

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

API .NET para execução paralela e distribuída

Micael Filipe Moreira Queiroz

Relatório de Projecto

Mestrado Integrado em Engenharia Informática e Computação

Orientador: António Miguel Pontes Pimenta Monteiro (PhD)

17 de Julho de 2009

API .NET para execução paralela e distribuída

Micael Filipe Moreira Queiroz

Relatório de Projecto

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: Ana Paula Cunha da Rocha (PhD)

Arguente: Carlos Miguel Ferraz Baquero Moreno (PhD)

Vogal: António Miguel Pontes Pimenta Monteiro (PhD)

17 de Julho de 2009

Resumo

Pelos mais variados motivos e englobando as mais diversas áreas, tem-se assistido a uma incessante luta evolutiva com o objectivo de fazer mais, e melhor, em menos tempo. É neste tipo de cenário que hoje em dia os sistemas computacionais desempenham um papel fundamental. De modo a acompanhar as crescentes necessidades computacionais, surgiu a ideia de distribuir o processamento. A transformação impetuosa deste mero conceito de computação distribuída em genuína realidade permitiu a um grande número de indivíduos usufruir de um poder computacional que estaria de outra forma desperdiçado.

Actualmente, os sistemas distribuídos são usados pelos mais variados motivos, estendendo-se ao longo de uma vasta diversidade de áreas, e operando em diferentes escalas, desde o aplicativo que unifica os computadores, presentes num escritório, até o que interliga computadores à escala mundial.

Existem diversas aplicações de computação científica e numérica que, devido ao volume de processamento envolvido na sua operação, podem ser divididas com vantagem em múltiplas tarefas independentes. De modo a tirar partido desta divisibilidade é necessário um sistema capaz de receber, e distribuir devidamente as tarefas, pelos demais computadores disponíveis na rede. Ao distribuir o processamento espera-se conseguir uma diminuição do tempo de operação da aplicação, face ao processamento local.

O projecto consiste na concepção, implementação e teste de uma *API* e respectiva plataforma de suporte para possibilitar o processamento distribuído. Este sistema, permitirá ao programador desenvolver aplicações para processamento distribuído, de forma simples e transparente. O cenário de utilização reduz-se a um laboratório onde não está prevista a existência de mais de duas dezenas de máquinas a interagir usando o sistema distribuído.

A linguagem de programação usada foi o *C#*, as tecnologias usadas pertencem à *framework .NET*. A comunicação assenta fundamentalmente na tecnologia *.NET Remoting*.

Os objectivos foram plenamente atingidos. Para comprovar que os requisitos iniciais foram efectivamente cumpridos foi desenvolvida uma aplicação de demonstração. Durante os testes e respectiva análise de resultados foi possível estabelecer uma comparação entre processamento local e distribuído, tornando assim fácil avaliar a eficácia do sistema desenvolvido.

Abstract

By a wide variety of reasons and encompassing a diversity of areas, the incessant struggle for evolution never stops. Nowadays the computational systems play a role of most importance in this kind of scenario. To overcome the rapidly increasing needs of computational power, the idea of distributed processing was taken into consideration. The impetuous transformation of this mere concept into a genuine reality, allowed a wide number of individuals to exploit a computational power that until this point was unusable.

Today, the distributed systems are used due to a wide variety of reasons, extend through a great number of areas of expertise and operating at different scales. Using this technology is equally possible to interconnect the computers of an office network or computers place all around the globe.

Several numeric and scientific applications require a great volume of processing time to operate. In some cases and due to this reason they can be sub-divided into multiple independent tasks. To take advantage from this divisibility a system capable of receiving, and conveniently distribute the tasks to the available computers in the network, is indispensable. By doing this, the application operation time will hopefully decrease, compared to local processing.

The project consists in the design, implementation and test of an *API* and respective support platform to enable distributed processing. Using this system a programmer will be able to painlessly and transparently develop a distributed application. The typical usage scenario is "reduced" to a lab containing a max of twenty computers. The *C#* was the used programming language and the communication essentially relies in the .NET *Remoting* technology.

The objectives were fully fulfilled. In order to show that the initial requirements were met, a demonstrative application was developed. During the tests and sub-sequent result analysis was possible to establish a comparison between local and distributed processing, due to this was easy to measure the performance of the developed system.

