Está-se a desenvolver um sistema de agentes móveis em que os agentes moveis interagem tanto com agentes estáticos como com moveis, em conversações complexas durante um determinado período de tempo. O cenário tipo são dois agentes que concordam em comunicar, interagem através da troca de mensagens, e finalmente param a sua comunicação.

- **Forças:**

  - Agentes móveis mudam de posição com frequência.
  - Mensagens de diferentes conversações chegam umas entre as outras numa maneira aleatória.
  - Interacções complexas envolvem muitas mensagens.
  - O desenho deve ser simples e fácil de utilizar.
  - O desenho deve restringir os agentes o menos possível.
  - A comunicação deve ser eficiente.
  - É necessário tornar possível para os agentes poderem ter conversações simultâneas e tratarem essas conversações de uma maneira simples e eficiente.

- **Solução:**

  Introduzir o conceito de "Sessão" (Session). A sessão é uma ligação de comunicação aberta entre dois agentes e que é representada pelo objecto sessão, em cada agente.

A utilização de sessões permite aos agentes resolverem varias preocupações. Em primeiro lugar, as conversações estão separadas. Agentes recebem mensagens de um especifico objecto sessão e cada pode ser atribuído manipulador (handler) para cada. Neste manipulador, o estado corrente da conversação pode ser encapsulado. E segundo lugar, é possível separar a manipulação sessão do problema da mobilidade. Podemos forçar um agente a não mudar de lugar de lugar, ou tal acção implicitamente a encerrar todas as sessões abertas. Se a troca de mensagens não constituir um problema para a mobilidade (troca de mensagens da sessão feitas com pouca frequência ou o agente move-se com pouca frequência) o agente pode mover-se em manter a sessões (sem perder o seu contexto), sem problemas. Isto permite uma comunicação directa e simples entre agentes, o que resulta num menor sobrecarregamento e melhor performance.

- **Referencias:**

  É inspirado no padrão Communication Session definido em [7], Session Pattern [27]

- **Implementação:**

  Para implementar este padrão, foram então implementadas as classes Session e SessionManager.

  A classe Session é uma classe abstracta independente de um formato predefinido da mensagem que está a ser trocada. O seu objectivo é englobar um conjunto de atributos e implementar algum comportamento.

  Como atributos temos por exemplo: o Id da Sessão, O Id do agente ou entidade com quem se está a ter a sessão, o Id da Sessão do outro agente que está a receber as mensagens desta Sessão, os objectos Login (objectos utilizados quando se quer que se faça alguma autenticação para que a Sessão seja aceite), etc.

  Como comportamento temos:

  - A troca de mensagens necessária para começar a sessão onde se faz uma autenticação para ver se a Sessão deve ser aceite ou não.
  - Verificação de se uma determinada mensagem é para esta sessão.
  - Identificação de mensagens com um formato invalido.
  - Algum encapsulamento no tratamento do envio de mensagens.
  - Identificação de algumas mensagens tipo (mensagem que: identifica fim de sessão, a resposta ao envio de uma mensagem de formato invalido, etc..
  - Autenticação de Ids das Sessões das mensagens que chegam depois da sessão já ter sido aceite.

  O objecto SessionManager que permite lidar com múltiplas sessões. Inclui:

  - O estabelecimento de novas sessões adicionadas e a geração dos seus Ids.
  - O encaminhamento de mensagens chegadas para os abjectos Sessão a que se destinam.
  - Remoção de sessões terminadas.
  - A invocação de métodos que se destinam a criar Sessões para mensagens que chegam para iniciar sessões.

  Na demonstração de um exemplo de utilização, explicaremos melhor o funcionamento destes objectos.

Este padrão implicou também a implementação dos objectos; Login (username + password), KQMLMessage (um objecto mensagem baseado no KQML) e KQMLSession (que é uma implementação da classe Session e que tem como base as mensagens do tipo KQMLmessage).
A objectivo da classe KQMLSession é ter já os métodos que a classe Session necessita que sejam implementos (que são dependentes do tipo da mensagem), facilitando assim de desenvolvimento de sessões que herdam desta e que portanto tenham com base este tipo de mensagem.

O diagrama de classes que se segue resume as relações entre as classes.

Diagrama de classes para o padrão Session

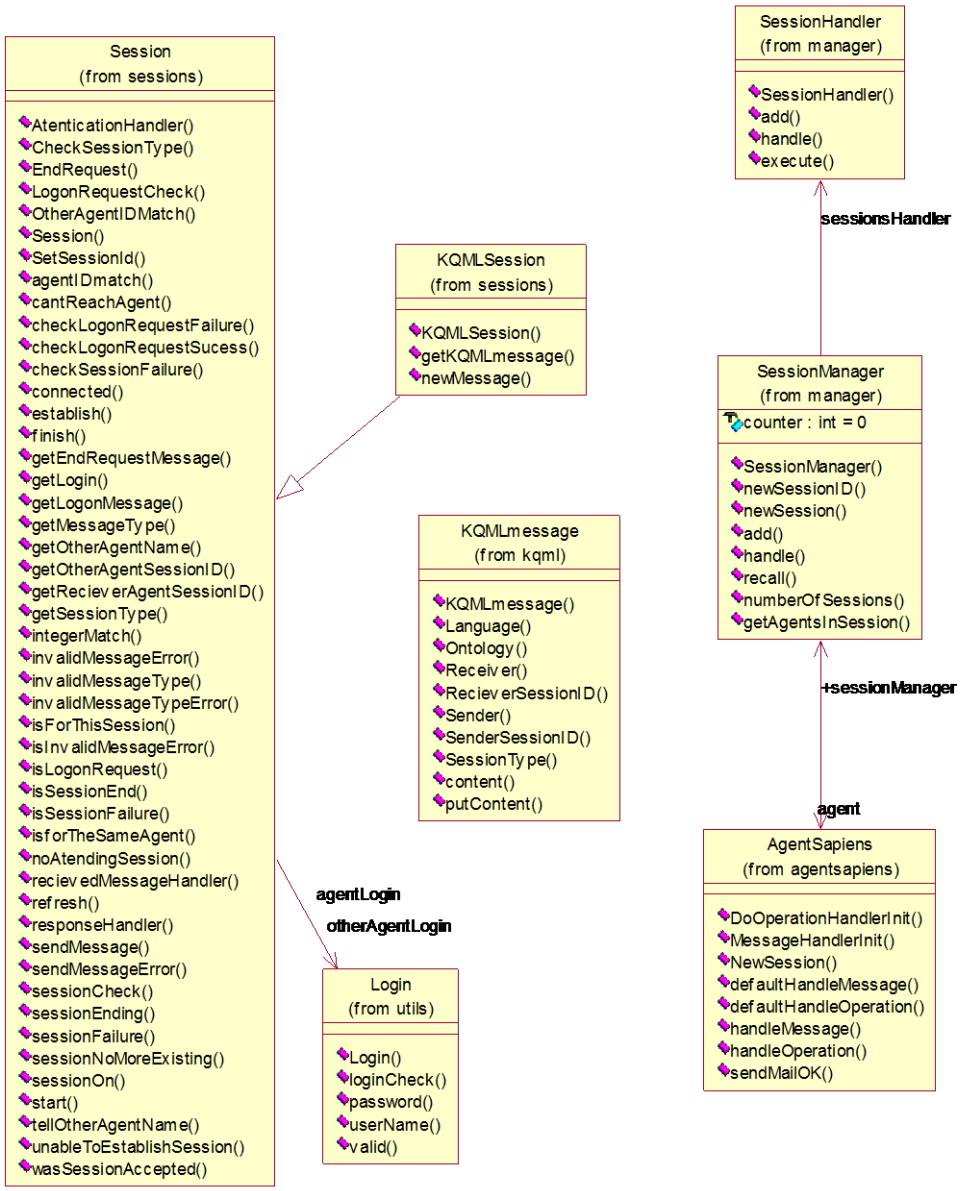

Figura 4 - Padrão Session

Exemplo de utilização:

Para demonstrar como utilização deste padrão, recorremos ás sessões implementadas para um outro padrão (Secretary/SecretaryCourier), que explicaremos mais á frente. As sessões correspondem respectivamente á sessão que um Agente do tipo Secretary utiliza para registar um agente do tipo SecretaryCourier e á sessão neste agente que recebe as mensagens dessa sessão e que permitem ao agente negar ou aceitar registar-se.

Passo 1 - Exemplo de como o agente do tipo Secretary inicia a sessão:

```
sessionManager.add( new SRegistSession(this,agentID,getCurrentContext()));
```

Passo 2 - Exemplo de como o agente SecretaryCourier aceita a sessão:

```
public Object SCourierRegistSession(Object msg){

try{
     if (SCourierRegistSession.CheckSessionType(msg))
     return new SCourierRegistSession(this,getCurrentContext());
    }
catch (Exception e) {  }
   return null;
 }
```

Este método é chamado pelo objecto SessionManager quando recebe uma mensagem para iniciar um sessão, um vez que na criação de um agente deste tipo e executado o seguinte código:

```
public void onCreationTime(Object init){

 sessionManager.recall("SCourierRegistSession");
 sessionManager.recall("jobSession");
 secretaryCourierCreation( init)  ;
 jobsInit();

}
```

De um maneira resumida, o que acontece no passo 1, é o seguinte:

- O objecto sessão é criado e adicionado ao SessionManager
- O SessionManager gera o Id para a sessão, atribui-lhe essa sessão, e inicia o processo chamando o método *establish* da sessão que se destina a enviar a mensagem que é usada para iniciar a sessão.

No passo 2, o SessionManager do outro agente recebe essa mensagem e vai chamando todos os métodos que se lembra (*recall*) que possam devolver objecto sessão. Eventualmente chega a vez que chama o método ScourierRegistSession. Neste método verifica-se a mensagem é para uma sessão deste tipo e em caso afirmativo é devolvido o objecto sessão respectivo. Uma vez devolvido o objecto sessão, o SessionManager encaminha a mensagem para o processo de autenticação.

Explicaremos agora de seguida o código das sessões:

```
public class SRegistSession extends KQMLSession implements Serializable{

Secretary agent=null;


  public SRegistSession(Secretary _agent,String OtherAgent,ContextView CV) {
  super(_agent.getId().toString(),OtherAgent,CV);

  agent=_agent;

  }

  public String getSessionType(){return "SCourierRegistSession";}

  public static boolean CheckSessionType(Object msg){
  return  CheckSessionType(msg,"SCourierRegistSession");
  }

  public void responseHandler(Object message){


    KQMLmessage msg= getKQMLmessage(message);

     if (msg.Ontology().equalsIgnoreCase("Accept")){

         agent.registAgent((RegistedAgentInfo) msg.content());
         sendMessage(getEndRequestMessage());
       }
     if (msg.Ontology().equalsIgnoreCase("Dont accept")){

         agent.declineRegist(tellOtherAgentName());
         sendMessage(getEndRequestMessage());
       }
   }

  public void start(){
  sendMessage(newMessage("Regist",agent.getSecretaryInfo()));
  }

}
```

O que acontece é o seguinte:

- Se No passo 2 a sessão é aceite será devolvida a mensagem respectiva e o método *start* desta sessão será então chamado.
- A troca de mensagens é então iniciada no método *start*, com a mensagem que manda a informação necessária para o pedido de registo.
- A resposta a mensagens enviadas serão então recebidas no método responseHandler.
- A resposta é tratada e um mensagem para terminar a sessão é mandada.

De um maneira indêntica, o código da sessão que recebe será o seguinte:

```
public class SCourierRegistSession extends KQMLSession{

  SecretaryCourier agent=null;
  boolean accept=false;

  public  SCourierRegistSession(SecretaryCourier _agent,ContextView cv) {
  super(_agent.getId().toString(),cv);
  agent=_agent;
  }

  public String getSessionType(){return "SCourierRegistSession";}

  public static boolean CheckSessionType(Object msg){
  return  CheckSessionType(msg,"SCourierRegistSession");
  }

  public Object recievedMessageHandler(Object message){

   KQMLmessage msg=getKQMLmessage(message);
    if (msg.Ontology().equalsIgnoreCase("Regist"))
     if (agent.acceptToRegister(tellOtherAgentName())){
         agent.newSecretary((Bag)msg.content());
         accept=true;
        return newMessage("Accept",agent.getRegistInfo());
        }
      else
        return newMessage("Dont accept",msg);

   return newMessage("unknownMessage",msg);
   }

   public void sessionEnding(){

   if (accept)
     agent.goToHomeDestinationPlace();
   }
}
```

O que acontece é o seguinte :

- As mensagens que foram enviadas são recebidas no método *RecievedMessageHandler* e portanto tratadas ai.

- Quando é recebida pela Sessão mensagem para terminar a sessão, o método *sessionEnding* é chamado.

