INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Plataforma de suporte ao alinhamento de agendas em dispositivos móveis

JAVIER ANTÓNIO D'ERCOLE

(Licenciado)

Trabalho Final de Mestrado para obtenção do grau de Mestre

em Engenharia Informática e de Computadores

Orientador:

Professor Paulo Trigo

Júri:

Presidente: Professor Pedro Pereira

Vogais:

Professor Luis Moniz

Professor Paulo Trigo

Outubro de 2012

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Plataforma de suporte ao alinhamento de agendas em dispositivos móveis

JAVIER ANTÓNIO D'ERCOLE

(Licenciado)

Trabalho Final de Mestrado para obtenção do grau de Mestre

em Engenharia Informática e de Computadores

Orientador:

Professor Paulo Trigo

Júri:

Presidente: Professor Pedro Pereira

Vogais:

Professor Luis Moniz

Professor Paulo Trigo

Outubro de 2012

Agradecimentos

À minha família, por compreenderem a minha ausência durante a elaboração deste trabalho e por me apoiarem incondicionalmente no seguimento da minha carreira académica.

Aos meus amigos, pelos momentos de escape e descontração proporcionados, sempre agradáveis e revigorantes.

Ao meu orientador, professor Paulo Trigo, pela paciência, disponibilidade e constante motivação que foram fundamentais para a conclusão deste trabalho.

Resumo

Desde o tempo em que as agendas eram feitas de papel até aos dias de hoje em que estão disponíveis em formato eletrónico, "online", em computadores pessoais ou em dispositivos móveis, as agendas sempre foram uma ferramenta essencial de suporte à organização do tempo no nosso dia-a-dia. No entanto, a marcação de eventos entre vários participantes obriga a um processo de negociação manual de alinhamento das agendas.

Este projeto implementa um sistema multi-agente para o alinhamento de agendas em dispositivos móveis, onde os agentes negoceiam de modo autónomo em representação de utilizadores considerando as suas preferências públicas e privadas. Estes agentes atuam sobre a plataforma multi-agente JADE [1], na qual comunicam e negoceiam entre si numa linguagem bem definida respeitando uma ontologia e um protocolo de interação baseado em Iterated Contract-Net. A negociação tem por objetivo procurar o alinhamento das disponibilidades fornecidas pelos vários participantes suportada por mecanismos de votação de forma a eleger a solução admissível mais adequada.

O sistema suporta a negociação com agentes que podem estar temporariamente indisponíveis e tem a capacidade de utilizar "dicas" baseadas em informação histórica de negociações prévias para otimizar o processo de negociação procurando a melhor solução admissível no menor espaço de tempo possível.

O sistema superou com sucesso os testes de carga e concorrência efetuados, assim como da eficiência da utilização das dicas no processo de negociação, providenciando sempre que possível uma solução válida.

Keywords: Agentes autónomos e móveis, Negociação em Sistemas multi-agente, Negociação suportada em ontologias.

[1] Java Agent DEvelopment Framework

Abstract

Since the time agendas where made of paper to nowadays, where we have electronic agendas that are available online, on personal computers and mobile devices, they always been an essential tool for our day-by-day time management. Never the less, scheduling a meeting and negotiating timeslots still is a manual task.

We present a multi-agent approach to schedule meetings (on the move) using mobile devices where agents autonomously negotiate the time slots that best suit their agenda regarding each one's private and public preferences. Agents act on the JADE[2] platform, negotiating and communicating respecting a well-defined language over an ontology and a interaction protocol based on Iterated Contract-Net. The negotiation goal is to find an alignment between participant's availabilities and elect the best suitable timeslot by voting.

The approach supports negotiation with temporarily off-line agents and uses hints based on historical information about previous negotiations for optimizing the negotiation process.

The system successfully performed the load and concurrency tests, as well as the efficiency of using hints on the negotiation processes, always providing valid solutions.

Keywords: Mobile and Autonomous Agents, Negotiation on Multi-agent system. Ontology based negotiation.

[2]Java Agent DEvelopment Framework

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Os dispositivos móveis estão agora a viver os seus anos de ouro, tornando-se cada vez mais populares e com maiores capacidades de processamento e conectividade. Uma das funcionalidades destes dispositivos é a sua agenda, que permite aos seus utilizadores gerir e consultar os seus compromissos, e inclusive mantê-las atualizadas "online". No entanto, todo o processo de negociação e marcação de novos eventos continua a ser uma tarefa manual.

Este projeto tem como principal contributo a automatização do processo de negociação típico de uma marcação de eventos, na qual a negociação é efetuada por agentes inteligentes em representação dos seus utilizadores. Os agentes comunicam entre si respeitando uma ontologia bem definida, consultando as agendas e tomando decisões com base nas preferências definidas pelos utilizadores.

Este projeto explora abordagens baseadas em sistemas multi-agente no desenvolvimento de uma ferramenta que contribui de forma eficiente como uma possível solução ao problema do alinhamento de agendas, baseando a sua implementação em tecnologias mais atuais em termos de plataforma e de dispositivos finais onde os agentes são executados.

1.1 Resumo da abordagem

A abordagem implementada consiste na utilização da plataforma multi-agente JADE[1] onde residem os agentes, que comunicam entre si utilizando uma ontologia bem definida encapsulada nas mensagens ACL[2] de forma a entenderem-se inequivocamente. Os agentes móveis executam sobre "Android" com recurso à extensão JADE-LEAP[3] para uma implementação adequada a dispositivos com recursos de memória, processamento e conectividade mais limitados.

As preferências dos utilizadores são definidas nos dispositivos e permitem aos agentes tomar decisões em nome dos utilizadores de acordo com as restrições definidas.

[1] Java Agent DEvelopment Framework
[2] Agent Communication Language
[3] Java Agent DEvelopment Framework - Lightweight Extensible Agent Platform

A marcação de eventos é iniciada por um dos agentes que deve indicar essencialmente a janela temporal para a qual gostaria de marcar a reunião, assim como a lista dos participantes e a sua importância para a participação no evento. Com base nesta informação, os agentes negoceiam entre si as várias possibilidades com maior grau de preferência global e elegem a hora do evento com recurso a uma votação. Por fim, os eventos são marcados, de modo automático, nas agendas pessoais de cada um dos participantes.

O essencial deste trabalho foi apresentado na CETC 2011 - Conference on Electronics, Telecommunications and Computers , tendo sido considerado o "best paper in the computer science track" [D'Ercole and Trigo(2011)].

1.2 Estrutura do relatório

Este relatório está organizado em sete capítulos. Neste primeiro capítulo é feita uma introdução ao tema do alinhamento de agendas em dispositivos móveis.

No segundo capítulo é feita uma análise aos trabalhos já elaborados neste mesmo âmbito.

No terceiro capítulo descreve-se a solução implementada em termos de ferramentas e tecnologias utilizadas, como a plataforma JADE e a utilização e implementação dos agentes autónomos. É descrita também a abordagem utilizada para a implementação das preferências de utilizador e a introdução ao conceito do Agente das Dicas.

O quarto capítulo faz uma introdução ao conceito e necessidade de utilizarmos uma ontologia e descreve a ontologia implementada para a solução.

O quinto capítulo aborda em detalhe o tema da negociação e tomadas de decisão, comunicação entre os agentes e os protocolos de interação utilizados e implementados.

No sexto capítulo são apresentados os testes efetuados ao sistema, resultados obtidos e respetivas conclusões.

Por fim, no sétimo capítulo temos as conclusões sobre o trabalho realizado e perspetivas de trabalho futuro.

Capítulo 2

Trabalho Relacionado

A utilização de sistemas multi-agente para o agendamento de eventos tem sido abordada em alguns trabalhos nos últimos anos. Um dos primeiros trabalhos nesta área foi o de [Mattern and Sturm(1989)] que desenvolveu um protótipo de um sistema automático para o agendamento distribuído de eventos, o qual incluía também na sua arquitetura o sistema de troca de mensagens entre os agentes. Nesta abordagem, o agente que pretende agendar o evento (anfitrião) envia inicialmente as suas propostas com os períodos de tempo sugeridos para os agentes dos restantes participantes, os quais verificam nos seus calendários a sua disponibilidade. O evento é marcado quando todos os participantes aceitarem a proposta do anfitrião. Se não chegarem a um consenso, o sistema prevê também um mecanismo de resolução de conflitos que tem em conta fatores como a prioridade do evento ou a importância da participação de cada um dos intervenientes.

Na proposta de [Mattern and Sturm(1989)] foi especificada toda a arquitetura de um sistema de negociação, desde a estratégia de negociação dos agentes, as comunicações entre eles, e sugestões para um possível ambiente gráfico das aplicações.

Posteriomente, [Eisinger and Elshiewy(1992)], no projeto MADMAN[1], criou um agente que recolhe as preferências de todos os intervenientes, assim como as suas disponibilidades. Com base nesta informação é calculado um perfil de preferências conjuntas que é utilizado para calcular a hora do evento a marcar.

Em 1992, [Sen and Durfee(1992)], utilizaram o protocolo de interação Contract-Net e abordam a negociação como se de um leilão se tratasse. Neste trabalho, o agente que pretende agendar o evento, o anfitrião, comunica com os outros agentes participantes no evento, os convidados, partilhando inicialmente os dados do evento que se pretende marcar. A partir deste momento, os convidados enviam ao anfitrião propostas das suas disponibilidades. O processo termina quando é encontrada uma data consensual por parte de todos os intervenientes, ou então é cancelado o agendamento se uma data não for acordada.

O trabalho de [Jennings and Jackson(1995)] tem uma abordagem diferente pois utiliza uma negociação em duas fases. Numa primeira fase, os agentes comprometem-se com a intenção de participar no evento e só depois é que iniciam a negociação. Neste caso, os agentes convidados, ao enviarem ao anfitrião as suas propostas, estão também a reservar temporariamente nas

[1] Multi-Agent Diary MANager

suas agendas os períodos de tempo propostos. Desta forma, qualquer outra negociação que corra em paralelo, terá conhecimento dos períodos possivelmente ocupados. Estes períodos são também classificados com uma prioridade e assume-se que estes poderão ser ocupados por eventos com uma prioridade superior, podendo ser necessário proceder a um reagendamento.

Num trabalho mais recente, [Niederer and Schatten(2009)], implementaram um protótipo de um agente para dispositivos móveis "Android" que comunicam entre si utilizando o protocolo XMPP[2], mas as mensagens não são baseadas numa ontologia de negociação mas sim na troca de objetos Java.

[2]Extensible Messaging and Presence Protocol

Capítulo 3

Modelo Proposto

Como automatizar o processo de alinhamento de agendas utilizando dispositivos móveis? Para construir uma solução para esta questão foram analisadas duas abordagens distintas:

(a) Solução cliente-servidor, onde um servidor central armazena todas as agendas e preferências dos intervenientes e toma todas as decisões. Nesta abordagem os clientes (dispositivos móveis) são meros interfaces para o utilizador e não tem qualquer autonomia para tomar decisões. Ver figura 3.1.

(b) Solução ponto-a-ponto, na qual as agendas e informação sobre as preferências dos intervenientes residem nos próprios dispositivos. Neste caso os dispositivos são agentes autónomos, capazes de comunicar entre si e de tomar decisões. Ver figura 3.2.