Agradecimentos

Em primeiro lugar queria agradecer a todos os que intervieram directa e indirectamente no meu processo académico, o que me permitiu chegar até aqui e ao mesmo tempo atingir estes resultados.

De seguida, gostaria de agradecer a todos os meus colegas e amigos que me auxiliaram no decorrer deste projecto.

Os meus agradecimentos vão agora para o meu orientador por todo o apoio prestado. Desde a disponibilidade e interesse demonstrado no projecto e no estágio, até ao rigor das decisões e opiniões.

Gostaria, também, de agradecer ao INEB pelo bom ambiente proporcionado durante o desenvolvimento do projecto.

Por fim, os meus agradecimentos vão para a minha namorada e futura esposa por todo o apoio prestado nos bons e maus momentos, sem o qual não teria sido possível alcançar os resultados obtidos.

Micael Filipe Moreira Queiroz

“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

Leslie Lamport
Founder of the theory of distributed systems

Conteúdo

1	Introdução	1
1.1	Contextualização	1
1.2	API .NET para processamento paralelo e distribuído (Divisão de Sinal e Imagem - INEB - FEUP)	2
1.3	Motivação e objectivos	2
1.4	Estrutura da dissertação	3
2	O Projecto	5
2.1	Descrição do projecto	5
2.2	Requisitos	6
2.3	Abordagem	8
2.4	Planeamento	8
3	Estado da Arte	10
3.1	Visão geral	10
3.2	Tecnologias específicas para computação paralela e distribuída	12
3.3	Projectos relacionados	16
3.4	Análise tecnológica (estudo do .NET Remoting)	18
3.4.1	Tipos de remoting	18
3.4.2	Tipos de activação	19
3.4.3	Tipos de invocação	21
3.4.4	Partilha de <i>assemblies</i>	23
3.4.5	Escolha do protocolo	24
3.4.6	Configurações multi-servidor	25
3.4.7	Configuração do .NET Remoting	26
3.4.8	Instalação	27
3.4.9	Segurança	27
3.4.10	Notificações	29
3.4.11	<i>Assemblies</i>	31
3.4.12	Transferência de <i>assemblies</i>	31
3.4.13	Comunicação entre domínios aplicativos e injeção de <i>assemblies</i>	32
3.4.14	Gestão do tempo de vida dos objectos	33
4	Arquitectura	36
4.1	Arquitectura física	37
4.2	Arquitectura lógica	38
4.2.1	Registo dos clientes	40

CONTEÚDO

4.2.2	Construção da Grid	40
4.2.3	Ciclo de execução	40
4.2.4	Cancelamento de pedidos	42
4.2.5	Ausências inesperadas	42
4.2.6	Abortar processamento	42
4.3	Diagramas de composição	43
4.3.1	Framework	43
4.3.2	API	44
4.3.3	GridMaster	45
4.3.4	GridWorker	46
4.4	Diagramas de interacção	48
4.4.1	API e GridMaster	48
4.4.2	GridWorker e GridMaster	50
4.4.3	WorkerAssembly e GridMaster	52
5	Implementação	54
5.1	GridMaster	55
5.1.1	Configuração	56
5.2	FrameWork	58
5.3	GridWorker	58
5.3.1	Configuração	59
5.4	API	60
5.4.1	Metodologia para uso da <i>API</i>	61
5.4.1.1	Conexão e Registo	62
5.4.1.2	Submissão de Pedidos	62
5.4.1.3	Espera	62
5.4.1.4	Cancelamento	63
5.4.1.5	Recepção de Resultados	63
5.4.1.6	Termo da conexão	63
5.5	Interface com o utilizador	64
5.5.1	GridWorkerManager	65
5.5.2	GridMasterManager	66
5.6	Testes	68
6	Aplicação de Demonstração e Análise de Resultados	69
6.1	Introdução	69
6.1.1	Contextualização	69
6.1.2	O Algoritmo	70
6.1.3	Motivação e objectivos	71
6.1.4	Descrição da aplicação	71
6.2	Arquitectura	72
6.2.1	Diagrama de composição	73
6.2.2	Diagrama de interacção	74
6.3	Implementação	75
6.3.1	Interface com o utilizador	76
6.3.2	Exemplos de execução	79
6.4	Análise de resultados	80