Pensamos que este exemplo, é um exemplo simples de como utilizar as sessões, mas se desejar ver um exemplo mais complexo, poderá recorrer ao exemplo do myGlobalNews, onde se estabelecem sessões entre um servlet e agentes. Neste poderá ser mais claro, nas vantagens que as Sessões trazem ao proporcionar um contexto de execução e troca de mensagens. No entanto, neste exemplo já podemos ver claramente como um objecto Sessão separa o tratamento das mensagens do código do agente e as agrupou num determinado contexto. Num exemplo mais complexo em pode haver muitos mensagens (de diferentes tipos, para vários tipos de interacção , vindas de simultaneamente de diferentes entidades, etc.), o código do agente iria se deteriorar. Com este padrão isso já não acontece pois ele oferece de uma maneira extremamente simples uma maneira estruturada de lidar com essas situações.

4.4 Padrão Secretary/Secretary Courier agent

- **Introdução:**

  Em diferentes tipos de locais de trabalho, existem secretárias que ajudam o seu patrão a realizar tarefas como: lembrá-lo da sua agenda, monitorizar os trabalhos que estão a ser realizados para ele, apresentar-lhe resultados intermédios quando requisitados, controlar e contactar as pessoas que estão a trabalhar para ele, etc.

- **Classificação**

  Navegação, Interacção, Tarefa, Agente.

- **Nome:**

  Secretary/SecretaryCourier

- **Problema:**

  Como contactar ou manter o contacto com agentes que mudam de localização e como tirar facilmente partido das funcionalidades destes.

- **Contexto:**

  Está-se a desenvolver uma aplicação que incorpora agentes móveis.

- **Forças:**

  - agentes móveis mudam de localização com frequência

  - um agente precisa de contactar outro agente mas não sabe a sua localização

  - o aumento do numero de agentes que executam as tarefas, não deve aumentar a complexidade

  - o tipo de tarefas que um determinado agente executa, pode afectar desempenho conforme o tipo de comportamento do agente ao enviar resultados.

  - o dono de um determinado agente quer contactar o agente para terminar a sua tarefa, ou para mandar um resultado intermédio.

- **Solução:**

  A utilização do padrão “Secretary/SecretaryCourier” como agentes, onde:

  - o agente do tipo Secretary, é o agente responsável pelos agentes encarregados de executar tarefas. Inclui a capacidade de poder “contratar” (registar) mais agentes, encomendar a realização de tarefas. É também o agente que recebe os resultados dessas tarefas.

- Os agentes do tipo SecretaryCourier são os agentes que realizam essas tarefas. Estes agentes incluem também comportamentos genéricos pré-definidos e seleccionáveis como:

    - envio de resultados sempre que apareça um
    - envio de resultados apenas no fim
    - envio de resultados sempre antes de se mover para outro sitio
    - envio de resultados de tempos a tempos
    - etc.

  Estes tipos de comportamentos podem assim ser escolhido tendo em conta factores como: o tamanho e a quantidade dos resultados, a necessidade temporal de se querer ter acesso a esses resultados… Assim pode-se influenciar o seu desempenho.

  Os agentes do tipo SecretaryCourier quando registados pode passar toda a informação necessária para que um determinado trabalho possa ser encomenda tendo em conta assim a escalabilidade da Secretária.

- **Referências:**

  Secretary Pattern [27]

- **Implementação:**

  Para implementar este padrão, foram então implementadas as classes Secretary, SecretaryCourier , Job , JobManager e JobInfo.

  As classes Secretary e SecretaryCourier correspondem ao agentes descritos na secção anterior. O agente SecretaryCourier é um agente do tipo Courier que executa um tipo particular de tarefas (Jobs).

  A classe Job corresponde ao tipo de tarefas que o agente SecretaryCourier executa e que permitem usufruir dos comportamentos descritos anteriormente.

  A classe JobManager deriva da classe TaskManager e destina-se a poder tratar e tirar partido dos Jobs de uma maneira simples. Este objecto substitui assim o TaskManger utilizado por defeito no objecto Itinerary.

  A classe JobInfo e que contem a informação necessária para o agente Secretary poder mandar o agente SecretaryCourier executar um determinado trabalho.

  Este padrão implicou também a implementação de outros objectos como; RegistedAgentInfo (que contem a informação de um agente SecretaryCourier e é passado ao agente Secretary na altura do registo), JobConfigurationPanel (que pode ser passado nos objectos JobInfo para o agente Secretary e permite a configuração dos parâmetros de um trabalho), e de alguns objectos Session que encapsulam a troca de mensagens entre os agentes (ver diagrama da página 23).

Exemplo de utilização:

Para demonstrar como utilização deste padrão, recorremos ao exemplo da aplicação AgentSpace PhotoPaint, mais concretamente ao código do agente ImageAplication e a um agente de procura de imagens.

O agente ImageAplication é um agente do tipo Secretary. O exemplo de utilização do padrão Sessão corresponde ao encapsulamento do processo de registro de um agente do tipo SercretaryCourier.

Exemplo de utilização do agente Secretary:

1. Registar um novo agente:

Para registar um novo agente basta evocar o seguinte método.

```
registNewAgent(String agentID);
```

(no exemplo o método é evocado na sessão “BuyNewAgentSession”, assim que se obtém o Id do agente que se adquiriu).

Como no final do registo o método “newRegistedAgent(RegistedAgentInfo registedAgentInfo)”, bastou redefinir este método no agente ImageAplication da seguinte maneira:

```
public void newAvailableAgents(Bag newAvailableAgents){

sendSapiensEvent("newAvailableAgents",newAvailableAgents);
}
```

o que fará executar o método que actualizará a interface respectiva. Após um agente se ter registado, o agente Secretary recebe um objecto do tipo RegistedAgentInfo.

2. Ordenar a um agente registado para executar um determinado trabalho:

Para ordenar a um agente de procura (SecretaryCourier) registado para executar o trabalho que irá retornar os previews das imagens basta chamar o seguintes métodos:

```
RegistedAgentInfo info= (RegistedAgentInfo) searchAgents.get(agentShortDescriptionChoice.getSelectedIndex() );
```

que obterá a informação do agente seleccionado. De seguida,

```
JobInfo jobInfo=info.getJobInfo("getImagePreviewsJob");
```

que obterá o objecto jobInfo respectivo ao trabalho (Job) , e que contem a informação necessária para se poder mandar o agente de procura executar esse trabalho ("getImagePreviewsJob").

Depois configura-se o tipo de comportamento que queremos que o agente tenha (no caso dos agentes de procura queremos que estes sempre que obtenham um resultado, ou seja, um preview de uma imagem, este seja entregue) :

```
jobInfo.setWhenGetAResultParameter();
```

De seguida é necessário actualizar os parâmetros do trabalho. Para tal, basta executar o método:

```
jobInfo.setParameters(this);
```

Se o agente de procura não necessita de parâmetros, nada acontecerá. Caso contrário (caso dos agentes de procura por palavras-chave ou por preço), a respectiva interface aparecerá.

De seguida bastaria executar o método do agente ImageAplication (herdado do agente Secretary):

```
public void doJob(String agentID,JobInfo jobInfo)
```

O respectivo agente de procura iniciaria o respectivo trabalho e o agente ImageAplication começaria então a receber as imagens

O trabalho é iniciado através da sessão “DoJobSession” encapsulada no agente Secretary. Sempre que chega um novo resultado, é evocado o método:

```
Method f = agent.getClass().getMethod(jobInfo.getJobName()+
                                      "Result",new Class[] {result.getClass()});
```

Isto significa que no caso trabalho "getImagePreviewsJob" é evocado o método:

```
public void getImagePreviewsJobResult(MetaImage img){

sendSapiensEvent("getImagePreviewsJobResult",img);
}
```

que foi implementado obviamente de maneira a enviar imagem recebida para a respectiva interface.

Por sua vez, quando o agente de procura termina o trabalho é evocado o método:

```
public void jobFinished(String agentID,JobInfo jobInfo){

sendSapiensEvent(jobInfo.getJobName()+"Finished",agentID);
}
```

Exemplo de utilização dos agentes do tipo SecretaryCourier

Os agentes de procura no exemplo do AgentSpace PhotoPaint são agentes deste tipo.

Inicialização dos trabalhos que o agente sabe executar:

No caso de um trabalho sem parâmetros:

```
public void jobsInit(){

 jobManager.addJob(new CallbackJob("getImagePreviewsJob",this));
 }
```

No caso de um trabalho com parâmetros:

```
public void jobsInit(){
 jobManager.addJob(new CallbackJob("getImagePreviewsJob",new KeywordPanel(),this));
 }
```

Execução do trabalho:

Se o agente receber a ordem para executar o trabalho acima descrito, é evocado o seguinte método:

```
public void getImagePreviewsJob(){

    newItinerary();

    addTasksList(new CallbackAgentTask(photoPaintReceptionistPlace,this,"askForImageStores"));
    addTasksList(new CallbackAgentTask(photoPaintReceptionistPlace,this,"goToAllImageStores"));

    addFinishTravelList(new CallbackAgentTask(this,"endJob"));

    GoTravel();
}
```

Isto significa que o agente constrói um itinerário (outro padrão de software), com as tarefas a realizar e começa a percorrer esse itinerário deslocando-se inicialmente para o local photoPaintReceptionistPlace onde realizará a tarefa e obterá os Ids e localizações de todas as lojas existentes. De seguida, na realização da segunda tarefa acrescentará ao itinerário para todas as lojas as seguintes tarefas:

```
public void goToAllImageStores(){


 for(int i=0; i < (imagesStoresIds.numberOfObjects()); i++){

  PlaceId place = new PlaceId((String) imagesStoresPlaceIds.get(i));

  addTasksList(new CallbackAgentTask(getDestinationPlace(place),
                          this,"askForImagePreviews",(String) imagesStoresIds.get(i)));
}
```

O que fará com que o agente de seguida se desloque para cada um dos lugares das lojas para realizar a respectiva tarefas (onde se iniciarão sessões com as respectivas lojas).

Entrega de um resultado:

Durante as sessões com as lojas sempre que o agente recebe um preview de uma imagem é evocado o seguinte método:

```
public void recieveNewImagePreview(MetaImage img){

 jobManager.addToCurrentJob(img);
 }
```

No caso de um agente de procura por critério:

```
public void recieveNewImagePreview(MetaImage img){

 if (satisfiesCriteria(img))
   jobManager.addToCurrentJob(img);
 }
```

4.5 Padrão Receptionist

- **Introdução:**

  Normalmente, uma pessoa sabe a partida o que se faz num determinado lugar, no entanto, quando lá chega, tem de obter informações para saber aonde ou a quem se dirigir para obter certo tipo de informação ou serviços lá existentes (ás vezes até nem sabe o que lá se faz e tem que perguntar).

  Este tipo de comportamento, pode ser visto como uma situação recorrente no nosso dia a dia. Pode e costuma ter uma solução simples, que é a existência de uma recepcionista que providencia este tipo de informação. Pode servir também para uma pessoa se registar á entrada, para facilitar certos tipos de interacção como: avisar á pessoa que está na hora de se ir embora, que existe alguém nesse local que quer falar com ela…

  Esta solução pode ser facilmente transposta e adaptada ao paradigma dos agentes de software.

- **Classificação**

  Navegação, Tarefa, Agente.

- **Nome:**

  Recepcionista

- **Problema:**

  Como é que agentes comunicam com outros agentes, localizados num determinado lugar, sem saber que agentes lá existem e como chegar até eles.

- **Contexto:**

  Está se a desenvolver um sistema multi-agente, e colaboração entre agentes é necessária. Para realizar tarefas, agentes tem que comunicar com outros agentes fisicamente localizados nessa máquina.

- **Forças:**

  - um agente quando chega a um lugar, pode não saber que tipos de serviços lá se disponibilizam.

  - um agente sabe que serviços num lugar se disponibilizam, mas não sabe como contactar os agentes que fornecem esses serviços.

- **Solução:**

  A utilização de um agente "recepcionista", que contem: uma descrição geral com o que existe ou se faz num dado lugar, o contacto de todos os agentes que prestam serviços nesse lugar, bem como a descrição do que cada agente faz e uma lista com todos os outros agentes e seus contactos.

  Um agente, quando chega a um determinado lugar, regista-se com a recepcionista por forma a se tornar publicamente visível a outros agentes ou pergunta, se existe algum agente que lhe faça um determinado tipo de serviço ou qual o contacto do agente que faz este tipo de serviço ou ainda que agentes existem num dado lugar.

  Para este tipo de interacção, deve ser utilizado o KQML uma vez que é um protocolo de troca de informação que se adequa a este tipo de situação.

- **Referencias:**

  É inspirado no padrão facilitator definido em [2]. Receptionist Pattern [27]

- **Implementação:**

As principais classes desenvolvidas para implementar o padrão Receptionist são as apresentadas no diagrama de classes a seguir apresentado. O padrão Receptionist foi assim implementado como um agente que oferece á partida um conjunto de funcionalidades acessíveis por outros agentes através de uma sessão de comunicação. As funcionalidades base oferecidas por este agente são:

- **Registo de um agente** – Os agentes enviam uma mensagem com um objecto do tipo *AgentInformation* (ver package inesc.as.devapi.objects) que contém a informação sobre si, seu nome, lugar onde fornecem o serviço, um texto descritivo e ainda a ontologia suportada. A ontologia especificada é importante, já que permite que outros agentes identifiquem desta forma se suportam uma sessão que seja capaz de comunicar com o agente registado (ver padrão Session). Note-se ainda que o lugar fornecido pelo agente não necessita de ser o local onde este se encontra no momento do registo.