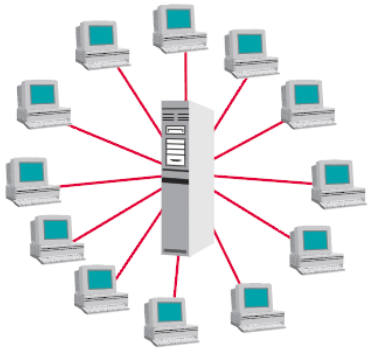

Figura 3.1: Arquitectura Cliente-Servidor.

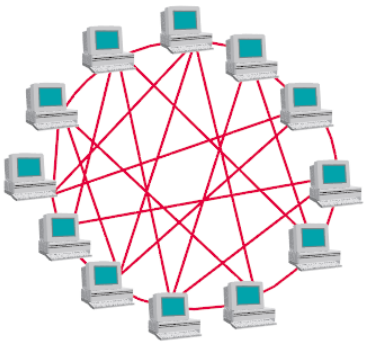

Figura 3.2: Arquitectura Distribuida Ponto-a-Ponto.

Na primeira abordagem, o servidor central ao funcionar como um centralizador da informação e do processo de negociação, torna-se um ponto único de falha, pelo que bastaria este serviço estar indisponível para todo o sistema falhar. A centralização pode também trazer outros problemas ao nível da segurança e privacidade, uma vez que toda a informação de agendas e preferências se encontra num único local, podendo haver clientes que não se sintam "confortáveis" com isso. Nesta abordagem os clientes comunicam apenas com o servidor e não entre si. Por sua vez, o servidor apenas consegue comunicar com os clientes depois destes terem tomado

a iniciativa de se ligarem ao servidor. Um modelo com estas restrições não é o mais apropriado pelo que foi seguida a abordagem de um sistema distribuído ponto-a-ponto, onde os clientes podem comunicar livremente entre si, iniciar ou responder a solicitações, serem proativos, funcionarem como um verdadeiro sistema distribuído onde tanto a lógica da aplicação como os dados estão distribuídos, e onde os clientes podem ligar ou desligar-se da rede sem impacto para todo o sistema. [Bellifemine et al.(2003)Bellifemine, Caire, Poggi, and Rimassa]

Para a implementação do sistema multi-agente foi utilizada a plataforma [JADE(2012)], que consiste num ambiente e numa plataforma para o desenvolvimento de agentes em Java. A plataforma JADE fornece um ambiente distribuído para alojar os agentes, assim como todos os mecanismos necessários para estes comunicarem entre si. Esta plataforma é uma implementação de referência dos modelos definidos pela [FIPA(2012)], organização internacional que define os standards relacionados com agentes, nomeadamente a especificação de agente e seu ciclo de vida, estrutura e transporte de mensagens, protocolos de interação, ontologias e segurança.

3.1 A plataforma JADE

A plataforma JADE é uma plataforma para o desenvolvimento de sistemas multi-agente baseados numa arquitetura distribuída ponto-a-ponto, onde os agentes podem facilmente comunicar entre si e entrar ou sair do sistema de acordo com as suas necessidades. A análise detalhada da plataforma não está no âmbito deste projeto, apenas se descrevem algumas das suas principais componentes, representadas na figura 3.3

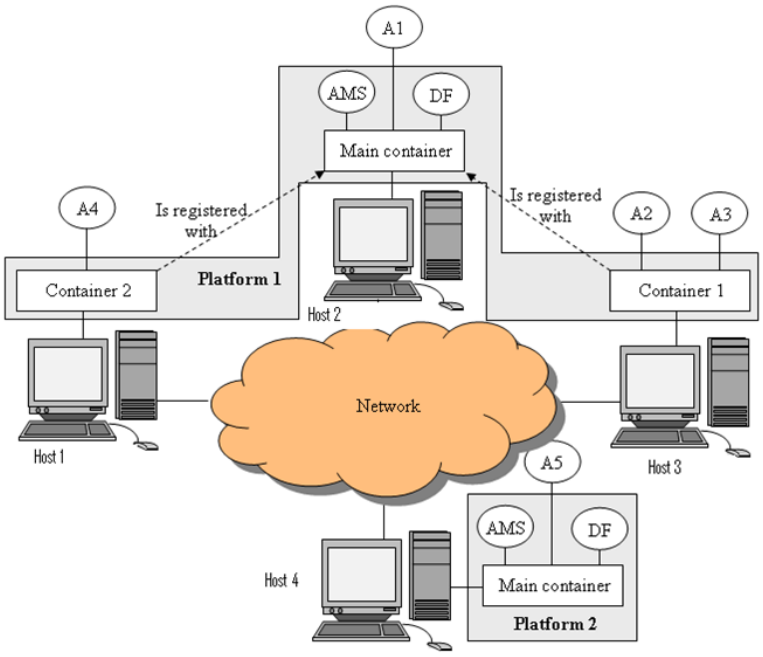

Figura 3.3: JADE Platform

Agentes São as entidades autónomas que comunicam entre si, tomam decisões e agem em

nome do individuo.

Plataforma A plataforma corresponde ao conjunto de todos os componentes que formam o sistema multi-agente.

Contentores Um contentor é um espaço controlado dentro da plataforma que normalmente corresponde a uma JVM[1], dentro da qual podem ser executados os agentes.

Contentor Principal É o contentor onde são executados os agentes especiais, nomeadamente o DF[2] e o AMS[3].

DF - Directory Facilitator Este agente fornece o serviço de "páginas amarelas" para a plataforma, ou seja, todos os agentes que queiram prestar algum tipo de serviço dentro de uma plataforma, devem registar esses seus serviços no DF, para que estes possam ser pesquisados e encontrados facilmente por quem os requisitar.

AMS - Agent Management System O AMS é um agente que tem de ser único dentro da plataforma, cujas principais funções são controlar os acessos dos outros agentes à plataforma e manter um registo dos agentes e respetivos estados.

3.1.1 JADE-LEAP

Os agentes desenvolvidos sobre a plataforma JADE são otimizados para executarem em ambientes sem grandes limitações de recursos, como memória ou capacidade de processamento, e principalmente em ambientes onde a conetividade não seja reduzida ou intermitente. Para ultrapassar estas limitações surge o JADE-LEAP, uma versão dos agentes JADE especializada para dispositivos móveis. O JADE-LEAP está preparado para suportar quebras de conectividade que são normais nos dispositivos móveis, sendo que o agente corre num modo dividido entre "front-end", que executa no dispositivo móvel, e "back-end", que é executado na própria plataforma JADE, mantendo-se este último vivo mesmo que o "front-end" esteja temporariamente indisponível. [Caire and Pieri(2010)]

A utilização do JADE-LEAP permite então desenvolver os agentes para dispositivos "Android" com suporte para percas de conectividade e com um reduzido consumo de recursos no dispositivo.

3.2 Os agentes

Os agentes são entidades que funcionam de forma contínua e autónoma, atuando em nome dos utilizadores e executam as ações de negociação e registo de novos eventos. De forma a tirar partido da interoperabilidade da linguagem Java e dos agentes JADE, foi implementada uma versão do agente para executar em "Android" e outra para ser executada em computadores pessoais. Para atingir este objetivo foram utilizados três módulos principais:

[1] Java Virtual Machine
[2] Directory Facilitator
[3] Agent Management System

1. A interface gráfica
2. O agente
3. O módulo de negociação

3.2.1 Interface Gráfica

Foram implementadas duas interfaces gráficas para os agentes, uma para "Android" e outra para computadores pessoais.

Na versão "Android", a aplicação é iniciada pela atividade "MainActivity", que além de servir de ponto de entrada, é onde se inicia a ligação do agente à plataforma. As propriedades da ligação são obtidas a partir dos parâmetros de configuração da aplicação e é nesta atividade que se iniciam todas as componentes necessária para a execução do agente, nomeadamente, um serviço que fica a correr em "background" mesmos que a atividade entre em "standby", o contentor onde o agente vai ser executado e finalmente o agente. Esta atividade funciona também como observador do estado do agente. Se o agente terminar a sua execução, envia o sinal "KILL" para a "MainActivity" e a aplicação termina, Quando o agente inicia, é enviado o sinal "START" e é iniciada a atividade "AgentActivity".

A atividade "AgentActivity" está sempre associada a um agente em execução, e caso o agente termine, a atividade termina também e voltamos para a anterior. Esta atividade funciona como uma interface entre a aplicação "Android" e o agente, pois é ela que recebe as notificações enviadas pelo agente sempre que este necessita de comunicar com a aplicação. Também é daqui que é dada a ordem ao agente para iniciar ou cancelar um evento. De notar que mesmo a partir desta atividade, o acesso ao agente não é direto, isto para garantir que quem controla o ciclo de vida do agente continua a ser a plataforma JADE. Para aceder ao agente, é necessário obter o seu controlador e a partir deste chamar o método "getO2AInterface" que retorna o interface implementado pelo agente, como se pode ver na implementação:

```
1 ...
  agentControler = MicroRuntime.getAgent(settingsInterface.getJadeUsername());
3 agentInterface = agentControler.getO2AInterface(AgentInterface.class);
  ...
5 agentInterface.schedule(event);
  ...
```

Obtenção do interface implementado pelo agente.

Na figura 3.4 é possível ver um exemplo da interface gráfica "Android" utilizado para a introdução de um evento. A figura 3.6 mostra a interface para consultar e alterar as propriedades de configuração da aplicação.

A outra interface gráfica foi implementada para utilização em computadores pessoais. Nesta versão, a aplicação desenvolvida permite iniciar a marcação de novos eventos através do preenchimento de um formulário, assim como a definição das preferências do utilizador. Em relação à consulta e registo de novos eventos efetuados pela aplicação, o objetivo (embora não implementado nesta versão para computador pessoal) seria interligar a aplicação com

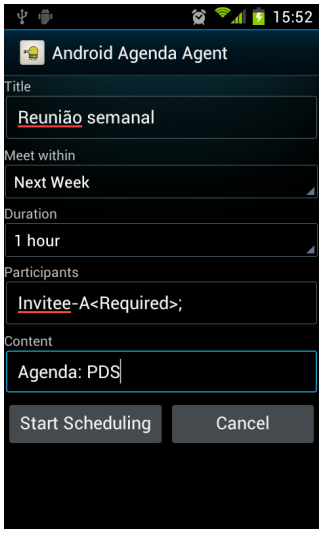

Figura 3.4: Agente "Android" - Marcação de um evento.

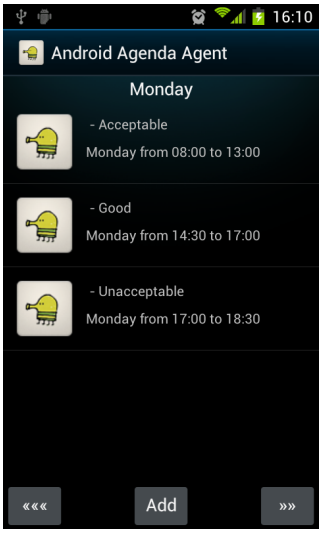

Figura 3.5: Agente "Android" - Definição das Preferências.

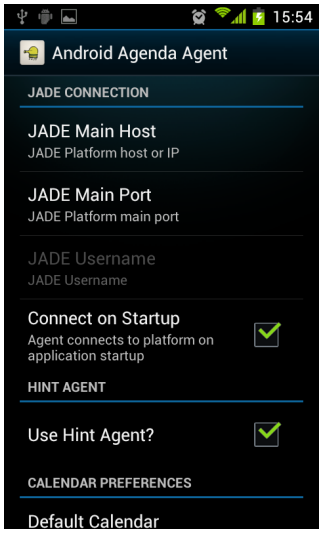

Figura 3.6: Agente "Android" - Configurações do agente.

o "Google Calendar", mediante uma autenticação, e utilizando como interface a "Google Calendar API" [Google(2012)].