CONTEÚDO

6.4.1	Ensaio 1	81
6.4.2	Ensaio 2	83
6.4.3	Ensaio 3	84
7	Conclusões e Trabalho Futuro	85
7.1	Acordo com o planeamento	85
7.2	Trabalho futuro	86
	Referências	90
A	Extractos de código	91
A.1	Método Timeout	91
A.2	Exemplo de uso da API	95
A.3	Algoritmo Mandelbrot	99
B	Exemplos de execução	100
B.1	Exemplo de execução local	100
B.2	Exemplo de execução distribuída	102

Lista de Figuras

2.1	Arquitectura do projecto (versão simplificada)	6
2.2	Planeamento do projecto	8
3.1	Tipos de activação	20
3.2	Tipos de invocação	22
3.3	Cenário de partilha de <i>assemblies</i> e interfaces	24
3.4	Cenário multi-servidor	25
3.5	Diferentes cenários de notificação	30
3.6	Diferentes cenários de gestão do ciclo de vida	35
4.1	Arquitectura física do projecto	37
4.2	Diagrama de composição geral	38
4.3	Arquitectura lógica do projecto	39
4.4	Composição do componente Framework	43
4.5	Composição do componente API	44
4.6	Composição do componente GridMaster	45
4.7	Composição do componente GridWorker	46
4.8	Interacção entre API e GridMaster	49
4.9	Interacção entre GridWorker e GridMaster	51
4.10	Interacção entre WorkerAssembly e GridMaster	53
5.1	Divisão percentual do tempo durante a fase de implementação	54
5.2	Árvore de ficheiros da solução GridMaster	55
5.3	Ficheiro de configuração da aplicação GridMaster	56
5.4	Árvore de ficheiros da solução Framework	58
5.5	Árvore de ficheiros da solução GridWorker	58
5.6	Ficheiro de configuração da aplicação GridWorker	59
5.7	Árvore de ficheiros da solução API	60
5.8	Funcionalidades disponibilizadas pela <i>API</i> ao programador	61
5.9	Estados possíveis de uma aplicação que use as funcionalidades disponi- bilizadas pela <i>API</i>	61
5.10	Interface da aplicação GridWorkerManager	65
5.11	Interfac da aplicação GridMasterManager	66
5.12	Divisão percentual do tempo durante a fase de testes, orientada às tarefas .	68
5.13	Divisão percentual do tempo durante a fase de testes, orientada às entidades	68
6.1	Exemplo de <i>fractal</i> representativo do conjunto de <i>Mandelbrot</i>	70
6.2	Arquitectura da aplicação de demonstração "MandelbrotViewer"	72

LISTA DE FIGURAS

6.3	Composição da aplicação de demonstração "MandelbrotViewer"	73
6.4	Interacção entre as componentes "Interface" e "BackGroundWorker"	74
6.5	Árvore de ficheiros da solução MandelbrotViewer	76
6.6	Interface da aplicação de demonstração "MandelbrotViewer"(tab 1)	77
6.7	Interface da aplicação de demonstração "MandelbrotViewer"(tab 2)	78
6.8	Exemplo de <i>fractal</i> representativo do conjunto de <i>Mandelbrot</i>	81
6.9	Exemplo de <i>fractal</i> representativo do conjunto de <i>Mandelbrot</i> , com uma região seleccionada	83
6.10	Gráfico de variação do tempo em função do número de executores	84
B.1	Exemplo de execução da aplicação MandelbrotViewer aquando a geração da imagem	100
B.2	Exemplo de execução da aplicação MandelbrotViewer após a geração da imagem. (A imagem contém um exemplo de selecção usando o rato)	101
B.3	Aspecto da aplicação MandelbrotViewer enquanto gera a imagem (tab 1) . . .	102
B.4	Aspecto da aplicação MandelbrotViewer enquanto gera a imagem (tab 2) . . .	102
B.5	Exemplo de execução da aplicação GridMasterManager	103
B.6	Exemplo de execução da aplicação GridWorkerManager	103
B.7	Aspecto da aplicação MandelbrotViewer após gerar a imagem (tab 1) . . .	104
B.8	Aspecto da aplicação MandelbrotViewer após gerar a imagem (tab 2) . . .	104