  Eis um exemplo extraído da aplicação PhotoPaint (ver capítulo 7) que ilustra um registo de um agente:

```
AgentView av = this.getCurrentContext().getAgentOf(this.getId().toString());
String place = this.getCurrentPlaceId().toString();
AgentInformation ainfo = new AgentInformation(av,"ImageShop",this.storeDescription,
place);
this.registerAtReceptionist(ainfo,recId);
```

- **Inquérito sobre agentes registados** – Para questionar o agente recepcionista sobre os agentes registados, um agente pode construir uma pergunta definido o critério que pretende. Isto é possível através das classes AgentQuery e QueryCriteria (ver package inesc.as.devapi.objects).

  No exemplo que se segue, extraído da aplicação PhotoPaint, o agente de procura na recepcionista os agentes "loja" que se encontram registados:

```
public void askForImageStores(){


     AgentQuery aq = new AgentQuery();
     aq.addCriteria(new QueryCriteria(AgentQuery.ONTOLOGY,"ImageShop"));
     this.getAgentsFromReceptionist(aq,photoPaintReceptionist);
   }
```

  Para receber a resposta há que implementar o método de callback `handleGetAgentsFromReceptionistOK`. Este método encontra-se na classe AgentSapiens (que a classe do agente de procura implementado deriva) e é automaticamente invocado no caso de uma resposta com sucesso da recepcionista:

```
public void handleGetAgentsFromReceptionistOK(AgentDirectory agdir){
     Bag agents = agdir.getAll();

     if (agents!=null){
       AgentInformation ainfo = null;
       for (int  i=0; i < agents.numberOfObjects(); i++){
         ainfo = (AgentInformation) agents.get(i);
                 try {

this.imagesStoresIds.add(ainfo.getAgentView().getId().toString());
                   this.imagesStoresPlaceIds.add(ainfo.getWorkingPlace());
                 } catch (Exception e){e.printStackTrace();}
     }
  }
```

- **Anular o registo na Recepcionista** – o agente pode querer deixar de estar "publicamente" visível a outros agentes. Para efectuar a anulação do registo basta usar o método *unRegisterAtReceptionist()*

Diagrama de classes da recepcionista

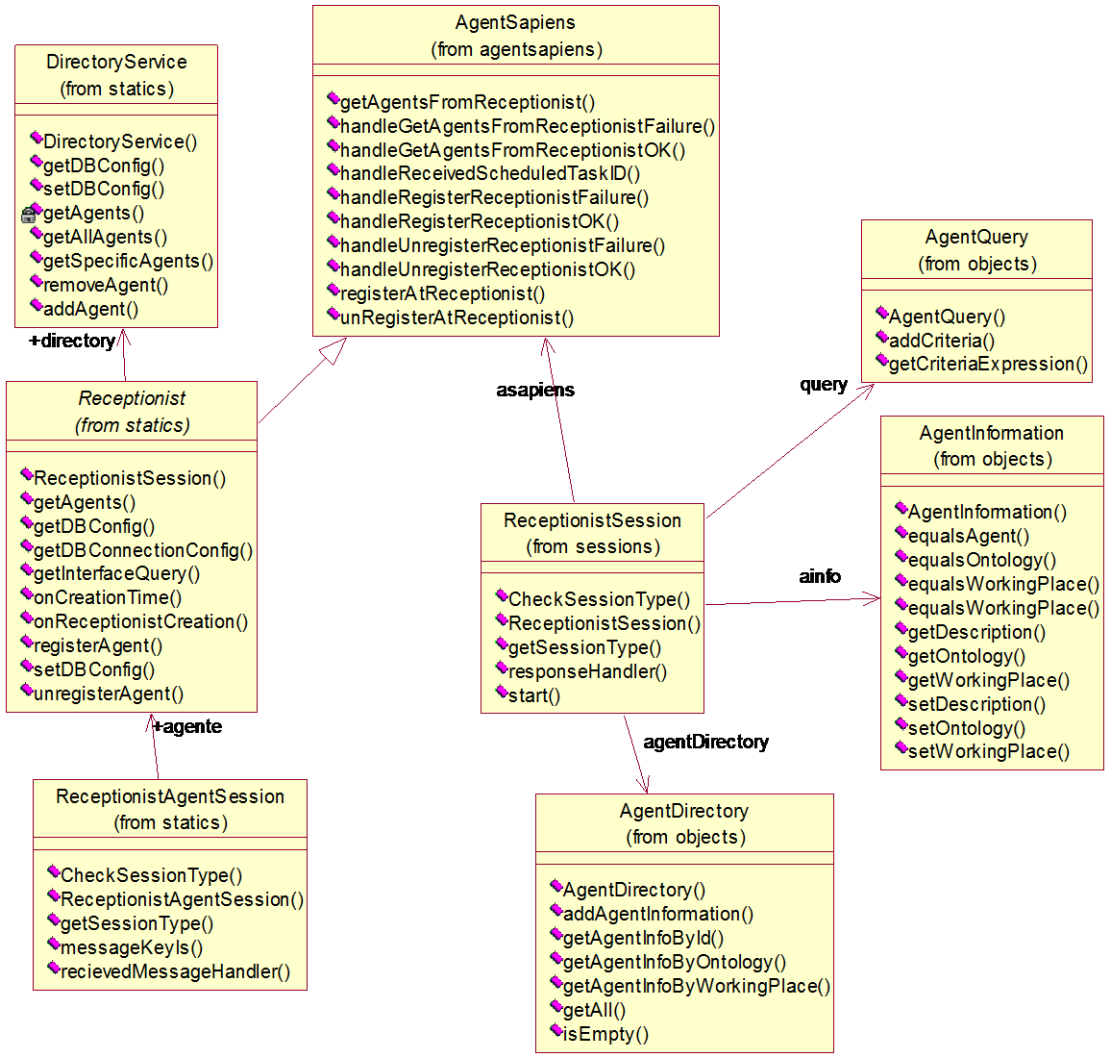

Figura 5 - Padrão Receptionist (Diagrama de classes)

Diagrama de sequência das sessões entre agentes e recepcionista

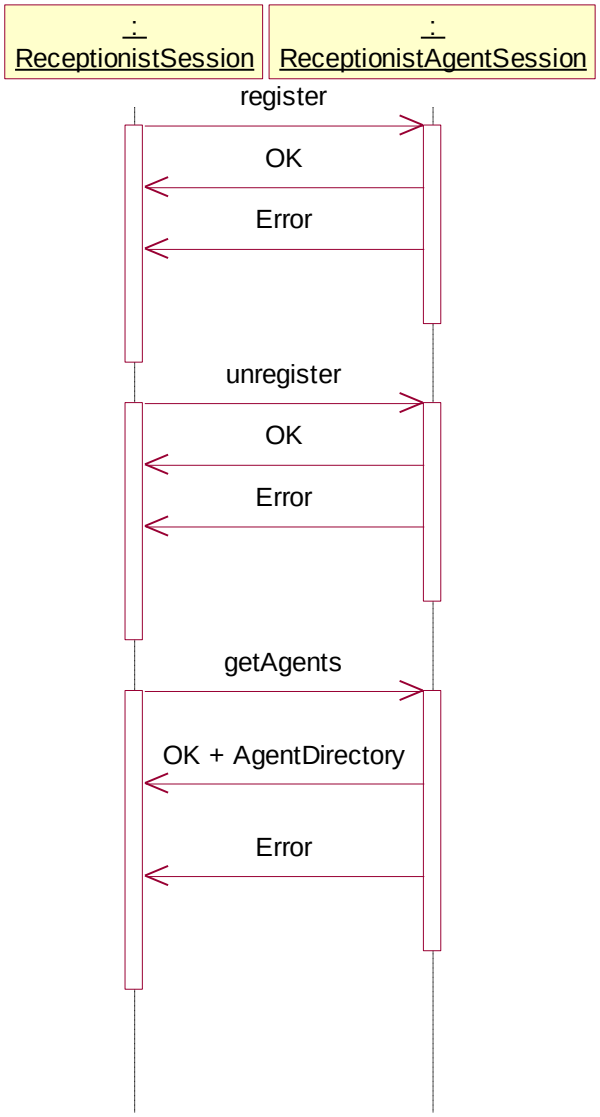

Figura 6 - Padrão Receptionist (Diagrama de Sequência)

O diagrama de sequência acima apresentado ilustra a comunicação que é realizada entre um agente do tipo AgentSapiens e um agente Receptionist.

O agente AgentSapiens utiliza um objecto do tipo ReceptionistSession , que encapsula a comunicação feita com o objecto do tipo ReceptionistAgentSession (controlado pelo agente ReceptionistAgent) aqui apresentada. Como é ilustrado na figura , existem 3 trocas de mensagens alternativas e que são realizadas conforme os métodos da classe AgentSapiens que são invocados, respectivamente registerAtReceptionist(), unRegisterAtReceptionist() ou getAgentsFromReceptionist().

5 DevAPI: Agentes

5.1 Classe de agentes AgentSapiens

A classe AgentSapiens resulta da derivação da classe Agent já existente no AgentSpace e na inclusão de diversos métodos que visam facilitar o uso dos mecanismos desenvolvidos neste TFC. Esta classe representa um papel fundamental na DevAPI já que define a classe de agentes que os programadores deverão usar como base para os seus agentes.

Destacamos aqui as principais funcionalidades oferecidas por este agente que congregam os mecanismos desenvolvidos e que tornam assim este agente num objecto mais rico que a anterior classe de agentes existentes no AgentSpace. Assim, é possível a partir da classe AgentSapiens:

- **Definir as sessões suportadas pelo agente** através do método *newSession* (ver capítulo 4, secção 3).

- **Comunicar com o agente de sistema MailAgent**, tornando o envio de emails transparente (ver secção 2 deste capítulo).

- **Comunicar com o agente de sistema ReminderAgent** (ver secção 3 deste capítulo), tornando a calendarização de mensagens transparente.

- **Comunicar com o agente de sistema ReceptionistAgent**, ou com outro agente que derive do agente genérico Receptionist, obtendo desta forma as funcionalidades básicas da recepcionista de forma transparente (ver secção 4 do capítulo 4 e secção 4 do presente capítulo).

- **Adicionar de uma forma simples tratamento de mensagens síncronas e assíncronas** para comunicação com as interfaces dos agentes com os métodos *DoOperationHandlerInit(), MessageHandlerInit(), handleOperation() e handleMessage()* (ver apêndice com a Documentação Java em anexo a este relatório para obter mais detalhes sobre estes métodos).

- **Simplificação do envio de eventos a captar pelas interfaces** de configuração e de agente através do método *sendSapiensEvent()* (ver com a Documentação Java em anexo a este relatório para obter mais detalhes sobre estes métodos).

Como os agentes desenvolvidos para implementar os padrões derivam deste agente, é possível assim combinar funcionalidades. Por exemplo, o agente Receptionista poderia enviar emails de notificação de novos agentes registados. Na pagina seguinte encontra-se um diagrama de classes que ilustra os métodos da classe AgentSapiens, bem como a relação dos vários agentes desenvolvidos com este.