3.2.2 Agente

A implementação dos agentes também foi diferenciada entre a versão para "Android" e a versão para computador pessoal por uma questão de simplicidade na implementação, pois em cada uma das versões, é necessário inicializar algumas implementações de classes utilitárias específicas da plataforma onde o agente está a ser executado, nomeadamente:

AbstractCalendarInterface Interface utilizada pelo agente para marcar os eventos no calendário da agenda pessoal. No caso do "Android", são armazenados no calendário da agenda pessoal do dispositivo. No caso do Java para computador pessoal, o objetivo (não implementado) seria criar e consultar os eventos na versão "online" do "Google Calendar".

AbstractPreferencesInterface Interface utilizado pelo agente para armazenar ou consultar as preferências sobre marcação de eventos do utilizador. Na implementação "Android", estas são guardados na base de dados interna, "SQLite", para a qual foi necessário criar uma interface para o modelo de dados. Na versão para computador pessoal, por uma questão de simplicidade, foi utilizada a "DerbyDB", uma base de dados relacional implementada em Java que pode ser executada embebida na aplicação.

AbstractEventInterface Interface utilizado pelo agente para armazenar ou consulta eventos que ainda estão em negociação. Na implementação "Android", estes são guardados na base de dados interna do "Android", para a qual foi necessário criar também um interface para o modelo de dados. Na versão Java, é usada a base de dados "DerbyDB".

AbstractSettingsInterface Interface para os parâmetros de configuração da aplicação, assim como os dados de ligação à plataforma JADE, utilização do agente das dicas, etc. Na versão "Android" são guardados como "Settings", que são uma implementação especifica para guardar propriedades em "Android", e na versão para computador pessoal são armazenados como "Java Properties".

Os agentes, na sua inicialização, fazem apenas a instanciação das classes utilitárias referentes à plataforma onde estão a executar. Logo de seguida iniciam as suas tarefas através de Comportamentos.

3.2.2.1 Comportamentos

Um agente deve ser capaz de executar várias tarefas concorrentes em resposta aos pedidos que lhe possam chegar do exterior ou sejam atribuídos por si mesmo. De forma a poder levar a cabo essas tarefas, estas são modeladas como objetos do tipo "Behaviour" (Comportamento) e podem ser adicionadas ou removidas da lista de tarefas de um agente [Bellifemine et al.(2010)Bellifemine, Caire, Trucco, and Rimassa]. A execução dos comportamentos é totalmente gerida pelo agente e apenas cabe ao programador escolher o tipo de comportamento pretendido, implementar a sua ação e atribuir o comportamento a um agente.

Todos os agentes do sistema implementado executam na sua inicialização um comportamento chamado "WaitBehaviour". Este comportamento é o que permite que todos os agentes saibam responder a pedidos de negociação de eventos, e a sua implementação apenas aguarda pela chegada de uma mensagem ao agente. Se essa mensagem for do tipo e ontologia esperados, é então lançado um novo comportamento para dar resposta à negociação.

```
public void action() {
  MessageTemplate mt = MessageTemplate.and(
    MessageTemplate.MatchOntology(SchedulerOntology.ONTOLOGY_NAME),
    MessageTemplate.MatchPerformative(ACLMessage.CFP));

  ACLMessage msg = myAgent.receive(mt);

  if (msg != null) {
    myAgent.addBehaviour(new SchedulerResponder(...
  }
  //Fica bloqueado à espera de receber novas mensagens
  //que correspondam ao template
  block();
}
```

Implementação do método action na classe WaitBehaviour.

3.2.3 Módulo de Negociação

A negociação é o processo que tem início quando um utilizador pretende marcar um evento. Para tal, foi definido que, no mínimo, são necessários os seguintes parâmetros para iniciar uma negociação:

Titulo Titulo sumário do evento que se pretende marcar.

Janela Temporal A janela temporal é uma indicação para um período lato de tempo no qual se pretende marcar o evento. As possibilidades de preenchimento são: Hoje, Amanhã, Esta Semana, Próxima Semana, Este mês e Próximo mês.

Duração Duração estimada para o evento.

Participantes Uma lista dos participantes, devidamente classificados pelo seu grau de importância de participação. As classificações possíveis são: Opcional, Mandatório e Presidente.

Uma vez especificados estes parâmetros, o agente pode dar início ao processo de negociação, iniciando para isso um novo comportamento da seguinte forma:

```
1 if (settingsInterface.useHintAgent()) {
2   SequentialBehaviour sb = new SequentialBehaviour();
      sb.addSubBehaviour(new HintRequester(negotiator));
4     sb.addSubBehaviour(SchedulerInitiator.getInstance(this, negotiator));
    addBehaviour(sb);
6 } else
    addBehaviour(SchedulerInitiator.getInstance(this, negotiator));
```

Lançamento do comportamento de negociação.

O comportamento que dá início à negociação entre os agentes verifica se deve consultar primeiro o Agente das Dicas (linha 1). Em caso afirmativo lança uma sequência encadeada de comportamentos a executar (linhas 2 a 5). Caso contrário, lança diretamente o comportamento "SchedulerInitiator" (linha 7), cujo funcionamento e implementação são apresentados no capítulo 5.

3.3 As preferências

Para os agentes autónomos, muitas vezes o conhecimento necessário para que estes possam resolver um determinado problema pode ser representado por um conjunto de restrições. Por exemplo, um agente cuja função seja repor "stock" de materiais numa fábrica, deverá ter em conta no seu algoritmo de aquisição, as suas restrições em termos de quantidade máxima de produtos a adquirir ou montante máximo que tem disponível. Este tipo de restrições pode ser classificada da seguinte forma:

Fortes Restrições fortes [Tsang(1993)] referem-se a restrições sobre um leque de alternativas que definem quais são válidas ou não. As restrições dizem-se fortes porque não há qualquer margem de tolerância; uma alternativa é válida ou não, não há ponto intermédio.

Fracas Uma restrição fraca permite maior flexibilidade, onde se pode optar por uma restrição em detrimento da outra, ou as alternativas podem ser avaliadas em termos de custo ou preferência.

O problema que se coloca é identificar que tipos de restrições ou preferências são possíveis de definir e avaliar no processo de alinhamento de agendas. Um exemplo de preferências neste mesmo âmbito foi testado em [Franzin et al.()Franzin, Freuder, Rossi, and Wallace], onde os utilizadores tinham de indicar o nível de preferência por cidade onde o evento poderia acontecer, para cada período de tempo possível.

A opção sobre o tipo de restrição a utilizar neste projeto recaiu sobre as restrições fracas por permitirem maior liberdade e autonomia aos agentes no processo de negociação. O tipo de preferência utilizada permite aos utilizadores indicarem no seu calendário de preferências, qual o nível de preferência para cada período de tempo. Os níveis de preferência possíveis são, 0 (Indisponível), 1 (Último recurso), 2 (Aceitável), 3 (Bom), 4 (Preferível) e 5 (Melhor).

Na figura 3.5 é possível ver o interface gráfico para "Android" das preferências de um utilizador para um dia.

3.4 O agente das dicas

O agente das dicas é um agente cuja função é manter um registo do histórico de marcação de eventos entre os participantes, para com base nesta informação poder fornecer dicas ao agente organizador que acelerem o processo de negociação. Este agente pode ser consultado no início de uma negociação, pois pode, com base nos dados do evento a agendar, prever a partir de informação histórica de eventos entre estes participantes, quais são as datas mais comuns de marcação de eventos e excluir aquelas de que já tem conhecimento de estarem ocupadas. Desta lista de datas obtidas, a que estiver mais próxima do dia corrente é fornecida como dica ao agente organizador que já não precisa de consultar os participantes e pode tentar avançar logo para a marcação do evento.

Este agente tem acesso a uma base de dados "DerbyDB" onde regista todos os eventos agendados pelo sistema, incluindo a lista dos participantes. Os seus serviços de fornecimento de dicas são registados quando o agente arranca no DF da plataforma JADE, o que permite que os agentes organizadores o encontrem facilmente fazendo a pesquisa apresentada no código seguinte:

```
1  ...
   DFAgentDescription template = new DFAgentDescription();
3  ServiceDescription sd = new ServiceDescription();
   sd.setType("HintAgent");
5  sd.setName("Agenda Hint Agent");
   template.addServices(sd);
7
   //Efectuar a pesquina no DF
9  DFAgentDescription[] result = DFService.search(myAgent, template);

11 if (result.length <= 0)
     //Não encontrou agente das dicas
13 return null;

15 hintAgent = result[0].getName();
   ..
```

Pesquisa de um Agente das Dicas na plataforma.

Toda a pesquisa e interação com o agente das dicas encontra-se implementada pelo comportamento "HintRequester", cujo funcionamento é uma máquina de estados finitos com quatro estados, tal como representado na figura 3.7. No primeiro estado efetua uma pesquisa pelo agente das dicas na plataforma. Se encontrar, avança para o segundo estado onde faz o pedido ao agente. No terceiro estado, aguarda por uma resposta, enquanto o quarto e último estado é o de saída.

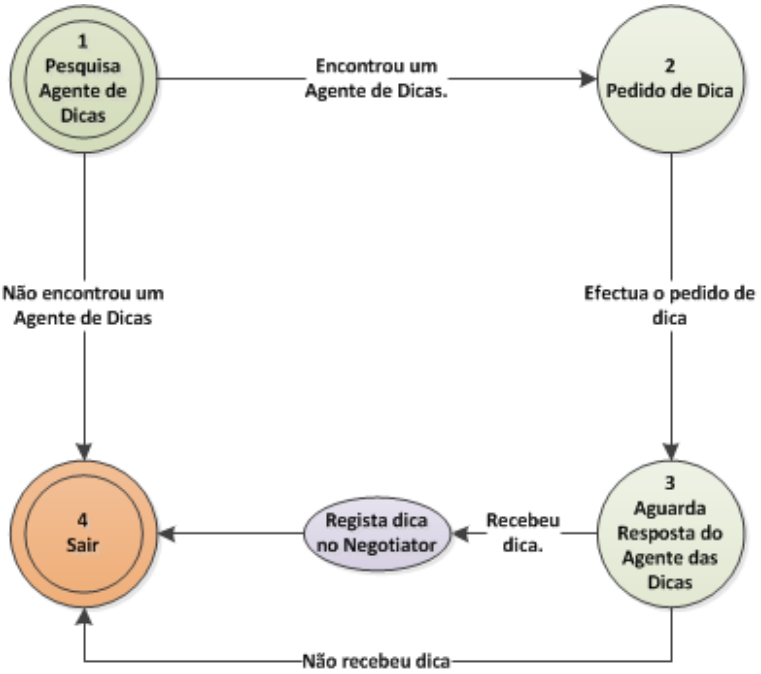

Figura 3.7: Agente das Dicas - Máquina de estados do comportamento HintRequester

No Agente das Dicas é o comportamento cíclico "HintResponder" que trata de responder aos pedidos de dicas recebidos e de armazenar em histórico os novos eventos marcados. Para sugerir uma dica, o agente consulta na sua base de dados de histórico pela existência de eventos agendados com as seguintes características:

1. A data do evento agendado não tenha passado há mais de um determinado número de dias (testado até 120 dias).
2. Os participantes sejam exatamente os mesmos do evento que se pretende marcar.
3. A data do evento agendado corresponda a um dia da semana dentro da janela temporal do novo evento.