Lista de Tabelas

6.1	Descrições dos computadores	80
6.2	Configuração distribuída	81
6.3	Ensaio 1	82
6.4	Ensaio 2	83
6.5	Configuração distribuída	84
6.6	Ensaio 3	84

Abreviaturas e Símbolos

.NET	Deriva da arquitectura para web-services da microsoft denominada "dot-net", apesar das letras capitalizadas não se trata de um acrónimo. Trata-se de uma marca de um conjunto de frameworks e tecnologias pertencentes á Microsoft
API	Application Programming Interface
BOINC	Berkeley Open Infrastructure for Network Computing
CAO	Client Activated Objects
COM	Component Object Model
CORBA	Common Object Request Broker Architect
CPU	Central Processing Unit
DCOM	Distributed Component Object Model
FEUP	Faculdade de Engenharia da Universidade do Porto
HTTP	Hyper Text Transport Protocol
I/O	Input / Output
IDE	Integrated Development Environment
IIS	Internet Information Server
INEB	Instituto Nacional de Engenharia Biomédica
IP	Internet Protocol
Java RMI	Java Remote Method Invocation
JVM	Java Virtual Machine
LAN	Local Area NetWork
MSMQ	Microsoft Message Queuing
MTS	Microsoft Transaction Server
OMG	Object Management Group
OO	Object Oriented
PC	Personal Computer
RPC	Remote Procedure Call
SAO	Server Activated Objects
SETI@home	Search for Extra-Terrestrial Intelligence at home
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
VS	Microsoft Visual Studio
WS	Web Service
WSDL	Common Object Request Broker Architect
XML	Extended Markup Language

Capítulo 1

Introdução

1.1 Contextualização

Desde há muito tempo que houve o anseio de construir mecanismos programáveis que se adaptassem às necessidades do ser humano [Onl09c]. Alguns anos depois do aparecimento dos primeiros computadores, assemelháveis aos computadores modernos, a crescente necessidade de computação [Onl09b] originou o aparecimento dos chamados "super computadores"[Onl09g]. Durante alguns anos esta constituiu a melhor solução para o chamado processamento "intensivo". A acelerada disseminação dos chamados *PCs*¹, ao longo dos anos, conduziu a uma globalização de recursos computacionais. O aperfeiçoamento da conectividade global conduziu à existência de modelos de computação distribuída [Onl09a].

Tendo como objectivo aproveitar a abundância global de recursos computacionais, surgiu o conceito de "Sistema de processamento distribuído"[Onl09f]. Este tipo de sistemas interliga vários recursos computacionais (não necessariamente iguais), tendo como objectivo o processamento paralelo e distribuído de tarefas (supostamente subdivisíveis e de elevado "peso" computacional).

O projecto que será aqui exposto insere-se neste arquétipo de sistema.

¹*PC: Personal Computer*, ou computador pessoal.

1.2 API .NET para processamento paralelo e distribuído (Divisão de Sinal e Imagem - INEB - FEUP)

Este projecto enquadra-se nas necessidades dos elementos relativos à divisão de sinal e imagem pertencente ao INEB², e sediada na FEUP³. Das suas actividades principais fazem parte o processamento e análise de sinais e imagens, no contexto da biomedicina.

1.3 Motivação e objectivos

As aplicações de computação científica e numérica para o processamento de sinal e imagem, devido ao volume de dados envolvidos, e ao respectivo processamento que estes implicam podem, muitas vezes, ser divididas com vantagem em múltiplas tarefas independentes, que podem ser executadas de forma paralela e distribuída. O objectivo deste projecto será, então, conceber, planear e implementar uma API⁴ que permita ao programador desenvolver aplicações para processamento paralelo e distribuído de forma simplificada, ou seja, sem ser obrigado a uma programação explícita para o efeito. Ficará a cargo do programador dividir o processamento em conjuntos de tarefas independentes entre si.

²**INEB**: Instituto Nacional de Engenharia Biomédica, trata-se de uma associação da Universidade do Porto, privada e sem fins lucrativos, cujo objectivo é criar uma ligação entre a academia e o sector da saúde nas áreas das engenharias biomédicas.