Diagrama de classes AgentSapiens

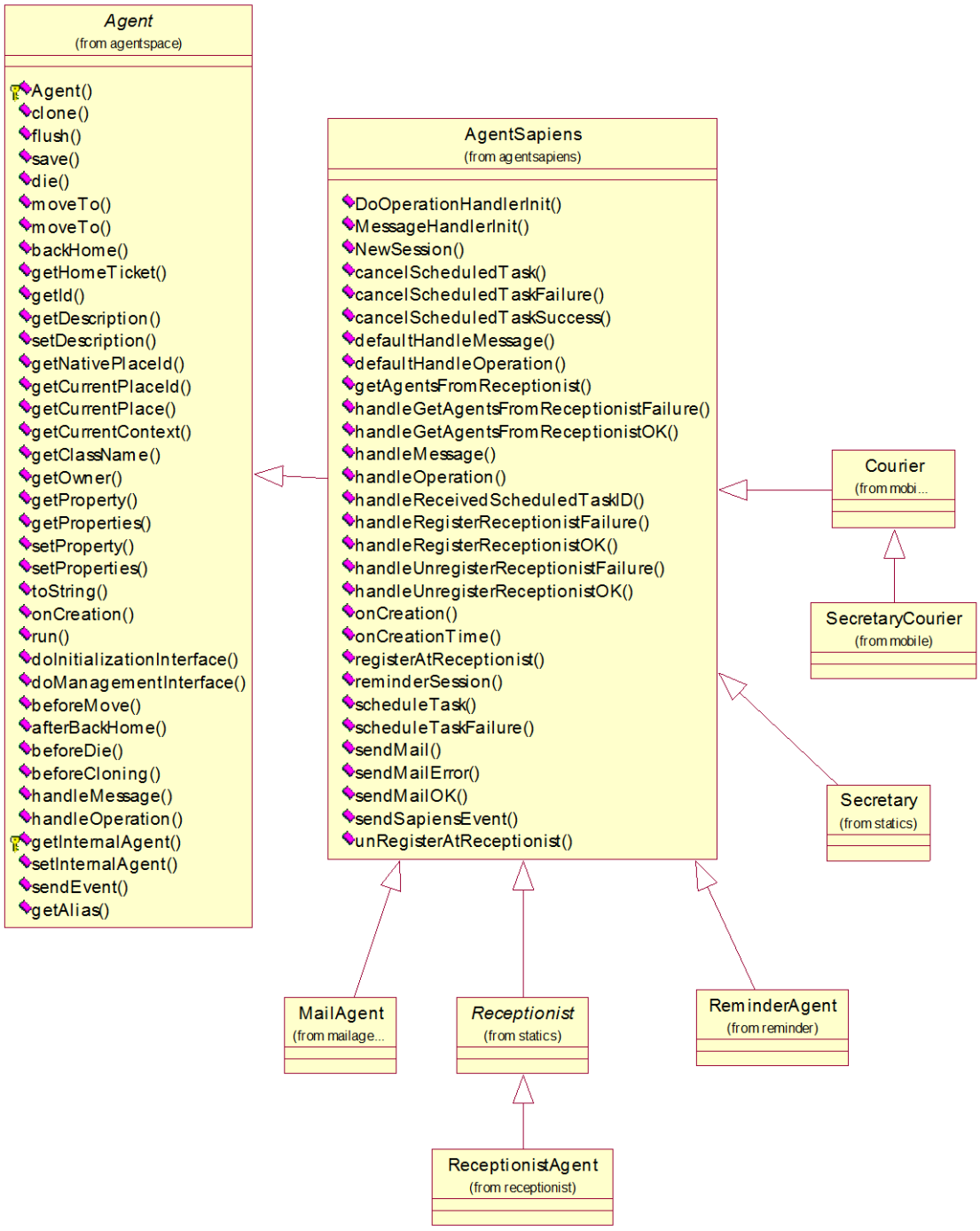

Figura 7 - AgentSapiens e seus derivados (Diagrama de classes)

5.2 Agente de Sistema Mail Agent

5.2.1 Descrição

A função do MailAgent é enviar mensagens de email em nome de outros agentes, aliviando-os dessa forma da responsabilidade de interacção com um servidor de mail SMTP. Como convém que o agente seja conhecido por todos os agentes que queiram enviar mails, convencionou-se um identificador específico, nomeadamente PID_1|AID_3 (e.g., o terceiro agente do lugar 1, de administração, da hierarquia de espaços do AgentSpace).

Em termos de implementação , este agente é extremamente simples. O diagrama de classes que a seguir apresentamos contém as classes envolvidas na implementação deste agente e os métodos mais importantes.

Diagrama de classes do Mail Agent

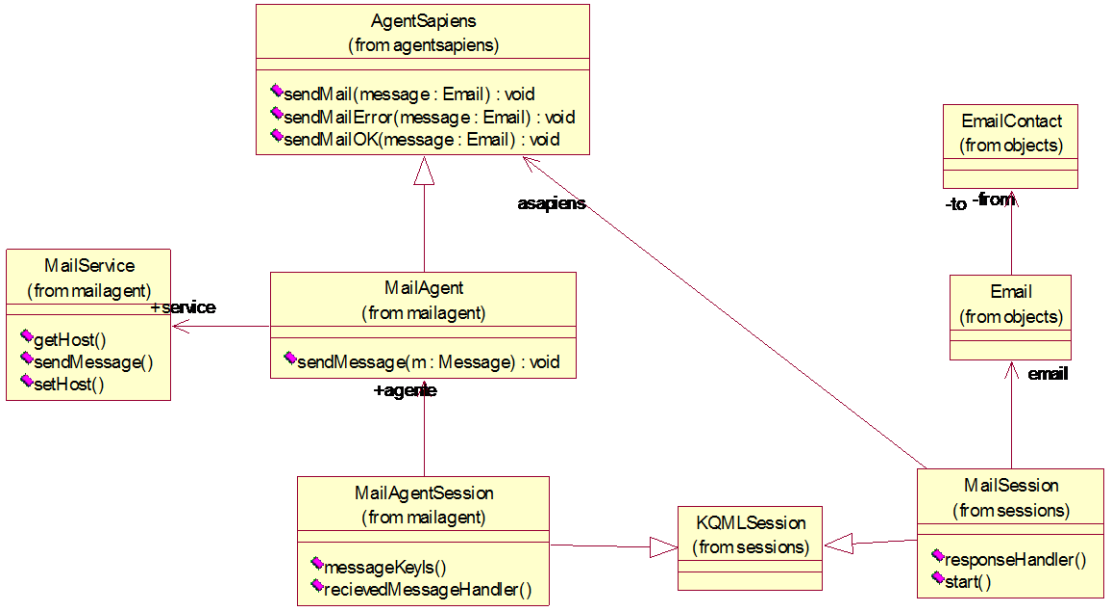

Figura 8 - Agente MailAgent (Diagrama de Classes)

MailAgent – classe que implementa o agente.

MailService – classe que comunica com o servidor SMTP

MailAgentSession – Sessão de comunicação do agente que recebe os pedidos de envio de email

MailSession – Sessão de comunicação a ser usada por outros agentes para o pedido de envio ao MailAgent.

Email – O objecto que representa a mensagem de email a ser enviada

EmailContact – representa um endereço de email.

5.2.2 Interfaces do agente

Para facilitar a vida ao administrador foram desenvolvidas duas interfaces java. A de configuração permite escolher o endereço do servidor de mail e a do agente permite testar o envio de emails.

Interface de configuração

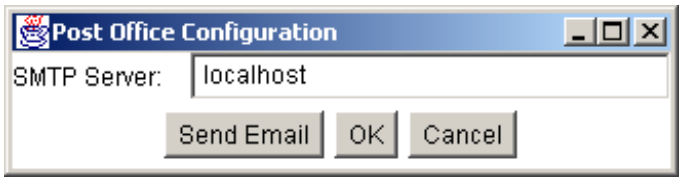

Figura 9 - Agente MailAgent (Interface de Configuração)

Interface do agente

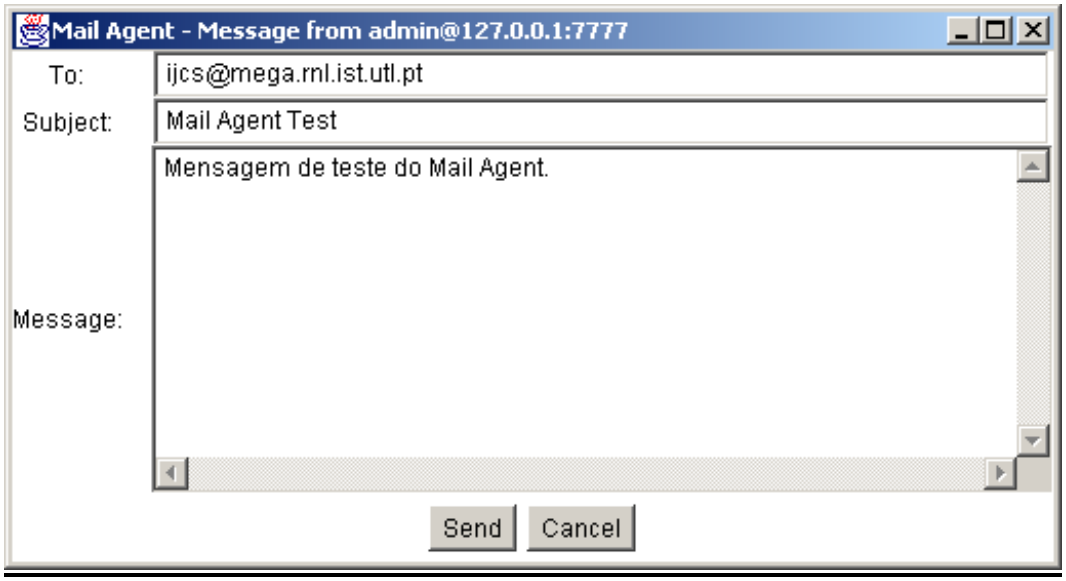

Figura 10 - Agente MailAgent (Interface do agente)

5.2.3 Comunicação

Diagrama de sequência das sessões entre agentes e MailAgent

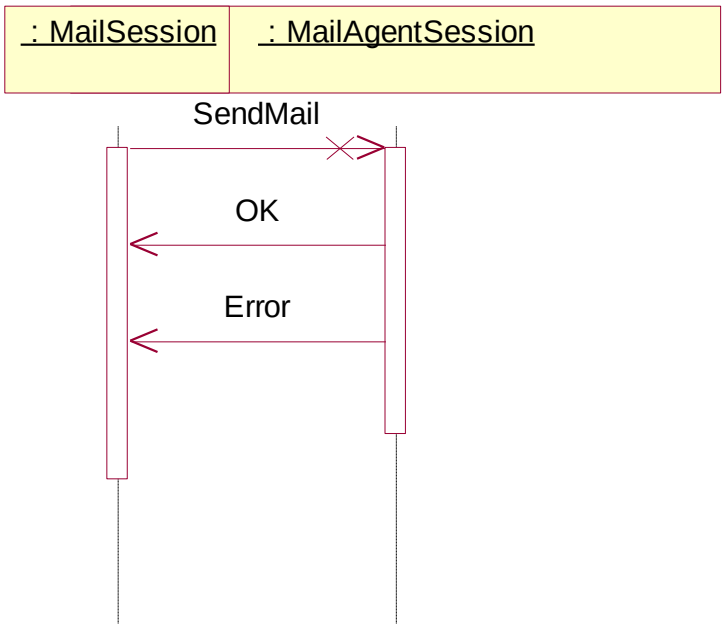

Figura 11 - Agente MailAgent (Diagrama de Sequência)

O diagrama de sequência acima apresentado ilustra a comunicação que é realizada entre um agente do tipo AgentSapiens e um agente MailAgent.

O agente AgentSapiens utiliza um objecto do tipo MailSession , que encapsula a comunicação feita com o objecto do tipo MailAgentSession (controlado pelo agente MailAgent) aqui apresentada. Esta comunicação ocorre quando é invocado o método *sendMail()* da classe AgentSapiens.

5.2.4 Para o programador

No que respeita à programação de qualquer agente que queira enviar mails, o código necessário é extremamente simples, como se depreende do exemplo que se segue tirado da aplicação myGlobalNews:

```
EmailContact from = new EmailContact(this.getAlias(),
                                getItemConfig("EMAIL_ADMIN"));

EmailContact to = new EmailContact( _dados_pessoais.getNome(),
                               _dados_pessoais.getEmail());

Email m = new Email(from, to, subject, "Exmo(a) Sr(a) "
          +_dados_pessoais.getNome()+ " :" + corpo);

this.sendMail(m);
```

Para tratar o sucesso ou insucesso do pedido de envio há que implementar as funções de callback da classe AgentSapiens *sendMailOK* e *sendMailError* respectivamente.

5.3 Agente de Sistema Reminder Agent

5.3.1 Descrição

A função do ReminderAgent é enviar mensagens calendarizadas para agentes em nome de outros agentes, aliviando-os dessa forma da responsabilidade de implementar o algoritmo de calendarização. Como convém que o agente seja conhecido por todos os agentes, convencionou-se um identificador específico, nomeadamente PID_1|AID_2 (e.g., o segundo agente do lugar 1, de administração, da hierarquia de espaços do AgentSpace).

Em termos de implementação , este agente é extremamente simples. O diagrama de classes que a seguir apresentamos contém as classes envolvidas na implementação deste agente e os métodos mais importantes.