Dos resultados obtidos desta pesquisa são agora projetados os períodos de tempo em que ocorreram, para o tempo presente. Ou seja, se for obtido um evento que ocorreu na passada terça-feira entre as 16:00 e as 17:00 horas, a projeção para o presente retorna o período correspondente à próxima terça-feira (a partir do dia corrente), entre as 16:00 e as 17:00 horas.

O período de tempo obtido que estiver mais próximo da data atual e para o qual não haja registo de nenhum dos participantes estar ocupado, é retornado como uma dica.

O Agente das Dicas voltará a ser detalhado na secção 5.2.3 dando um enquadrado sobre o seu papel no processo de negociação.

Capítulo 4

Ontologia

4.1 Ontologia

Uma ontologia pode ser vista como uma representação de um conjunto de conceitos respeitantes a um determinado domínio, e respetivas relações entre eles. Uma das mais conhecidas definições de ontologia foi dada por [Gruber(1992)]:

> "An ontology is an explicit specification of a conceptualization."

Outra definição do mesmo autor:

> "A specification of a representational vocabulary for a shared domain of discourse - definitions of classes, relations, functions, and other objects - is called an ontology."

Uma ontologia é portanto uma representação bem estruturada de um determinado domínio de conhecimento que se pretende partilhar e onde são definidos os elementos básicos (indivíduos), os conceitos (classes), atributos, funções e relações entre eles.

Indivíduos Os indivíduos são os componentes básicos de uma ontologia e podem ser desde coisas abstratas como números ou palavras, ou instanciações de conceitos concretos definidos na ontologia.

Conceitos Os conceitos (ou classes) são grupos abstratos, conjuntos ou coleções de objetos.

Atributos Os atributos são características das classes, compostas por pelo menos um nome e um valor.

Relações Geralmente, uma relação é um atributo cujo valor é outro objeto na ontologia.

4.2 Ontologia SchedulerOntology

Para que os agentes do sistema de alinhamento de agendas possam comunicar entre si, é necessário a definição de uma ontologia que abranja todos os conceitos e funções necessários para a negociação de um evento, de forma a permitir que uma mensagem recebida seja entendida de forma inequívoca por parte dos agentes.

O processo de definição da ontologia teve início num levantamento das ontologias já existentes em repositórios online, como o [W3C(2012)] ou [The GNOME Project(2012)]. Nestes repositórios é possível encontrar e reutilizar as mais diversas ontologias sobre vários domínios. No domínio da marcação de eventos ou da definição de calendários foram identificadas algumas ontologias que poderiam ser utilizadas no projeto, como por exemplo:

iCalendar O iCalendar, [Dawson et al.(1998)Dawson, Lotus, Stenerson, and Microsoft], surge como uma evolução do vCalendar e são ambos o resultado do esforço do IETF[1] na definição de um modelo para um formato de dados utilizado para comunicar informação sobre calendários e eventos entre aplicações. O iCalendar não é uma ontologia em si, mas apenas um modelo.

SCAL O SCAL[2] é uma ontologia criada pelo [The GNOME Project(2012)] baseada no modelo iCalendar, mas com várias simplificações para a tornar mais fácil de usar. Esta ontologia é utilizada para partilhar conceitos de calendários entre as aplicações da GNOME[3] e surge como uma alternativa à NCAL[4]. A NCAL é também uma ontologia criada pela GNOME, mas que é uma implementação fiel do modelo iCalendar, o que acabou por torna-la demasiado complexa de utilizar.

GDP O GDP[5] é uma definição do formato dos tipos de dados que podem ser trocados online entre as aplicações da [Google(2011)], entre as quais, o Google Calendar.

As ontologias NCAL, SCAL ou iCal contêm todos os conceitos identificados como necessários para a implementação, mas estas implementações continuam a ser demasiado complexas e detalhadas para o que se pretende utilizar. Desta forma, optou-se por simplificar uma ontologia existente de modo a obter apenas os conceitos, funções e relações estritamente necessárias. No entanto, os conceitos da ontologia tem como base conceitos e relações definidos pelo GDP, o que, a) nos permite reutilizar conceitos de eventos já definidos e aceites pela comunidade, b) facilita a integração da solução com o próprio Google Calendar.

4.2.1 Implementação

Uma vez identificada a ontologia que iria servir de base à implementação, foi necessário escolher também uma ferramenta e metodologia que permitisse desenhar e utilizar facilmente a ontologia na plataforma JADE.

[1] Internet Engineering Task Force
[2] Simplified Calendar Ontology
[3] GNU Network Object Model Environment
[4] Nepomuk Calendar Ontology
[5] Google Data Protocol

O JADE permite que se transporte nas mensagens trocadas entre os agentes um conteúdo codificado numa determinada ontologia, fornecendo métodos que permitem validar, pesquisar, codificar e descodificar estes conteúdos nas mensagens. Para tal, é necessário ter uma definição implementada em Java dos objetos que compõem o domínio, objetos como Termos, Predicados, Conceitos ou Ações.

Um exemplo da implementação de uma ontologia utilizando esta técnica:

```
import jade.content.onto.*;
import jade.content.schema.*;
public class SchedulerOntology extends Ontology {
  // O nome da ontologia
  public static final String ONTOLOGY_NAME = "Scheduler-ontology";
  // Vocabulário
  public static final String EVENT = "Event";
  public static final String WHEN = "When";

  private SchedulerOntology() {
    // A AgendaOntology deriva da classe base Ontology
    super(ONTOLOGY_NAME, BasicOntology.getInstance())
      try {
        add(new ConceptSchema(EVENT), Event.class);
        add(new ConceptSchema(WHEN), When.class);

        // Construção do schema para o conceito de Evento
        ConceptSchema csEvent = (ConceptSchema) getSchema(EVENT);
        // Um evento, tem um When
        cs.add(WHEN, (ConceptSchema) getSchema(WHEN));
      } catch (OntologyException oe) {
        oe.printStackTrace();
      }
   }
}
```

Ontologia - Exemplo parcial da definição de uma ontologia criando manualmente os schemas e os conceitos

No exemplo acima temos a criação manual de uma ontologia onde são declarados os nomes dos conceitos suportados (linhas 7 e 8), criação dos esquemas ("schemas") que contêm a definição dos conceitos (linhas 14 e 15), e finalmente a definição da relação entre entre os conceitos (linha 20), onde ao conceito de "EVENT" é associada uma instância de "WHEN".

A definição de uma ontologia é simplificada usando o "BeanOntology" que é uma extensão da classe "Ontology" da plataforma JADE para permitir a criação automática dos esquemas da ontologia com base apenas nas classes que implementam os Conceitos e Ações, desde que estes respeitem o modelo dos "JavaBeans". Tendo em vista a utilização do "BeanOntology", foi utilizada a ferramenta "OntologyBeanGenerator" em [Stanford Proteje WIKI(2008)], que permite desenhar e validar a ontologia na ferramenta "Protégé", e com base nessa definição criada, exporta-la para classes Java. [develop123(2010)].

Na figura 4.1 é possível ver a definição do conceito de evento na ferramenta "Protégé". A tabela 4.1 mostra todos os conceitos criados na ontologia para responder as necessidades de definições.

Além dos conceitos, foram também definidas as ações que explicitam a função da mensagem

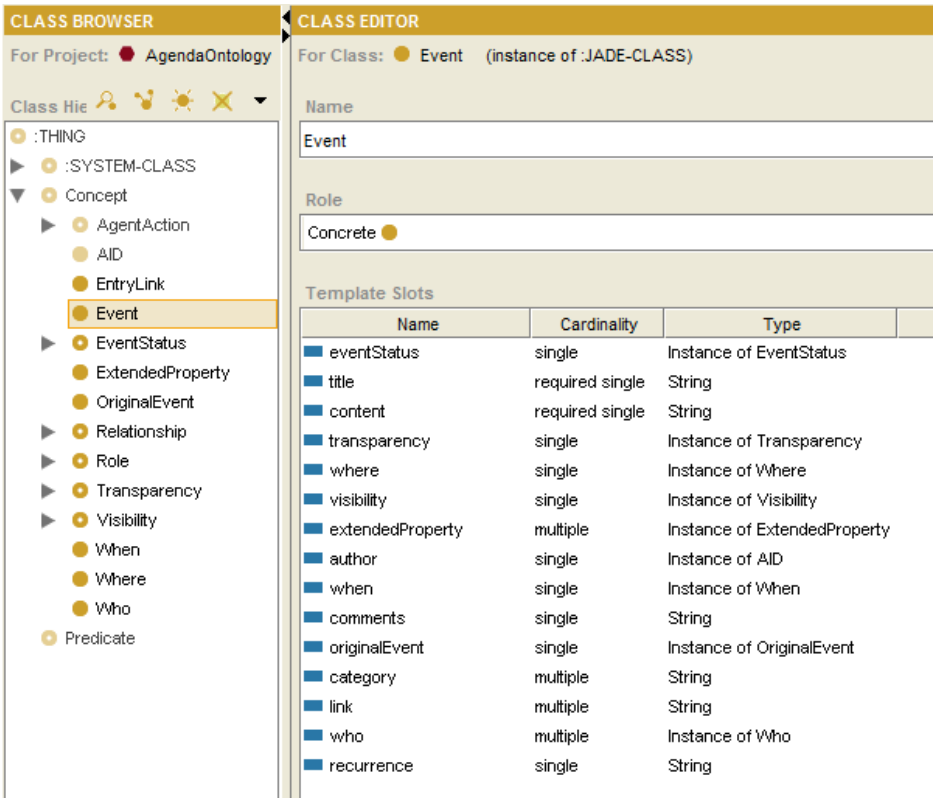

Figura 4.1: Detalhe da definição do conceito de evento na ontologia.

enviada ou respondida. Estas ações estão representadas na tabela 4.2.

Conceito	Descrição
AID	Agent Identifier - Conceito que identifica um agente dentro da plataforma.
Event	Conceito que representa um evento com as suas características.
EventStatus	Status de um evento - É um conceito abstrato com três possíveis implementações: Canceled;Confirmed;Tentative.
Extendend Property	É uma propriedade genérica que pode ser utilizada para colocar informação adicional nos conceitos que a contêm.
OriginalEvent	Conceito que existe como uma característica de um evento e que permite associar eventos entre si.
Role	Conceito abstrato utilizado para representar o papel de cada um dos participantes num evento. Contêm as seguintes implementações: Chair; RequiredParticipant; OptionalParticipant
Visibility	Conceito abstrato que define o grau de visibilidade do evento para terceiros. Tem as seguintes implementações: Confidential; Private; Default; Public
When	Define a janela temporal em que ocorre o evento.
Where	Define o local onde o evento terá lugar.
Who	Define o conceito de um participante no evento, assim como o seu papel.