³**FEUP**: Faculdade de Engenharia da Universidade do Porto.

⁴**API**: *Application Programming Interface*.

1.4 Estrutura da dissertação

Seguidamente, será apresentado, de forma sucinta, a disposição da presente dissertação bem como, uma breve descrição de cada uma das partes em que está dividida.

- **2. O Projecto**

Neste capítulo, apresenta-se uma descrição geral do projecto e dos respectivos requisitos. É, também, definida a abordagem e o respectivo planeamento para a sua concretização.

- **3. Estado da Arte**

No início do terceiro capítulo é feita uma análise ao panorama actual do conceito geral ”*Grid Computing*”, fazendo-se referência a alguns projectos globalmente conhecidos onde é aplicado este conceito. A segunda secção é dedicada à análise tecnológica, apresentando-se inicialmente algumas tecnologias alternativas. Descreve-se, ainda, a tecnologia *.NET Remoting*⁵ e mencionam-se alguns projectos que usam esta tecnologia. A última secção contém uma análise tecnológica, onde é explicado que funcionalidades do *.NET Remoting* serão usadas, e porquê.

- **4. Arquitectura**

O conteúdo deste capítulo pretende explicitar a arquitectura de toda *framework* em *.NET*⁶ que se pretendeu desenvolver. Os diagramas que serão apresentados irão evidenciar a composição e a interacção entre os diversos elementos que a constituem.

⁵**.NET Remoting**: API da Microsoft para a comunicação entre processos. Consultar [Cor09h].

⁶**.NET**: habitualmente refere-se à *framework .NET*. Esta trata-se de uma *framework* que inclui uma vasta biblioteca que contém soluções para os problemas comuns dos programadores. Consultar [Cor09g].

- **5. Implementação**

No quinto capítulo ilustra-se sucintamente a concretização da arquitectura apresentada anteriormente. A primeira parte do capítulo contém detalhes da implementação julgados pertinentes, quer pela sua importância, quer pela sua complexidade. Para melhor interagir com as componentes da *framework* foram criadas duas aplicações, que funcionam como interfaces facilitadoras. Estas aplicações e as interfaces que as definem serão apresentadas na segunda parte.

- **6. Aplicação de Demonstração e Análise de Resultados**

Para demonstrar as potencialidades da *framework*, implementou-se uma aplicação de demonstração. No decorrer deste capítulo será apresentada a sua arquitectura e serão mostrados alguns exemplos de execução. Na última secção será apresentada uma análise de performance em ambiente distribuído e as respectivas conclusões.

- **7. Conclusões e Trabalho Futuro**

Inicialmente apresentam-se considerações sobre o grau de sucesso do projecto. No final, discutem-se alguns possíveis melhoramentos e incrementos a aplicar ao projecto desenvolvido.

O documento contém, ainda, as referências utilizadas para a sua redacção, assim como os anexos, onde é possível encontrar trechos de código relevantes (anexo [A](#)), e exemplos de execução (anexo [B](#)).

Capítulo 2

O Projecto

Com o intuito de dar a conhecer um pouco do projecto, segue-se uma breve descrição, analisando, posteriormente, os seus requisitos. Para que seja possível concluir o projecto no curto prazo disponível é necessária uma abordagem sólida, complementada com um planeamento realista.

2.1 Descrição do projecto

O projecto consiste na implementação de uma *framework* orientada a objectos (em *.NET* e *C#*¹) que servirá de plataforma integradora de aplicações computacionais existentes na rede. É possível dividir as implementações em dois grupos: as que disponibilizam os recursos, e as que deles usufruem. Para que seja possível aos aplicativos usufruírem dos recursos da rede, é necessário desenvolver uma *API* simples e, ao mesmo tempo, poderosa e flexível. Assim, será possível a um programador definir, facilmente, um conjunto de tarefas, e colocá-las em execução simultânea num conjunto de recursos distribuídos e previamente configurados para o efeito. Os sistemas computacionais destinados à disponibilização de recursos serão representados por uma entidade "grelha" (denominada "*Grid*"). Este cenário pode ser observado na figura 2.1.