Diagrama de classes do ReminderAgent

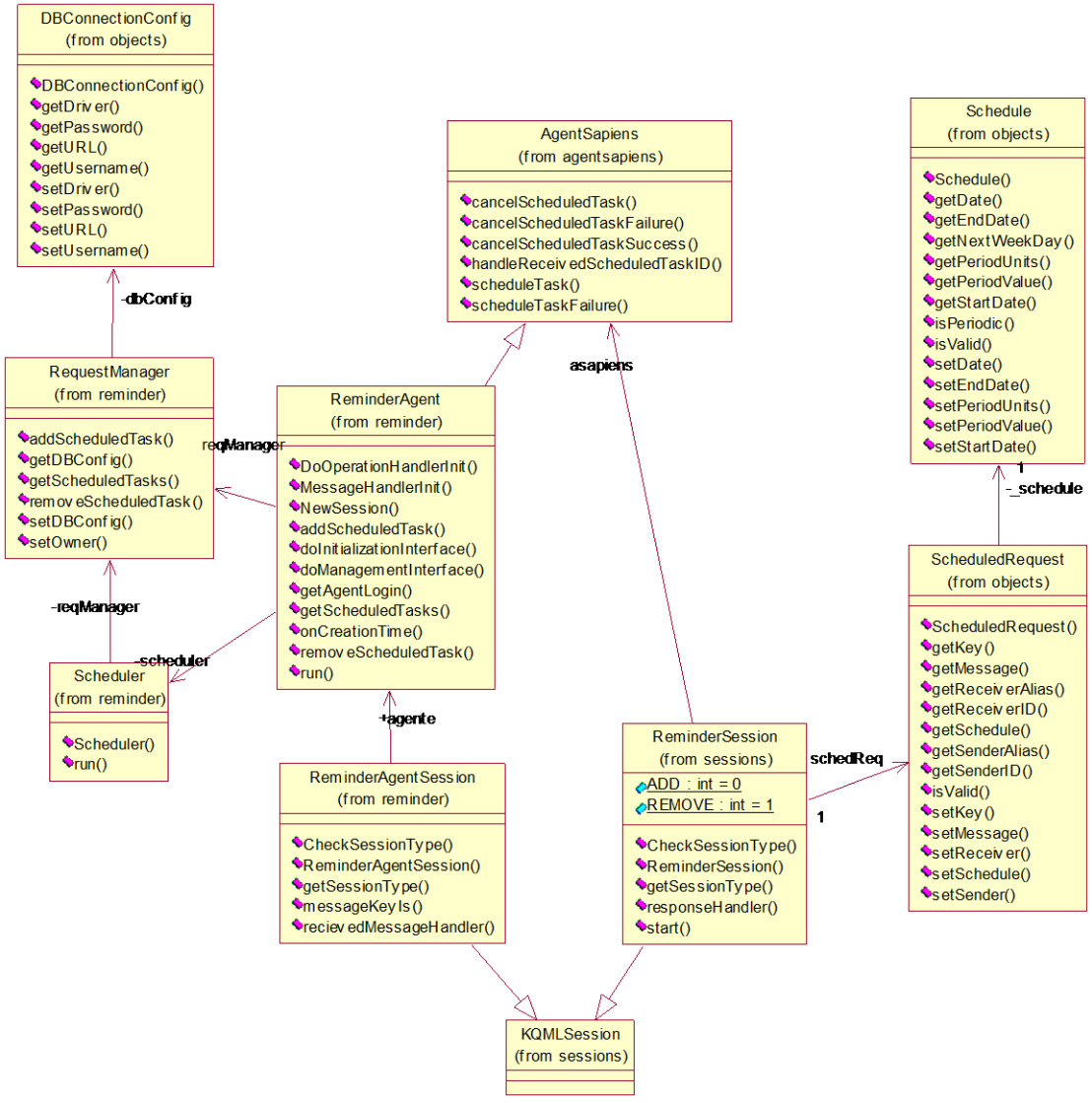

Figura 12 - Agente ReminderAgent (Diagrama de Classes)

5.3.2 Interfaces do agente

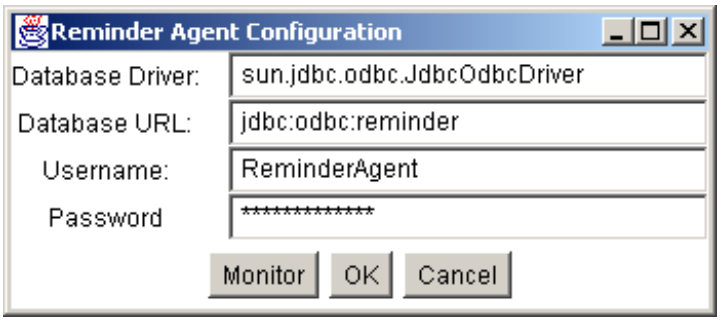

Figura 13 - Agente ReminderAgent (Interface de configuração)

A interface de configuração do agente ReminderAgent permite definir os parâmetros de acesso à base de dados onde são guardadas as mensagens a enviar aos agentes e respectivas calendarizações.

5.3.3 Comunicação

Diagrama de sequência para a sessão entre agentes e ReminderAgent

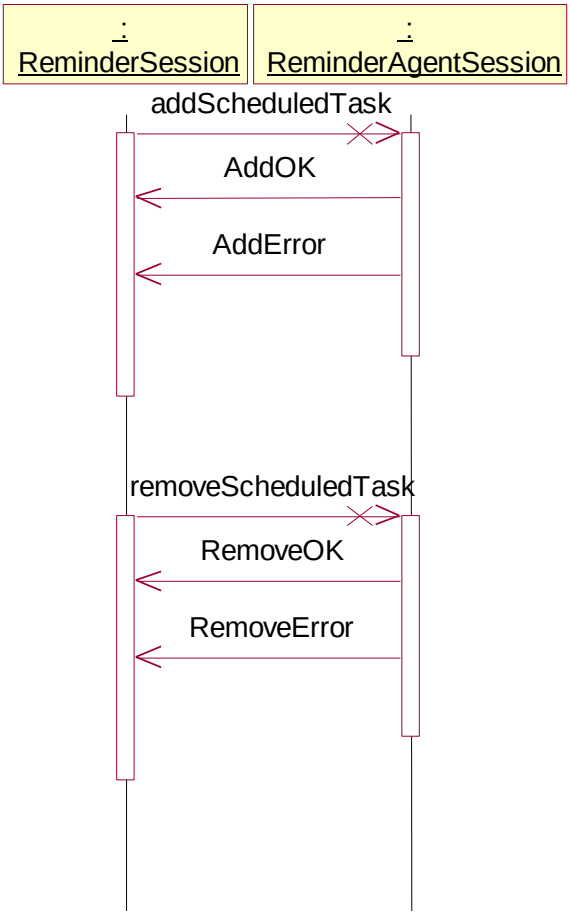

Figura 14 - Agente ReminderAgent (Diagrama de Sequência)

O diagrama de sequência acima apresentado ilustra a comunicação que é realizada entre um agente do tipo AgentSapiens e um agente ReminderAgent.

5.3.4 Para o programador

Do ponto de vista do programador, enviar uma mensagem calendarizada resume-se a três passos. No exemplo que se segue uma mensagem é calendarizada para ser enviada durante um mês de 3 em 3 dias, a iniciar em 1/3/2001 e a terminar em 30/3/2001.

1. Criar um objecto Schedule onde se define a calendarização da mensagem.

```
Schedule sched = new Schedule(new Date(2001,03,01),
                              new Date(2001,3,30),
                              3,Schedule.COUNT_IN_DAYS);
```

2. Criar um objecto ScheduleRequest que associa a mensagem à calendarização criada.

```
AgentView avROBOT = toAgent;

AgentView sist = getCurrentContext().getAgentOf(this.getId().toString());

ScheduledRequest request = new ScheduledRequest(sist, avROBOT, sched,
                                                "recolherNoticias", "none");
```

3. Fazer o pedido de calendarização

```
this.scheduleTask(request);
```

O programador deverá tratar as respostas vindas do ReminderAgent, isto é, o caso de sucesso ou insucesso do pedido. No caso de sucesso é retornado um id do pedido feito para tratamento posterior. Para realizar este tratamento há que implementar os callbacks definidos na classe de agentes AgentSapiens.

```
public void handleReceivedScheduledTaskID(String schedReqID){

     ultima_fonte_ID = schedReqID;
         //this.schedReqIDs.add(schedReqID);
  }

  public void scheduleTaskFailure(){
    System.out.println("Scheduled task add failure!");
 }
```

Para cancelar um pedido de calendarização basta fazer:

```
this.cancelScheduledTask(schedReqId);
```

Novamente, o pedido de cancelamento também poderá ser bem sucedido ou falhar, logo há que implementar os respectivos callbacks:

```
public void cancelScheduledTaskSuccess(String reqSchedID){
     System.out.println("Scheduled task removal sucess!");
}

public void cancelScheduledTaskFailure(String reqSchedID){
    System.out.println("Scheduled task removal failure!");
}
```

5.4 Agente de Sistema Receptionist Agent

5.4.1 Descrição

O agente ReceptionistAgent apresenta as funcionalidades descritas no padrão Receptionist (ver capítulo 4 secção 4). A implementação deste agente é muito simples já que este agente não é mais do que o agente genérico Receptionist com interfaces de configuração e de agente adicionadas! Sendo um agente de sistema, este agente tem á partida um id conhecido , que se convencionou ser PID_1|AID_1 (o 1º agente do 1º lugar da hierarquia de lugares do AgentSpace). Em baixo é apresentado o diagrama de classes deste agente bem como uma imagem das interfaces de configuração e do agente.

Diagrama de classes do ReceptionistAgent

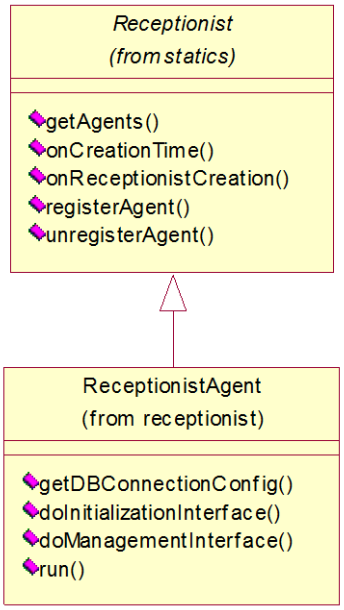

Figura 15 - Agente ReceptionistAgent (Diagrama de classes)

5.4.2 Interfaces do agente

Na interface de configuração estabelecem-se os parâmetros de acesso à base de dados onde o agente guarda as informações sobre os agentes.

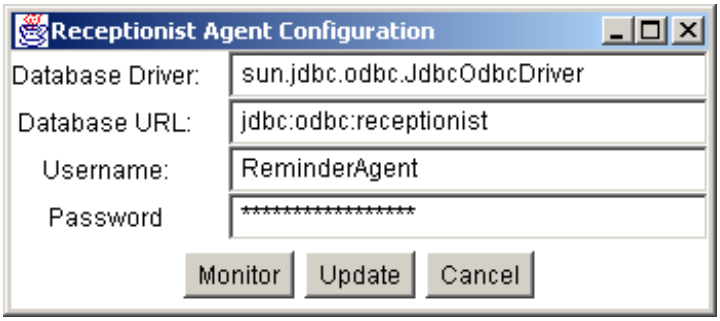

Figura 16 - Agente ReceptionistAgent (Interface de Configuração)

Na interface do agente é possível visualizar os agentes registados e ainda registar manualmente novos agentes.

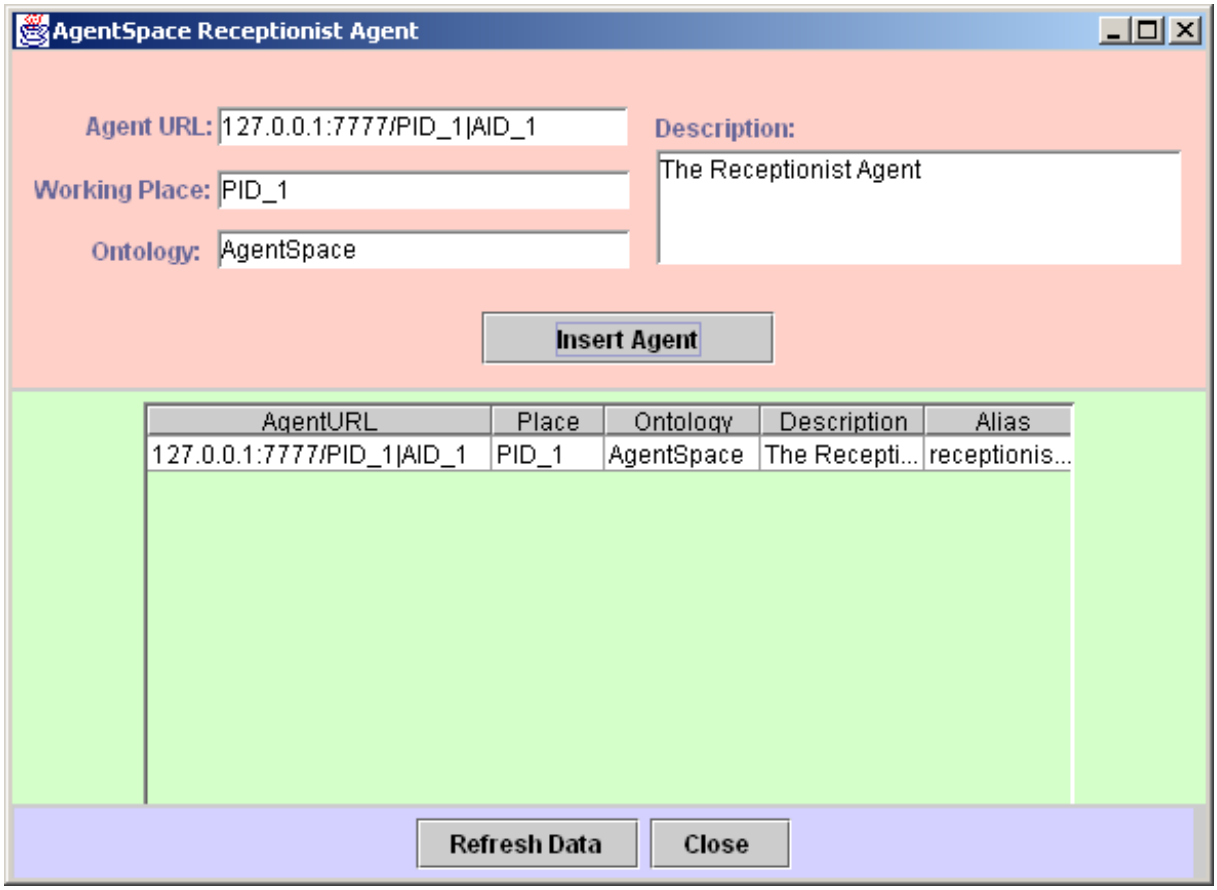

Figura 17 - Agente ReceptionistAgent (Interface do agente)

5.4.3 Comunicação

A comunicação entre um agente AgentSapiens e o ReceptionistAgent é a fornecida com a implementação do padrão (ver capítulo4 , secção 5).