Tabela 4.1: Ontologia SchedulerOntology - Conceitos

Ação	Descrição
ActionHintRequest	Pedido enviado pelo anfitrião ao agente das dicas, com um pedido de sugestões para o evento.
ActionHintPropose	Resposta do agente das dicas com as sugestões para o evento.
ActionHintNoHintInform	Resposta do agente das dicas com a indicação que não existem sugestões.
ActionHintEventInform	Mensagem enviada pelo agente coordenador para o Agenda Das Dicas, a informar o agendamento de um evento.
ActionAvailabilityCFP	Pedido de disponibilidades para um evento enviado aos participantes.
ActionAvailabilityPropose	Resposta dos participantes com as propostas de disponibilidades para o evento.
ActionAvailabilityRefuse	Rejeição por parte de um participante em fornecer as suas disponibilidades.
ActionAvailabilityRejectProposal	Rejeição por parte do anfitrião de uma proposta de disponibilidade de um participante.
ActionVoteCFP	Pedido de votação para a marcação de um evento.
ActionVotePropose	Resposta dos participantes com a votação para a marcação do evento.
ActionVoteRefuse	Rejeição por parte de um participante em votar.
ActionVoteRejectProposal	Rejeição por parte do anfitrião de uma votação.
ActionEventCFP	Pedido de aceitação de um evento enviado aos participantes.
ActionEventPropose	Proposta de aceitação de um evento por parte de um participante.
ActionEventRefuse	Rejeição por parte de um participante em aceitar o evento.
ActionEventRejectProposal	Rejeição por parte do anfitrião da aceitação de um evento por parte de um participante.
ActionEventAcceptProposal	Confirmação da aceitação geral do evento enviada pelo anfitrião aos participantes.
ActionEventInformDone	Confirmação da marcação do evento na agenda do participante.
ActionEventInformFailure	Informação de erro na marcação do evento na agenda do participante.

Tabela 4.2: Ontologia SchedulerOntology - Ações

Capítulo 5

Negociação

5.1 Tomada de decisão

Durante o processo de negociação, chegará a um ponto em que os agentes terão de tomar decisões de forma a cumprirem os seus objetivos: escolher em conjunto, um período de tempo para a realização de um evento que respeite e maximize as suas preferências. O trabalho de [Conitzer(2010b)] faz uma análise sobre várias abordagens possíveis para a tomada de decisões num sistema multi-agente. Destas abordagem, destacam-se essencialmente as seguintes:

- Votação - Dado um conjunto de alternativas válidas, os agentes apenas têm de votar. Ao fazê-lo, mais do que simplesmente votar, estão a classificar as alternativas de acordo com as suas preferências e a ordená-las. Do conjunto de votações efetuadas pelos agentes é possível então escolher um vencedor.
- Alocação de recursos e tarefas - Nesta situação existe um conjunto de tarefas ou recursos a alocar, e o desafio está em distribui-los de forma eficiente pelos agentes. O algoritmo para decidir a que agentes devem ser alocados os recursos ou tarefas, deve decidir de acordo com um objetivo bem definido, pois o conceito de "eficiente" pode ser ambíguo. Por exemplo, uma decisão pode ser eficiente em termos de custos, mas não o ser em termos de tempo.
- Leilão - Num leilão assume-me que um determinado agente possui um ou mais recursos, e que existem outros agentes interessados nesse recurso, os quais os podem licitar. A licitação normalmente implica que se atribua um valor ao recurso.

Considerando as necessidades de negociação dos agentes para um sistema de alinhamento de agentes, temos como objetivo a escolha de um período de tempo que respeite as restrições e maximize o nível de preferência de todos os participantes. A escolha do período é feita sobre um conjunto de períodos admissíveis e todos os participantes devem concordar com a escolha.

A alocação de recursos e tarefas não é aplicável neste contexto, pois o que se pretende não é distribuir os períodos de tempo admissíveis pelos participantes, mas sim que estes escolham um dos períodos.

António	Ana	José
Presidente (3)	Obrigatório (2)	Opcional (1)
Segunda	Terça	Quarta
Quarta	Quarta	Terça
Terça	Segunda	Segunda

Tabela 5.1: Votação - Exemplo de votação com preferências ponderadas de acordo com a importância

O leilão também não é a abordagem mais adequada para esta situação pois tem implícita uma alocação de recursos (o vencedor do leilão ganha o recurso) e há normalmente um preço a pagar por este. No entanto, no caso particular do leilão combinatório, onde os agentes podem licitar não apenas um dos recursos mas uma combinação destes atribuindo-lhes um valor, esta valorização pode ser utilizada como uma representação das preferências dos agentes sobre o conjunto de períodos. Neste artigo, [Conitzer(2010a)] faz uma comparação entre a abordagem do leilão combinatório e a votação.

Na votação, os participantes de um evento fazem o papel de eleitores no qual devem votar no conjunto de períodos possíveis, de acordo com as suas preferências.

5.1.1 Votação

Para compreender melhor o processo de votação, vamos começar com um exemplo semelhante ao utilizado num artigo por [F. Brandt and Endriss(2012)], mas adaptado ao tema do alinhamento de agendas:

Consideremos uma situação na qual três participantes pretendem marcar uma reunião entre si, nomeadamente a Ana, que deverá participar obrigatoriamente, o José, cuja participação é opcional, e o António, que é o Presidente. Após consultarem as suas disponibilidades para a reunião, os dias disponíveis em comum são Segunda, Terça e Quarta-feira, sendo que cada um deles definiu a seguinte ordem de preferência:

- António - Primeiro Segunda, depois Quarta e por último Terça-feira.
- Ana - Primeiro Terça, depois Quarta e por último Segunda-feira.
- José - Primeiro Quarta, depois Terça e por último Segunda-feira.

Vamos considerar que na votação, o nível de importância do participante é um fator ponderador do seu voto, assumindo como fator multiplicador 1 para os participantes opcionais, 2 para os obrigatórios e 3 para o presidente.

Estas preferências e ponderações estão representadas na tabela 5.1.

Com base nesta informação, qual vai ser o dia escolhido? Depende da regra utilizada. Se considerarmos a ordem de preferência como um "ranking" onde a alternativa com menor preferência tem o valor 1, a seguinte tem o valor 2, etc, a soma dos pontos dá a vitória à Quarta-feira. Por sua vez, se considerarmos apenas o número de vezes que uma alternativa aparece em primeiro lugar, a escolha vencedora é a Segunda-feira.

Mêtodo	Segunda	Terça	Quarta
Borda	12	11	13
Pluralidade	3	2	1
Anti-pluralidade	-3	-3	0

Tabela 5.2: Votação - Resultados da contagem de votos utilizando os métodos Contagem de Borda, Pluralidade e Anti-Pluralidade

Descrevemos agora algumas das regras de votação mais conhecidas e utilizadas.

Contagem de Borda ("Borda Count") Na Contagem de Borda, as alternativas são todas votadas por ordem de preferência formando um "ranking", onde a última alternativa é valorada em 1 ponto, a penúltima é valorada com 2 pontos, e assim sucessivamente até valorar a primeira com a pontuação máxima. A alternativa vencedora é aquela que obtiver a maior soma de pontos de todas as votações.

Regra de pluralidade ("Plurality rule") A Regra da Pluralidade considera para a escolha do vencedor apenas o número de vezes que uma alternativa aparece em primeiro lugar. [Baharad and Nitzan(2005)]

Regra de anti-pruralidade ("Anti-plurality rule") Ao contrário da regra anterior, na regra da anti-pluralidade ganha a alternativa que ficar menos vezes em último lugar. [Baharad and Nitzan(2005)]

STV ("Single Transferable Vote") - Esta regra necessita de várias rondas de votação para encontrar um ou mais vencedores. Na primeiro ronda, o processo é o mesmo da Contagem de Borda, todas as alternativas são votadas formando um "ranking" de preferências. A cada ronda é eliminada a alternativa que tiver menos pontos e iniciada uma nova ronda com as restantes, até sobrarem o número de vencedores pretendido. Com esta regra, os "eleitores" que votaram previamente na alternativa eliminada são obrigados a recalcular as suas preferências e possivelmente a que era uma segunda escolha, passou agora para primeira. O objetivo desta regra é minimizar votos desperdiçados em alternativas sem hipótese de vencer, obrigando todos a votar de forma mais proporcional nas "verdadeiras" alternativas. [Aseere et al.(2010)Aseere, Gerding, and Millard]

Todas estas regras são passíveis de utilizar no projeto, no entanto, a regra STV tem a particularidade de necessitar de várias rondas para conseguir obter um vencedor, pelo que foi descartada por consumir demasiados recursos que devemos ter em conta quando se trata de comunicação e processamento em dispositivos móveis. Na implementação são suportadas estas três regras: Contagem de Borda, Regra da Pluralidade e Regra da Anti-Pluralidade, ficando ao critério do agente anfitrião qual a regra a utilizar no processo de negociação.

Na tabela 5.2 estão os resultadas da contagem dos votos para os três algoritmos suportados pela aplicação, Contagem de Borda, Pluralidade e Anti-Pluralidade.

5.2 Comunicação entre agentes

Identificada a abordagem da votação como mecanismo para encontrar um vencedor, é necessário definir a forma como os agentes vão comunicar entre si. Uma conversação entre agentes tende para certos padrões de sequências de mensagens enviadas e respondidas. A estes padrões típicos dá-se o nome de Protocolos de Interação.

5.2.1 Protocolos de Interação

Na implementação de um sistema multi-agente, os agentes são programados para responderem aos mais variados tipos de mensagens que se esperam trocar durante uma conversação. Numa implementação "ad-hoc" existe total abertura para as mensagens trocadas, o que a torna difícil de implementar e fácil de perder o controlo. É por esta razão que existem os Protocolo de Interação, os quais definem alguns padrões de conversação típicos e especificam inequivocamente a sequência de mensagens possíveis durante toda uma conversação.

A FIPA[1] especifica protocolos que podem ser utilizados e adaptados para responder a uma determinada necessidade de comunicação, entre os quais se destacam:

FIPA Request Interaction Protocol Este protocolo permite que um agente peça a outro para executar uma determinada tarefa. Em resposta, o segundo agente pode concordar ou discordar. Se concordar, informa o primeiro agente do resultado da execução da tarefa.

FIPA Query Interaction Protocol Este protocolo é utilizado caso um agente pretenda questionar outro sobre a validade ou existência de algo.

FIPA Request When Interaction protocol Protocolo idêntico ao Request, mas no qual o segundo agente apenas executa a tarefa após uma determinada condição se verificar.

FIPA Contract Net Interaction Protocol Este protocolo é iniciado por um agente que pede aos outros uma proposta sobre um determinado tema, os quais podem rejeitar, se não tiverem propostas, ou em caso afirmativo, responder com as condições da proposta. O primeiro agente pode então aceitar ou não as propostas, informado os outro agentes da sua decisão. Se não aceitar, a comunicação termina, se aceitar, os outros agentes devem então responder ao primeiro o resultado da execução da proposta.

FIPA Iterated Contract Net Interaction Protocol Este protocolo é uma variante do anterior, onde o primeiro agente pode reiniciar o protocolo, enviando um novo pedido de propostas após receber as respostas dos outros agentes. A figura 5.1 mostra o diagrama de sequência para este protocolo.