5.4.4 Para o programador

Do ponto de vista de programação de um agente para tirar partido das funcionalidades oferecidas pelo ReceptionistAgent, as operações são as mesmas que estão descritas na implementação do padrão (ver capítulo4 , secção 5). Existe no entanto uma diferença na assinatura dos métodos a invocar (*registerAtReceptionist*, *unRegisterAtReceptionist* e *getAgentsFromReceptionist)*, já que não é necessário especificar o id do agente Receptionist, visto que sendo um agente de sistema, este já tem um id standard (PID_1|AID_1), conhecido pelo agente AgentSapiens e encapsulado no objecto que implementa a comunicação (ver inesc.as.devapi.sessions.ReceptionistSession na documentação da API, em anexo a este relatório).

6 DevAPI: AgentSpace Servlet e Interfaces Utilizador

6.1 Introdução

Neste capítulo é feita uma descrição dos restantes mecanismos que compõem a plataforma DevAPI: AgentSpace Servlet e Interfaces Web do Utilizador para comunicação com os agentes. A necessidade de um mecanismo que permita construir interfaces em html para as ABA surge de uma forma natural, já que muitas vezes a versatilidade das interfaces java não compensa o custo do carregamento remoto de classes "pesadas" como são usualmente aquelas que implementam as interfaces java.

6.2 AgentSpace Servlet

Arquitectura

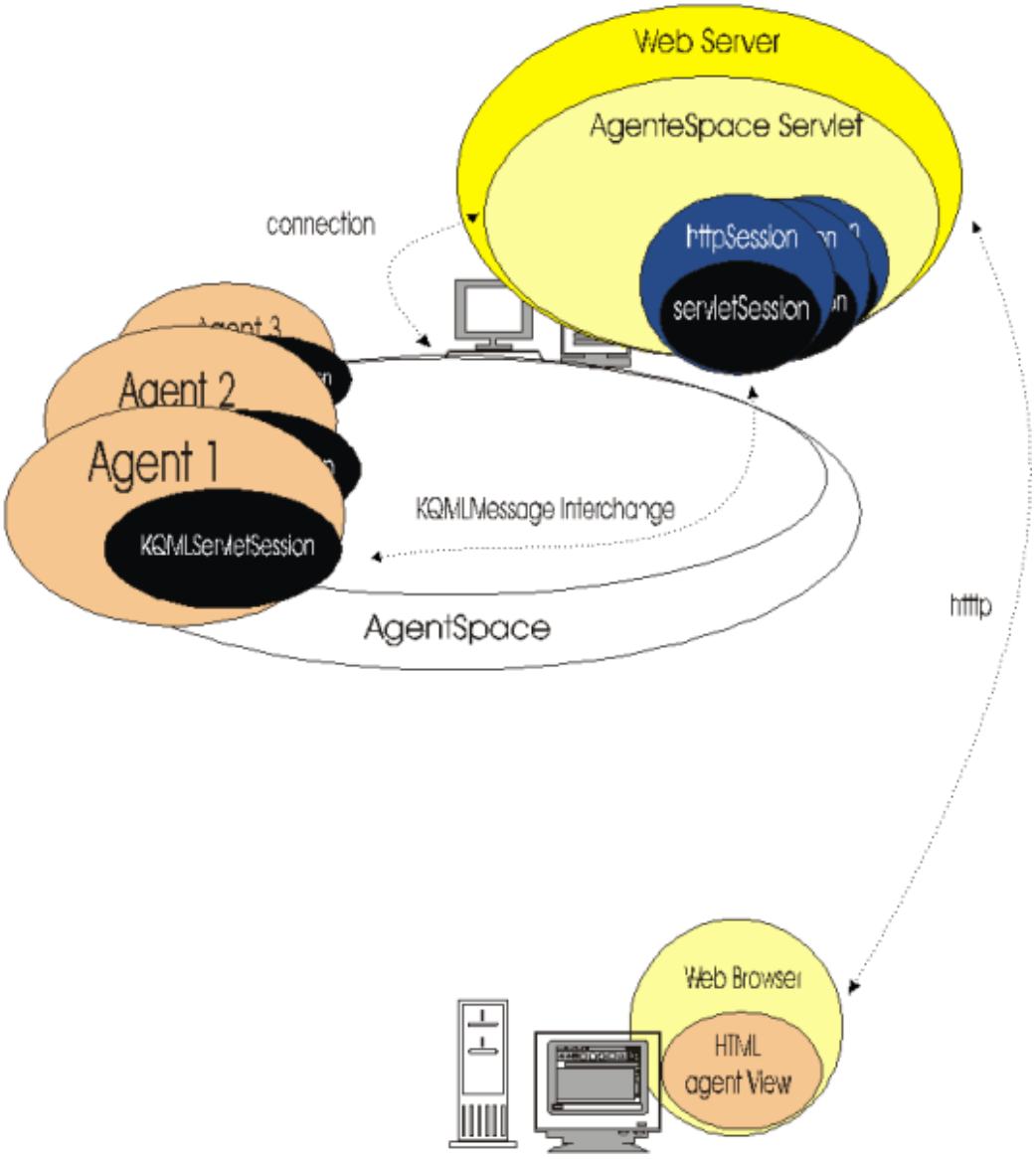

Figura 18 - Arquitectura AgentSpace Servlet

A arquitectura que envolve a AgentSpace Servlet encontra-se representada na figura da página anterior. Como se depreende da figura, a AgentSpace servlet age como que um conversor entre o protocolo HTTP e o protocolo KQML.

A principal função desta servlet java é identificar os clientes e converter os pedidos vindos do browser web em mensagens enviadas para os agentes respectivos. Os agentes por sua vez retornam html que é redireccionado para o cliente.

Em termos de implementação interna, a AgentSpace Servlet pode ser vista como um cliente , que tal como o cliente AS de gestão remota, utiliza a plataforma Voyager para aceder aos objectos remotos de uma forma transparente. A servlet não necessita por isso de correr na mesma máquina que o servidor de suporte de agentes, constituindo isso uma vantagem em termos de distribuição do peso computacional.

Para se iniciar o contacto com um agente é necessário informar o servlet do agente a contactar, bem como fornecer o username e password que dão acesso a esse agente. Essa informação é enviada através de um POST http das variáveis respectivas. Eis um exemplo de um form html que contém a informação necessária para comunicar com a servlet no inicio.

```
<form action=http://localhost:8080/servlet/inesc.as.servlet.ASServlet method="POST">
   <input type="hidden" name="message" value="connect">
   <input type="hidden" name="agentname" value="localhost:7777/PID_3|AID_2">
   <input type="hidden" name="password" value="anonymous">
   <input type="hidden" name="username" value="anonymous">
   <input type="submit" value="Connect">
</form>
```

Feito o primeiro contacto é criada uma nova sessão de comunicação com o browser na servlet com vista a implementar a manutenção de estado. A informação relativa ao id do agente, username e password é desta forma passada uma só vez já que a servlet mantém o objecto que implementa a sessão (ServletSession) em memória durante a duração da sessão http. Esta solução exige naturalmente que o browser suporte cookies.

Os parâmetros fornecidos à servlet são então passados pela Servlet para os agentes por intermédio da classe ServletSession (ver diagrama de classes apresentado em baixo). No lado do agente deverá existir um objecto sessão que saiba responder ás mensagens enviadas pela servlet. A solução oferecida pela plataforma consiste em criar, para cada agente, um objecto que derive da classe KQMLServletSession. Existem nesta classe alguns métodos que deverão ser implementados pelo programador que aqui destacamos:

init – invocado na criação da sessão, este método permite efectuar inicialização do estado do objecto.

userConnect – invocada automaticamente quando é feito o primeiro contacto (isto é, a sessão se inicia) a partir da servlet. Este método é utilizado geralmente para retornar a página inicial do agente.

userDisconnect – invocada automaticamente quando a sessão é terminada.

httpRequestHandler – método que permite captar a informação enviada pela servlet. Este método recebe um objecto da classe HttpStringParameters que contém a informação do form da página que é submetida pelo browser web. O programador poderá a partir daqui captar os comandos submetidos através da função de ajuda messageKeyIs. Esta função retira do objecto

HttpStringParameters o valor do campo “message” do form e que contém o comando a enviar ao agente. Aqui deixamos um pequeno exemplo de como estas funções podem ser usadas:

```
public Object httpRequestHandler(HtttpStringParameters message){

   // menu principal
    if (messageKeyIs("Menu",message))
              return respondPrint(view.getInitialPage());

    if (messageKeyIs("Ver Estatisticas",message))
              return getEstatisticasChoice(message);
    return respondPrint(view.getErrorPage("Erro","Mensagen desconhecida"));
}
```

Diagrama de classes da AgentSpace Servlet

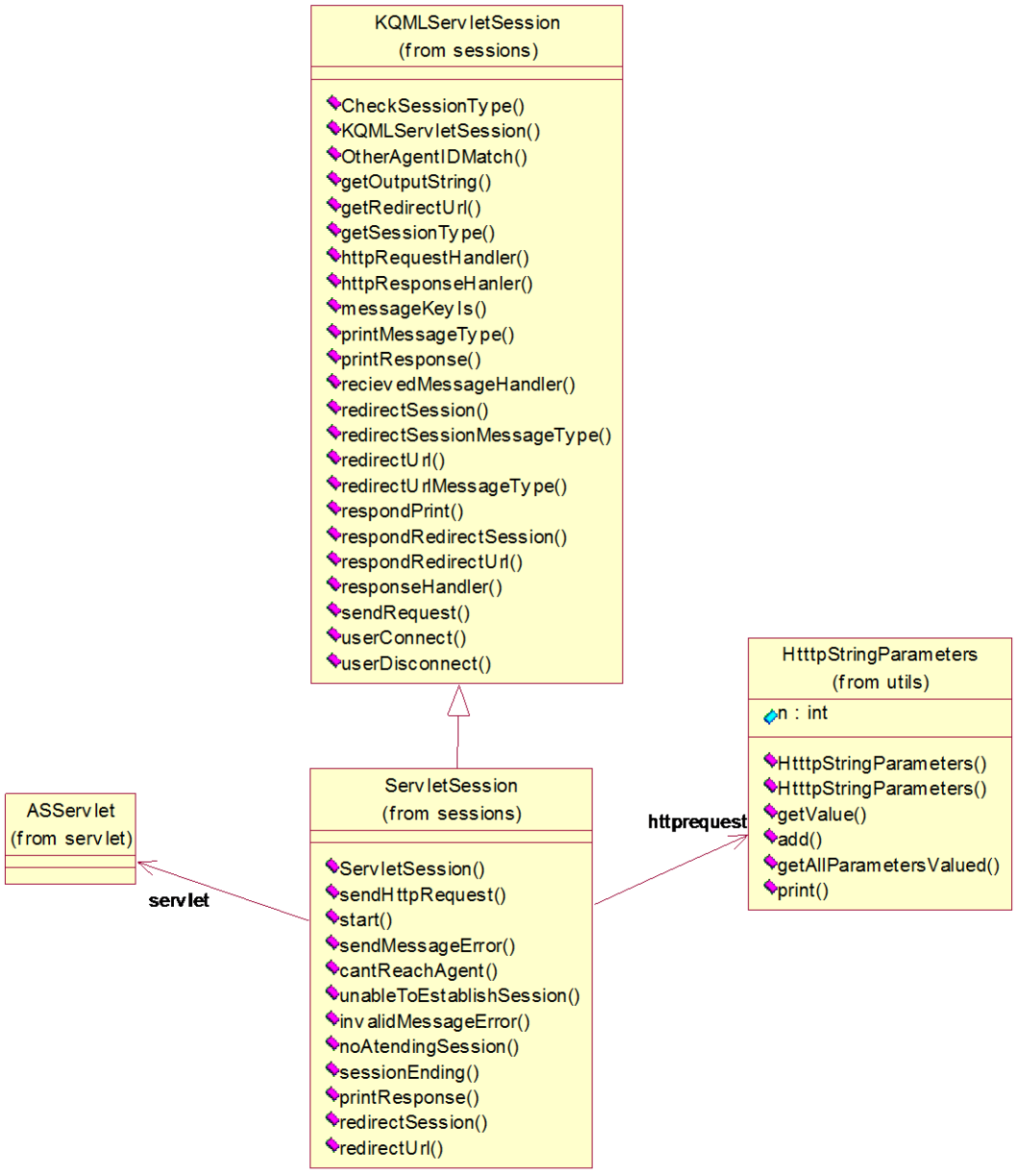

Figura 19 - AgentSpace Servlet (Diagrama de classes)

6.3 Interfaces HTML

Foi ainda desenvolvido um conjunto de classes que visam facilitar a construção das interfaces html nos agentes e tornar o código menos confuso e consequentemente mais legível, já que os tags html são construídos de forma transparente. Além disso, é facilitada a construção dos forms, essenciais em todas as páginas html retornadas pelo agente, já que contém os campos (ainda que muitas vezes possam ser escondidos) que irão permitir continuar a conversão com o agente. As classes desenvolvidas estão ilustradas no diagrama de classes que se segue.

Diagrama de Classes das Interfaces HTML

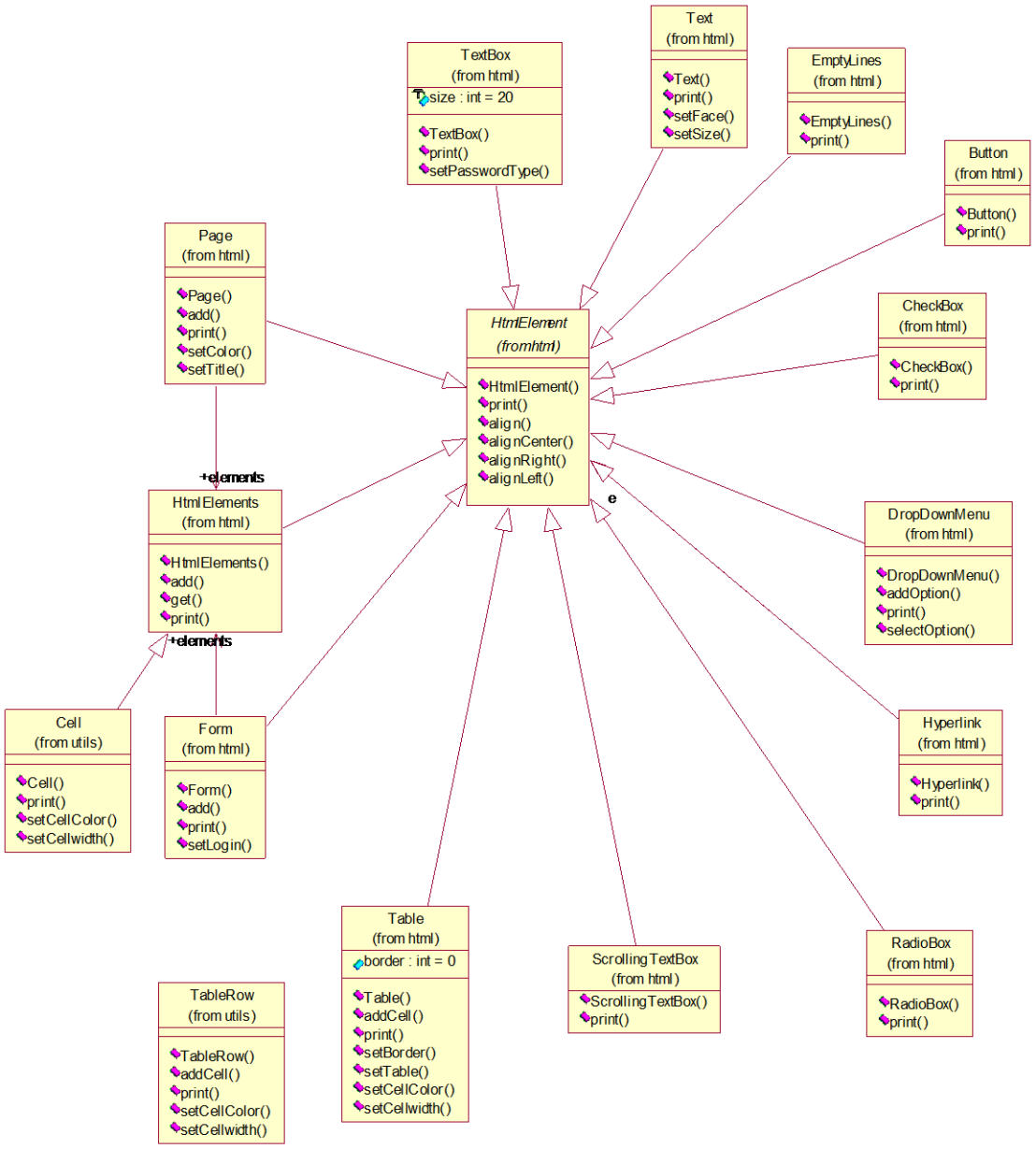

Figura 20 - Interfaces HTML (Diagrama de classes)

7 Exemplos de utilização da DevAPI

7.1 Aplicação myGlobalNews

O **myGlobalNews** foi um projecto originalmente realizado durante o ano lectivo de 1999/2000, no âmbito do trabalho final de curso da licenciatura de Matemática Aplicada e Computação, ramo de Ciência da Computação, do IST, Universidade Técnica de Lisboa.

O objectivo desse trabalho foi construir uma aplicação em **Java**, baseada em agentes sobre a plataforma **AgentSpace**, que disponibilizasse na Internet um serviço de notícias electrónicas personalizadas.

Pretende-se implementar um processo automático de recolha de notícias de diversas fontes de informação (jornais, revistas, agências noticiosas, rádios, bolsas de valores, etc.) e inserção numa base de dados, com a finalidade de fornecer ao utilizador apenas as notícias que lhe interessam. Para isso, é dada ao utilizador a possibilidade de seleccionar termos e fontes de informação sobre os quais pretende visualizar notícias.

Além disso, o utilizador que pretenda usufruir de mais do que uma simples pesquisa momentânea, tem a possibilidade de se registar no myGlobalNews passando a ser um utilizador registado. Como tal, tem uma interface pessoal onde pode:

(1) visualizar as notícias da sua preferência;

(2) efectuar pesquisas momentâneas sem limite do número de fontes e termos;

(3) ser notificado conforme a periodicidade e forma de notificação pretendidas, diminuindo consideravelmente o tempo que teria de dispender à procura das notícias;

(4) obter os contactos de outros utilizadores registados com quem partilhe interesses, ou seja, pertençam à sua comunidade de interesse.

A aplicação foi reformulada por forma a dotar a aplicação de uma interface web acessível por um browser web em contraposição à solução existente de interfaces java, notoriamente mais pesadas e menos práticas em termos de utilização real do sistema. Foi também objectivo melhorar a robustez geral da aplicação submetendo-a a testes mais intensivos. O principal objectivo foi no entanto demonstrar algumas das funcionalidades oferecidas pela DevAPI , nomeadamente:

- A utilização da **AgentSpace servlet** como uma solução genérica para dotar os agentes AgentSpace de interfaces web em html.
- A utilização de **sessões de comunicação** entre os agentes e a servlet como meio de modularização das funcionalidades dos agentes e uma solução mais elegante para implementar a manutenção de estado que a restrição deste tipo de interfaces impõe.
- A utilização das potencialidades oferecidas pela classe de **agente AgentSapiens** no que respeita ao **envio de emails**, encapsulando a comunicação com o agente de sistema

AgentSpace MailAgent e tendo assim uma forma centralizada de envio e gestão do servidor de mail e simplificando o desenvolvimento de aplicações AgentSpace.

- A utilização das potencialidades oferecidas pela classe de agente AgentSapiens no que respeita à **calendarização do envio de mensagens** para outros agentes, encapsulando a comunicação com o agente de sistema AgentSpace **ReminderAgent**, fazendo assim uso de um agente genérico de envio de mensagens calendarizadas e simplificando o desenvolvimento da aplicação

- A utilização das **classes de desenvolvimento de páginas html** que facilitam o processo de construção das mesmas por forma a que estas estejam preparadas para comunicar com os agentes através da servlet.

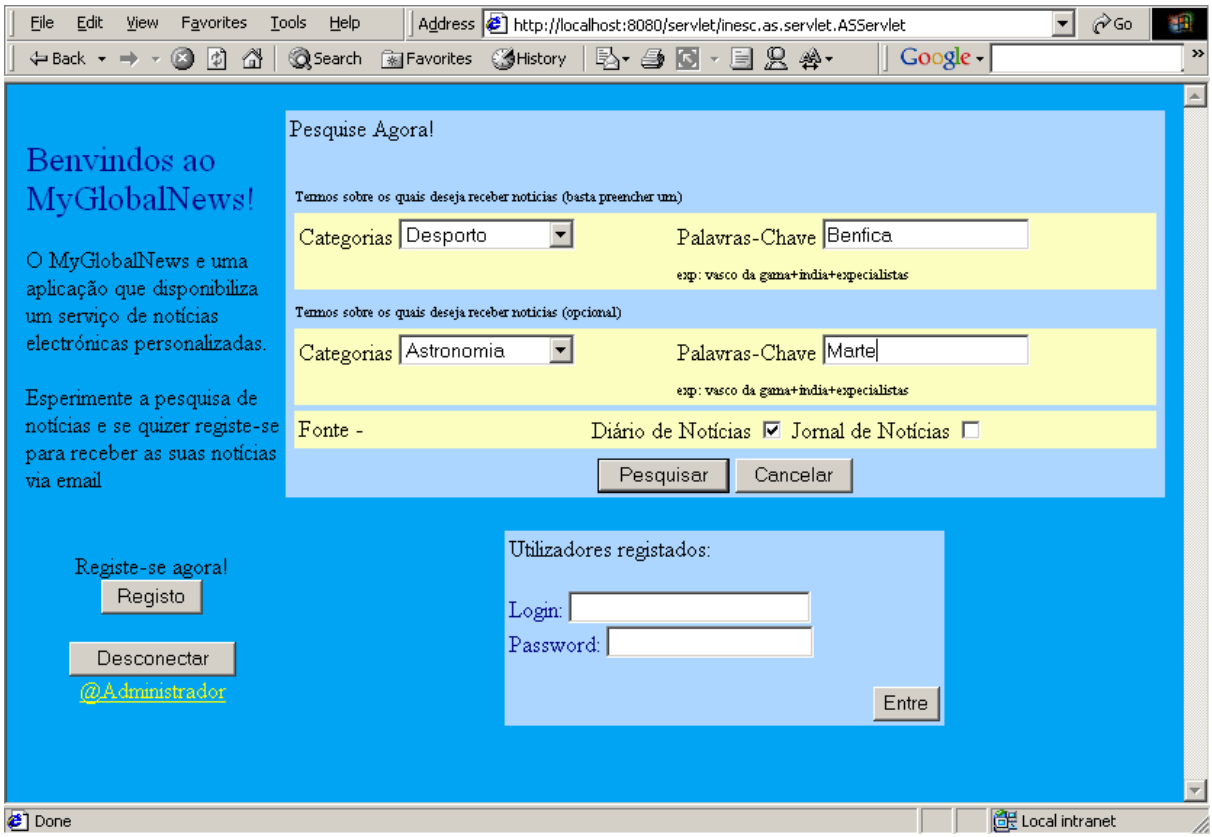

Figura 21 - Aplicação myGlobalNews (Interface do utilizador)

Em anexo a este relatório é apresentada documentação mais detalhada sobre esta aplicação, nomeadamente nos apêndices “Aplicação Exemplo myGlobalNews - Manual do Utilizador”, “Aplicação Exemplo myGlobalNews - Manual Técnico e “Aplicação Exemplo myGlobalNews - Código”.

7.2 AgentSpace Photopaint

O AgentSpace Photo Paint consiste num conjunto de aplicações que interagem entre si. A Interacção pode ser vista como um maneira prática e inovadora de disponibilizar e adquirir produtos através da Internet, tirando partido das vantagens fornecidas pela computação distribuída.

O conjunto de aplicações consiste em:

- uma **aplicação de edição e tratamento de im**agens,
- uma **aplicação que disponibiliza agentes de procura de imagens** e que permite a esses mesmos agentes saberem quais as aplicações que disponibilizam imagens,
- e **aplicações que disponibilizam e/ou vendem imagens**.

Temos assim 3 tipos de aplicações para 3 tipos de utilizadores distintos.

Sucintamente, esta aplicação visa demonstrar a utilização dos padrões **Itinerary, Courier, Secretary Courier, Receptionist e Session**.