No processo de negociação para o alinhamento de agendas foi utilizado como base o protocolo de interação Iterated Contract-Net, para o qual é fornecida uma implementação de referência pela plataforma JADE através da extensão das classes "ContractNetInitiator" e "SSIterated-ContractNetResponder". Ambas as classes são implementadas como comportamentos; a

[1] Foundation for Intelligent Physical Agents

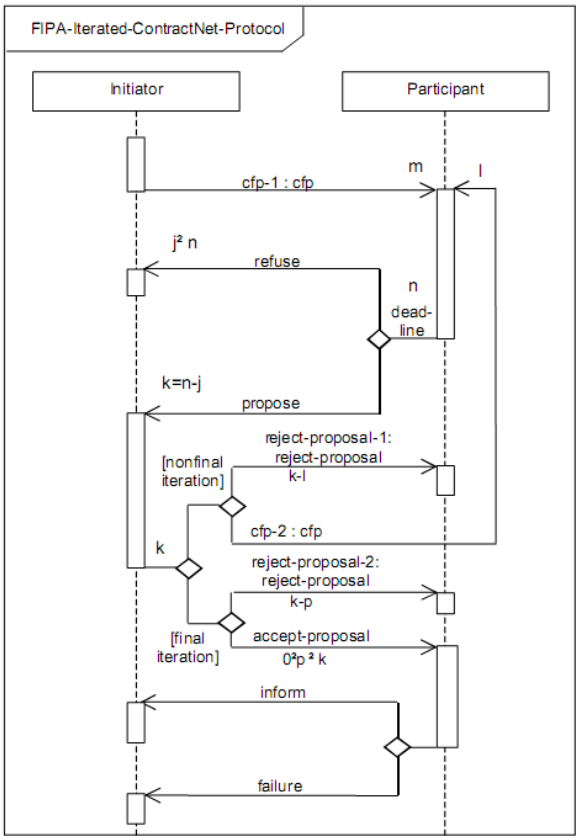

Figura 5.1: Negociação - Protocolo de Interação Iterated Contract-Net

primeira é o comportamento implementado pelo agente que pretende iniciar a conversação, neste caso, o agente responsável pela marcação do evento e negociação com os restantes, a quem chamaremos a partir de agora de coordenador. A segunda classe implementa o comportamento dos agentes com quem o coordenador está a negociar, que no nosso caso são os participantes no evento.

5.2.2 Scheduler Interaction Protocol

O Scheduler Iteration Protocol é uma extensão ao Iterated Contract-Net para a implementação do protocolo de interação, através do qual os nossos agentes trocam informação sobre disponibilidades, votam nos períodos propostos e aceitam ou não a marcação de um evento. A implementação deste protocolo divide-se em três componentes:

SchedulerInitiator É uma extensão da classe "ContractNetInitiator" que implementa o comportamento correspondente ao agente coordenador.

SchedulerResponder É uma extensão da classe "SSIteratedContractNetResponder" que implementa o comportamento dos agentes participantes.

Negotiator O Negotiator é uma classe que implementa um conjunto de métodos auxiliares para o processo de negociação e que são partilhados por ambos os comportamentos

"SchedulerInitiator" e "SchedulerResponder".

O processo de negociação tem início quando o agente coordenador recebe o pedido de marcação de um evento. Ao receber o pedido, o agente adiciona o comportamento "SchedulerInitiator" à sua lista de tarefas, dando início ao protocolo de interação, o qual é composto por três fases, nomeadamente, a) o pedido das disponibilidades aos participantes para a realização do evento, b) o processo de votação aos períodos propostos pelo coordenador e c) a aceitação e marcação do evento nas agendas.

5.2.2.1 Pedido de disponibilidades

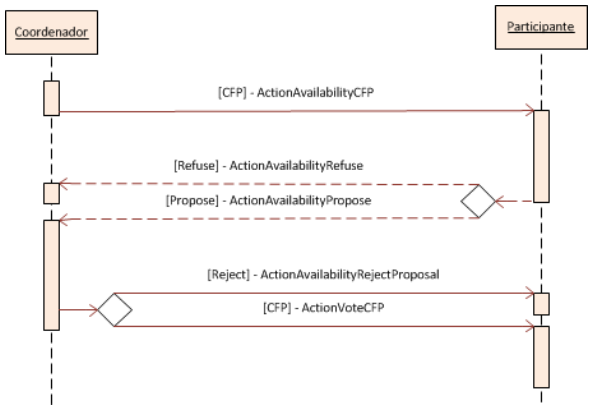

Figura 5.2: Scheduler Interaction Protocol - Pedido de disponibilidades

Na primeira fase do protocolo, o coordenador envia para todos os participantes um pedido de propostas das suas disponibilidades e preferências para participar no evento, para o qual apenas lhes envia a janela temporal onde se pretende realizar e a duração prevista. Os participantes, ao receberem o pedido de proposta, dão início ao comportamento "SchedulerResponder", o qual vai calcular as suas disponibilidades da seguinte forma:

1. Calcula os dias possíveis para a realização do evento de acordo com a janela temporal. Por exemplo, se a janela temporal for "Esta Semana", os dias possíveis são todos os dias úteis entre o dia corrente e sexta-feira dessa mesma semana.

2. Para cada um desses dias possíveis, obtêm as preferências definidas pelo utilizador. Para cada período de preferência obtido, se este estiver acima de um determinado grau de preferência, considera esse período como elegível para a marcação.

3. Dos períodos elegíveis resultantes do ponto anterior, são removidos da lista todos aqueles que colidam com eventos já agendados no calendário do utilizador.

Esta primeira fase pode terminar de duas formas distintas. Se do processo anterior não resultou nenhum período admissível, significa que este agente não vai poder participar no evento por incompatibilidade de horário. Nesse caso, é retornada a mensagem "ActionAvailabilityRefuse" com essa indicação. Caso exista um, ou mais, períodos admissíveis, é retornada a

mensagens "ActionAvailabilityPropose" que contêm a proposta (a lista de períodos admissíveis com a devida classificação de preferência). A figura 5.2 representa o diagrama de sequência desta primeira fase do protocolo.

5.2.2.2 Votação

Ao receber todas as respostas, o agente coordenador, ou mais especificamente, o seu comportamento "SchedulerInitiator" vai ter agora de processar os períodos recebidos de forma a ficar apenas com os períodos coincidentes entre todas os participantes.

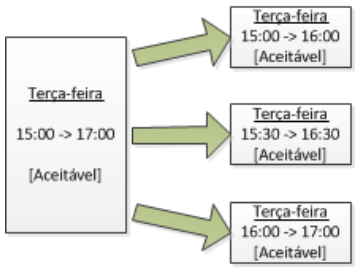

Figura 5.3: Scheduler Interaction Protocol - Exemplo da divisão de um período para uma reunião de uma hora.

Para conseguir obter apenas os períodos coincidentes, os períodos recebidos vão ser divididos em frações mais pequenas de duração igual à do evento e em intervalos de 30 minutos, como exemplificado na figura 5.3. Desta forma estamos a reduzir todos os períodos à mesma duração, tornando assim possível a sua comparação direta. Desta forma, os períodos coincidentes podem agora ser obtidos calculando a interceção dos conjuntos.

Se existirem períodos coincidentes entre todos os participantes, estes são novamente classificados tendo em conta as preferências iniciais reportadas pelos participantes, ponderadas pelo grau de importância de participação que lhe foi atribuído. Do resultado desta classificação são escolhidos os períodos com melhor classificação (no máximo 10), os quais são enviados aos participantes para votação. Se não existirem períodos coincidentes entre todos, é calculada novamente a interceção do conjunto de períodos, excluindo os sugeridos pelos participantes que foram classificados como opcionais. Se a exclusão destes elementos não resultar, significa que não vai ser possível agendar o evento e termina a negociação.

Voltando aos participantes, estes receberam agora o pedido de votação, com a lista de períodos sugeridos pelo coordenador, os quais vão ter de reclassificar e devolver ordenados por nível de preferência. A figura 5.4 representa o diagrama de sequência das mensagens trocadas nesta fase.

Ao receber todas as respostas, o coordenador procede então à contagem dos votos recebidos de acordo com a regra definida nos parâmetros de configuração, nomeadamente Contagem de Borda, Regra da Pluralidade ou Regra da Anti-Pluralidade. Desta contagem podem resultar um ou mais períodos vencedores. No caso de haver vários vencedores é escolhido o período mais próximo da data corrente, dando início à terceira e última fase do protocolo.

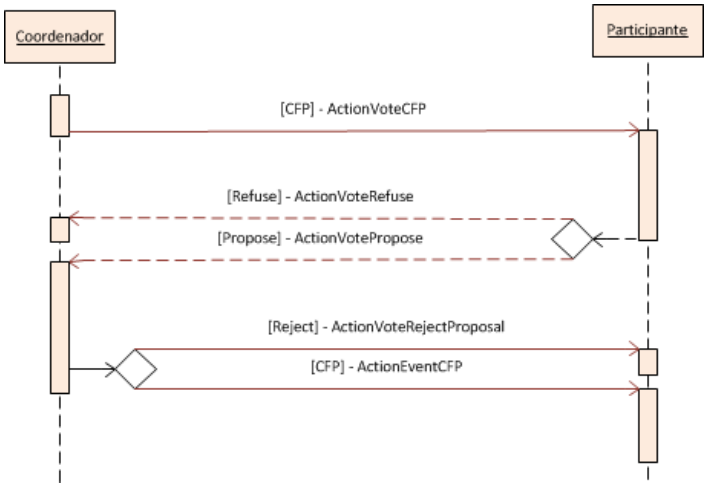

Figura 5.4: Scheduler Interaction Protocol - Votação

5.2.2.3 Aceitação e Marcação do evento

Obtido o período de tempo vencedor, o coordenador deve proceder agora à sua divulgação aos participantes, através da mensagem "ActionEventCFP". Esta mensagem serve para que todos os participantes confirmem a sua aceitação ao evento e reservem o período nos seus calendários. Nesta fase, os participantes procedem a uma última consulta às suas agendas de forma a garantir que continuam disponíveis e que nenhum outro evento foi agendado no mesmo período. Se assim for, é efetuado o registo do evento no calendário e marcado como "Tentative" (Opcional), o que serve como reserva temporária até à confirmação final. Se todos os participantes responderem afirmativamente, o coordenador envia a mensagem "ActionEventAcceptProposal", o que significa que os participantes podem finalmente marcar o evento no calendário como definitivo e informar o coordenador quando estiver feito através da mensagem "ActionEventInformDone", dando por terminada a negociação com sucesso. A figura 5.5 representa o diagrama de sequência das mensagens trocadas nesta fase.

Nesta última fase, é possível abortar a negociação em algumas situações, nomeadamente:

1. Caso um participante não concorde ou não tenha disponibilidade no período definido para o evento, deve responder com a mensagem "ActionEventRefuse". Se o grau de importância do participante for diferente de "Opcional", então todo o processo de negociação é cancelado.

2. Se algum dos participantes obrigatórios responder com "ActionEventRefuse", ou se o agente coordenador não tiver disponibilidade, é enviada a mensagem "ActionEventRejectProposal" para todos os participantes com a indicação que a negociação foi cancelada. Os participantes devem então remover dos seus calendários o evento correspondente.

3. Se todos os participantes confirmarem a aceitação ao evento e for enviada pelo coordenador a mensagem "ActionEventAcceptProposal", todos o participantes devem marcar o evento nas suas agendas como definitivo. Se algum deles tiver problemas a efetuar este

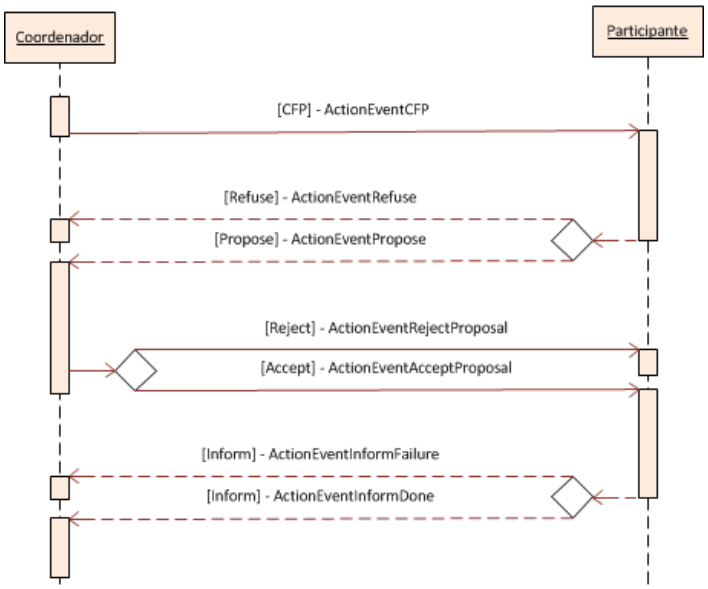

Figura 5.5: Scheduler Interaction Protocol - Aceitação e marcação do evento na agenda

registo, deve responder com a mensagem "ActionEventInformFailure". Na implementação, esta situação não cancela o agendamento do evento, pois assume-se que o agente ao já ter aceite previamente o evento, este já se encontra registado na sua agenda, mesmo que esteja como opcional.

5.2.3 Agente das dicas

O papel do agente das dicas é fornecer ajudas com base em informação previamente armazenada sobre eventos marcados no sistema, que possam ajudar a acelerar o processo de negociação. O diagrama de sequência das mensagens utilizadas na comunicação com este agente estão representadas na figura 5.6.

Desta forma, o agente coordenador, antes de dar início ao comportamento "SchedulerInitiator", lança previamente o comportamento "HintRequester". Se este encontrar um Agente das Dicas, e o agente retornar uma dica para o agendamento do evento, o comportamento "SchedulerInitiator" vai ser lançado mas não serão executadas as fases de obtenção de disponibilidades nem de votação, passando logo para a fase de aceitação do evento, considerando como período o sugerido pelo agente das dicas. Se a dica for boa, os participantes aceitam o evento, e este é marcado "poupando" todas as mensagens e processamento necessário para as duas primeiras fases da negociação. Se a dica não funcionar e algum dos participantes rejeitar o evento, o processo é reiniciado normalmente, com recurso à negociação.

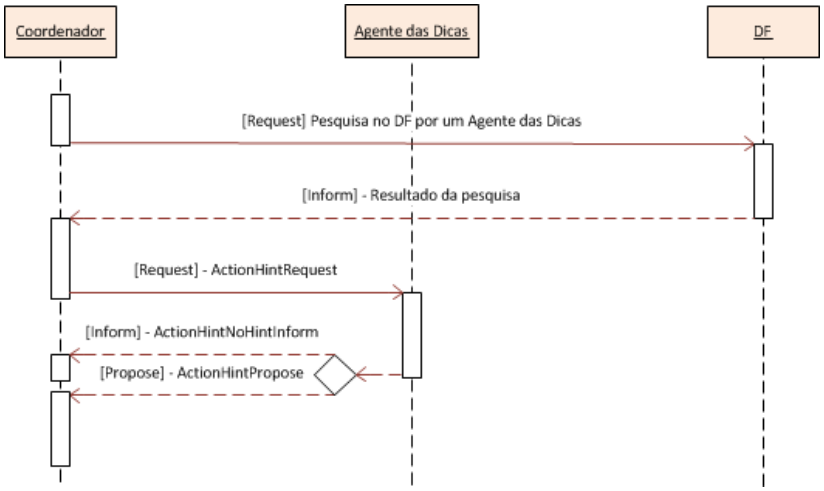

Figura 5.6: HintAgent Interaction Protocol - Protocolo de comunicação com o Agente das Dicas

Capítulo 6

Testes

6.1 Testes Realizados

Os testes foram realizados utilizando até oito agentes como participantes a serem executados na plataforma JADE. Estes agentes obtém as suas preferências e eventos agendados através de um ficheiro XML especificado como parâmetro. Foi utilizado também o agente com a implementação do interface gráfico em Java a executar num computador pessoal, e claro, uma instância do agente "Android".

6.1.1 Teste de Carga

O teste de carga tem como objetivo testar a robustez e eficiência dos agentes e do sistema quando sujeitos a uma maior carga.

No primeiro teste efetuado, não foi incluído o agente "Android", apenas os agentes a executar no mesmo computador pessoal e na mesma rede da plataforma JADE, cujos resultados esperados deveriam ser os mais rápidos dos testes. Como se pode ver na tabela 6.1, o teste consistiu em aumentar a carga, incrementando a janela temporal e o número de participantes progressivamente, utilizando sempre a mesma ocupação e preferências.

Janela Temp.	Convidados	Tempo (s)	Mensagens	Encontrou solução?
Hoje	1	1	8	Sim
Hoje	8	1	64	Sim
Próxima Semana	1	1	8	Sim
Próxima Semana	8	1	64	Sim
Próximo Mês	1	1	8	Sim
Próximo Mês	8	2	64	Sim

Tabela 6.1: Teste de carga - Tempo, volume mensagens e eficácia, utilizando apenas agentes a executar em computadores pessoais

No segundo teste, foi incluído o agente "Android" apenas como participante, repetindo-se a sequência de testes anterior, cujos resultados estão na tabela 6.2. O agente a executar através

do dispositivo móvel estava ligado através de uma rede WIFI.

Janela Temp.	Convidados	Tempo (s)	Mensagens	Encontrou solução?
Hoje	1	1	8	Sim
Hoje	8	2	64	Sim
Próxima Semana	1	1	8	Sim
Próxima Semana	8	2	64	Sim
Próximo Mês	1	1	8	Sim
Próximo Mês	8	3	64	Sim

Tabela 6.2: Teste de carga - Tempo, volume mensagens e eficácia, utilizando agentes a executar em computadores pessoais e um a correr num dispositivo móvel, ligado por WIFI

No terceiro teste de carga, o agente "Android" estava ligado à rede através de uma ligação móvel 3G, mas continuando com o papel de participante. Os resultados do teste estão na tabela 6.3.

Janela Temp.	Convidados	Tempo (s)	Mensagens	Encontrou solução?
Hoje	1	2	8	Sim
Hoje	8	2	64	Sim
Próximo Mês	1	3	8	Sim
Próximo Mês	8	4	64	Sim

Tabela 6.3: Teste de carga - Tempo, volume mensagens e eficácia, utilizando agentes a executar em computadores pessoais e um a executar num dispositivo móvel, ligado por rede 3G (via Internet)

Por último, o agente "Android" foi testado agora com o papel de coordenador e ligado através da rede móvel 3G. Os resultados estão na tabela 6.4.

Janela Temp.	Convidados	Tempo (s)	Mensagens	Encontrou solução?
Hoje	1	4	8	Sim
Hoje	4	8	32	Sim
Hoje	8	9	64	Sim
Próximo Mês	1	4	8	Sim
Próximo Mês	4	7	32	Sim
Próximo Mês	8	9	64	Sim

Tabela 6.4: Teste de carga - Tempo, volume mensagens e eficácia, utilizando agentes a executar em computadores pessoais e um a correr num dispositivo móvel, ligado por rede 3G (via Internet), atuando como coordenador

6.1.2 Teste de Concorrência

Este teste avalia o comportamento dos agentes quando confrontados com situações de concorrência. O caso de testes consistiu em utilizar dois coordenadores a negociar um evento cada um, em simultâneo com o mesmo participante. Numa primeira fase para testar apenas o processamento de mensagens em simultâneo por parte do participante, os eventos utilizados tinham janelas temporais dispares, de forma a garantir que os períodos vencedores não

colidiam. Numa segunda fase, utilizamos a mesma janela temporal de forma a forçar a colisão dos períodos vencedores.

6.1.3 Teste de eficiência do agente das dicas

Esta teste avalia a eficiência da utilização de um Agente de Dicas em termos de impacto no tempo e número de mensagens trocadas entre os agentes. Foram testadas situações de negociação com e sem Agente das Dicas, com e sem dica fornecida pelo agente, e ainda o impacto do fornecimento de uma dica errada (um período não admissível para um dos participantes). Os resultados do teste encontram-se na tabela 6.5.

Utiliza Agente Dicas?	Tem Dica?	Participantes	Tempo (s)	Mensagens
Sim	Sim	1	1	8
Sim	Não	1	1	12
Sim	Sim(não admissível)	1	1	14
Não	-	1	1	8
Sim	Sim	8	1	36
Sim	Não	8	2	68
Sim	Sim(não admissível)	8	2	84
Não	-	8	2	64

Tabela 6.5: Teste de eficiência da negociação utilizando agente das dicas

6.2 Resultado dos Testes

Nos testes de carga efetuados é possível concluir que o incrementar da janela temporal não provoca uma degradação significativa no tempo de processamento dos agentes. O incremento do número de participantes também não tem um impacto negativo no tempo de processamento, mas sim no tempo consumido em comunicações. Nos testes realizados nas redes mais rápidas, entre os agentes da plataforma ou mesmo com o agente "Android" utilizando WIFI, o tempo total de negociação não é muito afetado, mas é notável a degradação nas comunicações quando se utiliza a rede móvel 3G, principalmente devido à latência da rede.

Nos testes de concorrência efetuados não foram detetados problemas nem degradação significativa no desempenho dos agentes, mesmo quanto efetuando negociações em paralelo. A coordenação e sequenciação das mensagens é garantida pelo protocolo de interação, que marca cada uma das conversões com um "Conversation-id" e sincroniza as resposta com recurso ao "Reply-with" ou "In-reply-to", ambos propriedades intrínsecas das mensagens ACL. Na segunda fase do teste dois agentes coordenadores negoceiam eventos com o mesmo participante para a mesma janela temporal, dando lugar a duas negociações paralelas, nomeadamente a' e b'. Foram identificadas duas situações distintas de concorrência:

1. A negociação a' marcou o evento no calendário antes da negociação b' consultar as disponibilidades.
2. A negociação a' marcou o evento no calendário depois da negociação b' consultar as disponibilidades e antes de as confirmar novamente na fase de aceitação do evento.

Na primeira situação, o agente ao consultar as disponibilidades para a negociação b', identifica o período como ocupado e a negociação prossegue com os restantes períodos admissíveis. Na segunda situação, como o período de tempo em causa ainda não estava reservado no calendário, este também é proposto como período admissível na negociação b'. Na fase de aceitação de b', o agente verifica novamente as disponibilidade do período e identifica que este já está ocupado, enviado a mensagem "ActionEventRefuse" para o coordenador, o que causa o cancelamento de toda a negociação caso a importância do participante seja diferente de "Opcional". Na fase de aceitação do evento por parte dos participantes, o evento é registado no calendário como "Tentative" (Provisório), funcionando como uma pré-reserva, o que impede que um período seja ocupado por outra negociação entre a fase de aceitação e marcação.