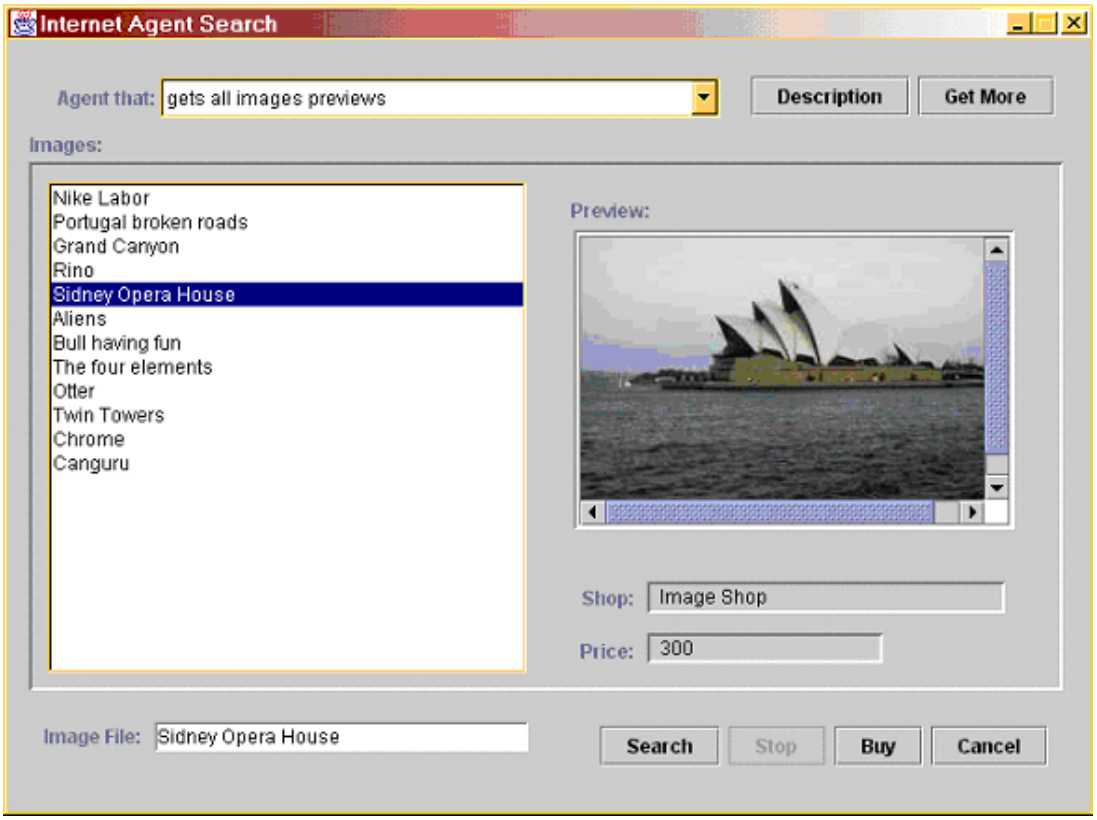

Figura 22 - Aplicação PhotoPaint (Interface do utilizador)

Em anexo a este relatório é apresentada documentação mais detalhada sobre esta aplicação, nomeadamente nos apêndices “Aplicação Exemplo AgentSpace PhotoPaint - Manual do Utilizador”, “Aplicação Exemplo AgentSpace PhotoPaint - Manual Técnico” e “Aplicação Exemplo AgentSpace PhotoPaint - Código”.

8 Conclusões

8.1 Sobre o trabalho realizado

Foi preocupação no desenvolvimento desta plataforma dotá-la dos mecanismos básicos para o desenvolvimento de aplicações baseadas em agentes no contexto da internet. Foram por isso escolhidas áreas de desenvolvimento que considerámos mais importantes neste contexto e que por isso seria de maior relevância os agentes terem. Foi por isso dado primazia a áreas como a comunicação entre agentes , delegação e calendarização de tarefas dos agentes e mobilidade dos agentes. Julgamos que estes mecanismos contribuem para a autonomia dos agentes, sendo certo no entanto que muitos outros mecanismos seriam desejáveis, nomeadamente aqueles que possam dotar o agente de alguma inteligência, nomeadamente aqueles que abrangem as áreas do planeamento, aprendizagem, reconhecimento de voz, interfaces de agentes, etc. Daí a classe de agentes AgentSapiens não ter sido denominada AgentSapiensSapiens!

Os exemplos implementados myGlobalNews e AgentSpace Photo Paint, são exemplos claros da utilização da biblioteca de classes e interfaces desenvolvidas neste trabalho para a API do AgentSpace. Demonstram bem como a utilização de padrões de software pode contribuir de maneira significante para o paradigma da Programação Orientada a Agentes trazendo não só, maior expressividade , mais estruturação no desenvolvimento de código, como também a diminuição no esforço do desenvolvimento de aplicações que escolham utilizar esta tecnologia.

Com o exemplo AgentSpace Photo Paint demonstramos como utilizar os padrões de software Secretary, SecretaryCourier, Itinerary e Receptionist, evidenciando a simplicidade da sua utilização e uma maneira de como estes podem ser utilizados em conjunto de modo a trazer uma nova forma de adquirir e disponibilizar: produtos (como é o caso das imagens neste exemplo) e funcionalidades que podem tirar partido da computação distribuída (como é o caso dos agentes de procura). Pode-se fazer uma transposição deste exemplo para o desenvolvimento de aplicações de Information Retrieval. Para isso, basta associar os agentes do tipo SecretaryCourier com os agentes que realizam as tarefas de information retrieval, o agente do tipo Secretary com o que recebe e disponibiliza os resultados e o agente do tipo Receptionist com o agente que disponibiliza não só, novos agentes de information retrieval como informação acerca das fontes de informação.

Com o exemplo myGlobalNews ficou demonstrado o uso da AgentSpace Servlet, permitindo desta forma dar uma melhor acessibilidade a esta aplicação e tornar a sua divulgação mais fácil com a publicação das interfaces dos agentes na internet. Permitiu ainda demonstrar a utilização dos agentes de sistema MailAgent e ReminderAgent e como a comunicação com estes passou a ser feita de uma forma transparente a partir do momento em que os agentes myGlobalNews passaram a derivar da classe AgentSapiens.

8.2 Trabalho Futuro e Melhorias

Convém inicialmente referir que dado a vastidão de mecanismos que seria possível dotar os agentes por forma a torná-los cada vez mais ricos (no sentido da inteligência e da autonomia dos

agentes) as sugestões que aqui deixamos não são mais do que uma percentagem muito reduzida (que não nos atrevemos sequer a quantificar) do trabalho futuro e melhorias ao existente que é possível realizar nesta área. Deixamos no entanto algumas ideias que retirámos do trabalho de investigação que foi realizado no início deste Trabalho Final de Curso.

Além dos padrões implementados, muitos outros ficaram obviamente por implementar. Alguns desses padrões encontram-se documentados em [0,3]. Alguns padrões que destacamos são o **Meeting with Moderator** [27], **Master-Slave** [3] e o **Locker** [3].

Além da implementação de padrões, a criação ou integração no AgentSpace e DevAPI de APIs existentes com soluções sobre outras áreas do desenvolvimento de software (como reconhecimento de voz, algoritmos de aprendizagem, data mining, reconhecimento de imagem, e muitos outros).

Ficou também por realizar uma componente importante de assistência ao desenvolvimento, nomeadamente uma interface que permitisse de forma visual desenvolver as ABA ou esqueletos dessas aplicações. Uma primeira solução que visionamos para desenvolver esta solução seria usando a ferramenta CASE Rational Rose que conjuntamente com Rose Scripts constitui uma ferramenta de desenvolvimento visual bastante potente.

Em relação ao trabalho implementado aqui ficam algumas sugestões de melhorias a realizar:

- Neste momento, o agente MailAgent envia mensagens de email em nome dos agentes. Era interessante que o **MailAgent pudesse ser estendido para suportar a recepção de mensagens** vindas dos utilizadores ou administradores dos agentes, possibilitando assim mais um meio de comunicação com os agentes. Uma solução possível para esta funcionalidade poderia ser obtida através da biblioteca JavaMail, que permite a criação de um servidor de mail pop3 e das respectivas caixas de email (inbox). O agente MailAgent poderia assim redireccionar as mensagens para os agentes respectivos.

- Era também desejável que o agente **ReminderAgent pudesse receber acoplado às mensagens objectos genéricos**. Sendo o envio de objectos genéricos “serializáveis” já possível, era suficiente que o agente ReminderAgent “serializasse” os objectos directamente para a base de dados onde guarda as mensagens calendarizadas.

- Para além dos mecanismos básicos de comunicação, gestão de tarefas, etc., era desejável começar a dotar os agentes do AgentSpace de alguns mecanismos que lhes confiram maior inteligência. Um contributo importante seria **estender o padrão Itinerary de modo a adicionar um componente de planeamento** permitindo assim aos agentes uma gestão mais eficiente das suas tarefas e um melhor aproveitamento da sua mobilidade.

- A classe **KQMLSession necessita de continuação no seu desenvolvimento** por forma a permitir a conversão do objecto KQMLMessage em formato de texto segundo a sintaxe do protocolo e consequentes testes com outros agentes KQML que corram sobre outras plataformas de suporte de agentes.

- O padrão **Secretary necessita de posterior desenvolvimento**, já que ficaram funcionalidades possíveis por implementar, como decorre da comparação entre a definição do padrão aqui apresentado e da implementação realizada. É necessário por isso a extensão do padrão Secretary/SecretaryCourier com a adição de mais comportamentos genéricos úteis ao agente SecretaryCourier.

9 Referências

[1] "A case for Mobile Agent Patterns", Dwight Deugo, Michael Weiss, Carleton University, 1999

[2] "A case for Mobile Agent Patterns", Dwight Deugo, Michael Weiss, Carleton University, 1999

[3] "Agent Design Patterns - Elements of Agent Application Design", Yariv Aridov, Danny B.Lange, Proceedings of the Mobile Agents'98, ACM Press, 1998.

[4] "Agentes de Software na Internet", Alberto Silva, Centro Atlântico - ISBN 972-8426-10-0, 1999.

[5] "An overview of Agent Space: A next Generation Mobile Agent System", Alberto Silva, Miguel Mira da Silva, José Delgado – Proceedings of the Mobile Agents'98, ACM Press, 1998.

[6] "Communication as a Means to Differentiate Objects, Components and Agents", Dwight Deugo, Franz Oppacher, Michael Weiss", Carleton University, 1999.

[7] "KQML - A language and Information Exchange", Tim Finnin, Rich Fritzon, University of Maryland, Technical Report CS-94-02,1994.

[8] "Patterns as a Means for Intelligent Software Engineering", D.Deugo, F.Oppacher, J.Kerester, I.Von Otte - IC-AI99, 502SA, 1999.

[9] "Plataforma para desenvolvimento de Agentes de Software: Visão", Ivo Conde e Silva e Nuno Meira

[10] "The Agent Pattern for Mobile Agent Systems", Alberto Silva, José Delgado, 1998, In European Conference on Pattern Languages of Programming and Computing, EuroPlop'98.

[11] "A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems", N. Jennings, K. Sycara, M. Wooldridge., Kluwer Academic Press, 1998.

[12] "Agent Construction Tools" - http://www.agentbuilder.com/AgentTools/index.html

[13] "AgentSpace: An Implementation of a Next-Generation Mobile Agent System", Alberto Silva, Miguel Mira da Silva, José Delgado, (Mobile Agents'98) Lecture Notes in Computer Science, 1477, Springer Verlag, 1998.

[14] "Concurrency, a Case Study in Remote Tasking and Distributed IPC", D. Milojicic et al., Proceedings of the 29th Annual Hawaii International Conference on System Sciences, January 1996.

[15] "Design Patterns: Elements of Reusable Object-Oriented Software", E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison-Wesley, Reading, MA, 1995.

[16] "Designing a Process Migration Facility: The Charlotte Experience", Y. Artsy, R.Finkel. IEEE Computer, September 1989.

[17] "Extending UML for Agents", James Odell, H. Parunak, Bernhard Bauer, AOIS Worshop at AAAI 2000.

[18] "FIPA Contract Net Protocol Specification", Foundation for Intelligent Physical Agents, 2000

[19] "Is it an agent or just a program? A Taxonomy for Autonomous Agent", S. Franklin, A. Graesser, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, 1996.

[20] "ObjectSpace *Voyager*" – http://www.objectspace.com/products/voyager/

[21] "OMG Unified Modeling Language Specification. Version 1.3", Object Managament Group, UML Revision Task Force., 1999.

[22] "Pattern Languages of Program Design" 1-4 (Software Patterns Series), Addison-Wesley, 1995-1999.

[23] "The Timeless Way of Building", C.Alexander, Oxford University Press, 1977.

[24] "The Unified Modeling Language User Guide", G.Booch, J.Rumbaugh, I. Jacobson, Addison Wesley 1999.

[25] "Towards a Reference Model for Surveying Mobile Agent Systems", Alberto Rodrigues da Silva, Artur Romão, Dwight Deugo and Miguel Mira da Silva, Autonomous Agents and Multi-Agent Systems Journal. Kluwer Publisher (to be Published).

[26] "UML Homepage" – http://www.rational.com/uml/

[27] "A Set of Agent Patterns for a More Expressive Approach", Nuno Meira, Ivo Conde e Silva, Alberto Silva, Proceedings of the European Conference on Pattern Languages of Programming (EuroPlop) 2000.