No teste efetuado à eficiência da utilização de um Agente de Dicas no processo de negociação verificou-se que este trás grandes vantagens nas situações positivas (em que fornece dicas admissíveis), reduzindo consideravelmente o número de mensagens trocadas entre o coordenador e os participantes. Nesta situação, como a negociação não passa pelas fases de pedido de disponibilidade e votação, são poupadas quatro mensagens por participante e consumidas apenas quatro pela pesquisa e interação com o Agente das Dicas. Na situação negativa (em que são fornecidas dicas não admissíveis) são gastas as quatro mensagens de pesquisa e interação com o Agente das Dicas, mais duas mensagens por participante no processo de aceitação. Como nesta situação um dos participantes rejeita a proposta, é necessário reiniciar o processo de negociação normal, desta vez sem a utilização das dicas.

6.3 Conclusões dos testes

Os testes de carga efetuados revelam que o incremento da janela temporal não têm um impacto significativo no desempenho do sistema, particularmente no tempo de processamento dos agentes. Por sua vez, com o incremento do número de mensagens, resultado do incremento dos participantes, verifica-se uma deterioração na performance especialmente na utilização de redes mais lentas. O tipo de rede utilizado pelo agente coordenador é o que têm maior impacto, pois é com este agente que são trocadas todas as mensagens numa negociação.

Nos testes de concorrência verifica-se que os agentes suportam a negociação paralela de vários eventos, mesmo na situação em que os períodos sujeitos a aceitação colidem, não ocorrendo situações em que vários eventos possam marcar o mesmo período.

Em relação à eficiência do Agente das Dicas, como mostram os resultados, este permite reduzir para metade o número de mensagens trocadas numa negociação entre o coordenador e os participantes, com o custo de apenas mais quatro mensagens adicionais. Nas situações negativas, em que a dica não foi aceite por um dos participantes, o custo em termos de mensagens não é significativo.

Capítulo 7

Conclusões

7.1 Trabalho Realizado

Neste trabalho de projeto desenvolveu-se uma aplicação para dar resposta ao desafio do alinhamento de agendas em dispositivos móveis. Um dos primeiro desafios foi o da escolha do tipo de terminal móvel a utilizar. Foram analisadas as soluções "Windows Phone" e "Apple iPhone" antes de avançar para o "Android". A primeira não disponibilizava nenhuma API que permitisse ligar ao exterior sem ser por "WebServices", e a segunda tinha custos associados ao desenvolvimento, pelo que a escolha foi fácil e seguramente a mais acertada dada a potencialidade e facilidade com que se desenvolvem aplicações para esta arquitetura. A utilização de "Android" e Java permitiu usar o JADE como plataforma para o desenvolvimento dos agentes. O JADE é uma ferramenta já bastante madura, bem documentada e com uma comunidade de utilizadores muito ativa.

A existência de preferências de utilizador permitem aos agentes agir e tomar decisões não apenas baseadas nas disponibilidades, mas também num conjunto de regras e restrições fracas que representam a vontade de um indivíduo.

No mecanismo de negociação implementado inicialmente, os participantes enviavam as suas propostas para o coordenador até que este encontra-se um alinhamento ("matching") total ou parcial dos períodos recebidos. Esta abordagem foi abandonada por ter vários problemas de desempenho (o incremento do número de dias da janela temporal do evento implicava também um aumento no número de mensagens trocadas entre os participantes e o coordenador) e de eficácia em chegar a uma solução. A abordagem mais natural do pedido de disponibilidades e da votação reduziu o número de interações entre os participantes (a janela temporal do evento não afeta o número de mensagens trocas) e melhorou a qualidade das soluções encontradas. A utilização de um protocolo de interação baseado em Iterated Contract-Net permitiu controlar a sequência de mensagens necessárias para a negociação, tirando partido do lançamento de novas interações para suportar as várias fases do processo (Pedido de Disponibilidades, Votação e Aceitação).

O Agente das Dicas provou ser capaz de chegar a uma solução com qualidade e com um consumo de recursos mais baixo do que o necessário numa negociação normal, conseguindo reduzir para metade o número de mensagens trocadas entre os participantes e o coordenador.

Nas situações em que não são fornecidas dicas, ou estas não são aceites pelos participantes, causa um reduzido impacto no número de mensagens trocadas.

7.2 Trabalho Futuro

Durante a análise e desenvolvimento do projeto, foram identificadas pelo menos três possibilidades de evolução do trabalho realizado que o poderiam complementar com novas funcionalidades ou reforçar características já implementadas, mas que em ambos os casos são áreas de trabalho muito interessantes.

1. Agentes estratégicos - Os agentes implementados neste projeto atuam de forma transparente e verdadeira na comunicação das suas disponibilidades ou no processo de votação, atuando sem qualquer tipo de estratégia definida. Os agentes poderiam no entanto manipular a informação que partilham com os outros agentes de forma a influenciar o resultado final. No processo de votação por exemplo, se um agente souber à partida que há dois fortes candidatos a vencer a votação, este pode colocar no topo do seu "ranking" o seu preferido, e no fim o outro candidato, mesmo que a sua classificação com base apenas nas preferências diga o contrário. Fica portanto em aberto o estudo das possibilidades de manipulação da negociação, a implementação destas estratégias nos agentes participantes e também a análise de estratégias de defesa ou prevenção desta manipulação.

2. Geo-referenciação - Outra possível evolução do trabalho consiste em considerar a posição geo-referenciada dos vários participantes como mais um fator a ter em conta no processo de negociação. A posição geográfica aproximada é muito fácil de obter tanto em dispositivos móveis como em postos fixos, e esta poderia ser considerada como fator influênciador tanto nas disponibilidades (podemos reconsiderar as disponibilidades de um agente se este estiver deslocado alguns quilómetros do seu local de trabalho predefinido) como na negociação (se o anfitrião tiver conhecimento que os participantes estão distanciados por mais de x km entre si, poderá não fazer sentido escolher períodos de tempo a curto prazo).

3. Eventos multi-período. A implementação atual permite agendar um tipo de evento em que o agente coordenador pretende reunir com todos os participantes em simultâneo, ou seja, no mesmo período e no mesmo local. Seria interessante ter a possibilidade de agendar um outro tipo de evento, onde o coordenador disponibiliza não um, mas vários períodos, sendo o objetivo final encontrar-se não em simultâneo, mas individualmente com cada um dos participantes em períodos distintos. Neste caso, a negociação com os participantes e a eleição dos períodos já não seria por meio de votação, mas sim por licitação, onde cada um dos agentes participantes deveria licitar os períodos ainda disponíveis, como se de um leilão se tratasse. Só a definição do conceito de "preço" ou "valor" da licitação já é um desafio em si!

Referências

[Aseere et al.(2010)Aseere, Gerding, and Millard] A. M. Aseere, E. H. Gerding, and D. E. Millard. A voting-based agent system for course selection in e-learning. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Toronto, Canada, 2010.

[Baharad and Nitzan(2005)] E. Baharad and S. Nitzan. The inverse plurality rule - an axiomatization. Technical report, Department of Economics, The University of Haifa and Bar Ilan University, Israel, 2005.

[Bellifemine et al.(2003)Bellifemine, Caire, Poggi, and Rimassa] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE - A White Paper. Technical report, Italia, 2003.

[Bellifemine et al.(2010)Bellifemine, Caire, Trucco, and Rimassa] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. JADE programmer's guide. Technical report, Italia, 2010.

[Caire and Pieri(2010)] G. Caire and F. Pieri. LEAP user guide. Technical report, Italia, 2010.

[Conitzer(2010a)] V. Conitzer. Comparing multiagent systems research in combinatorial auctions and voting. Technical report, Department of Computer Science, Duke University, Durham, NC 27708, USA, 2010a.

[Conitzer(2010b)] V. Conitzer. Making decisions based on the preferences of multiple agents. In *Communications of the ACM*, pages 84–94, New York, NY, USA, 2010b.

[Dawson et al.(1998)Dawson, Lotus, Stenerson, and Microsoft] F. Dawson, Lotus, D. Stenerson, and Microsoft. *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*, 1998. URL `http://www.ietf.org/rfc/rfc2445.txt`.

[D'Ercole and Trigo(2011)] J. D'Ercole and P. Trigo. Multi-Agent Negotiation using Mobile Devices - Ontology and strategies for autonomous meeting scheduling. In *CETC 2011 - Conference on Electronics, Telecommunications and Computers*, Lisboa, Portugal, 2011. ISEL. Best paper in the computer science track.

[develop123(2010)] develop123. *From Protégé to JADE ontology classes*, 2010. URL `http://agents.develop123.com/index.php/getting-started-with-jade/90-from-protege-to-jade-ontology-classes`.

[Eisinger and Elshiewy(1992)] N. Eisinger and N. Elshiewy. Madman - multi-agent diary manager. In *Proc DAI - Workshop at European Conf. on Artificial Intelligence*, Vienna, Austria, 1992.

[F. Brandt and Endriss(2012)] V. C. F. Brandt and U. Endriss. Computational social choice. In *G. Weiss (Ed.), Multiagent Systems, MIT Press*, 2012.

[FIPA(2012)] FIPA. *The Foundation for Intelligent Physical Agents*, 2012. URL `http://www.fipa.org/`.

[Franzin et al.()Franzin, Freuder, Rossi, and Wallace] M. Franzin, E. C. Freuder, F. Rossi, and R. Wallace. Multi-agent Constraint System with Preferences: Efficiency, Solution Quality, and Privacy Loss. Technical report, Department of Pure and Applied Mathematics, University of Padova and Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Cork.

[Google(2011)] Google. Google Data Protocol, 2011. URL `http://code.google.com/apis/gdata/docs/2.0/elements.html`.

[Google(2012)] Google. *Google Calendar API*, 2012. URL `https://developers.google.com/google-apps/calendar/v2/developers_guide_java#GettingStarted`.

[Gruber(1992)] T. R. Gruber. A translation approach to portable ontology specifications. Technical report, Computer Science Department - Stanford University, Stanford, California, USA, 1992.

[JADE(2012)] JADE. *Java Agent DEvelopment Framework*, 2012. URL `http://jade.tilab.com/`.

[Jennings and Jackson(1995)] N. R. Jennings and A. J. Jackson. Agent-based meeting scheduling: A design and implementation. Electronic letter, Queen Mary & Westfield College, London, 1995.

[Mattern and Sturm(1989)] F. Mattern and P. Sturm. An automatic distributed calendar and appointment system. Technical report, INCAS Project, University of Kaiserslautern, Kaiserslautern, Germany, 1989.

[Niederer and Schatten(2009)] M. Niederer and A. Schatten. Agent-based meeting scheduling support using mobile clients. Technical report, Institute of Software Technology and Interactive Systems, Vienna, Austria, 2009.

[Sen and Durfee(1992)] S. Sen and E. H. Durfee. Automated meeting scheduling among heterogeneous agents. In *Proc AAAI92 Workshop on Cooperation Among Heterogeneous Agents*, pages 116–120, San Jose, CA, 1992.

[Stanford Proteje WIKI(2008)] Stanford Proteje WIKI. *OntologyBeanGenerator 4.0*, 2008. URL `http://protegewiki.stanford.edu/wiki/OntologyBeanGenerator_4.0`.

[The GNOME Project(2012)] The GNOME Project. *GNOME Dev Center*, 2012. URL `http://developer.gnome.org/ontology/unstable/ontology.html`.

[Tsang(1993)] E. Tsang. Foundations of constraint satisfaction. In *Academic Press*, London and San Diego, 1993.

[W3C(2012)] W3C. *W3C Semantic Web*, 2012. URL `http://www.w3.org/2001/sw/`.