A implementação residente no cliente será capaz de estabelecer uma conexão com o servidor para que, de seguida, seja possível encapsular e submeter (ao servidor da grelha) as tarefas para execução, ficando (ou não) a aguardar pela sua conclusão. Toda a informação necessária à conclusão da tarefa é enviada com a própria, pelo que não há trocas posteriores, ou seja, as tarefas são independentes entre si. É possível ao programador usar a aplicação residente para interagir com o servidor, visto esta tratar-se de uma *API*.

¹ *C#*: Linguagem de programação multi-paradigma, funcional, imperativa, genérica, orientada a objectos e orientada a componentes. Foi desenvolvida pela Microsoft no contexto da iniciativa *.NET*.

O Projecto

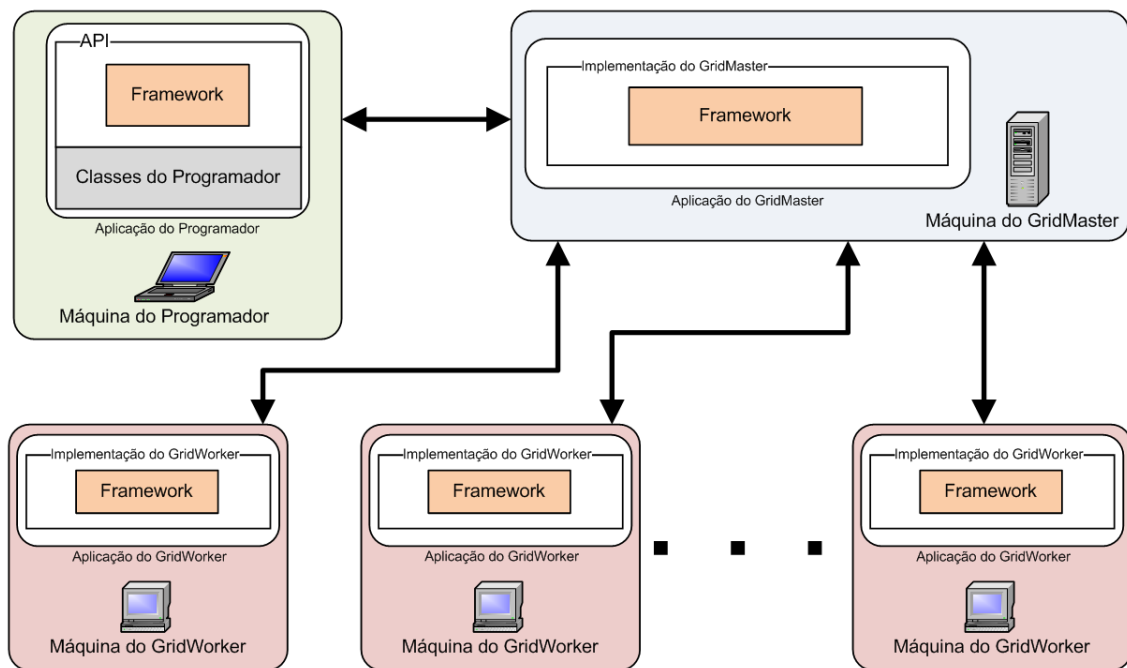


Figura 2.1: Arquitectura do projecto (versão simplificada)

Usando os métodos presentes nesta biblioteca, seguindo uma certa sequência, é possível colocar uma aplicação a executar de forma paralela e, simultaneamente, distribuída (em vários recursos de computação). Para o fazer é necessário conhecer as configurações de acesso ao servidor da grelha e compreender o modo de funcionamento da *API*.

2.2 Requisitos

Para responder com eficácia ao funcionamento pretendido e descrito na secção anterior, a *API* a disponibilizar ao programador deve respeitar os seguintes requisitos:

- Disponibilizar um conjunto de métodos devidamente divididos em classes, para que, ao serem usados, segundo uma certa ordem, possibilitem o processamento paralelo e distribuído de tarefas.
- Possuir nomenclatura adequada, ou seja, as suas classes e os métodos que as definem devem possuir nomes elucidativos, para que se torne fácil, para um programador, desvendar rapidamente que métodos usar, e que ordem seguir.
- Ser compacta, ou seja, deve possuir a quantidade adequada de métodos, introduzindo a flexibilidade necessária para ir ao encontro das necessidades do programador mas, ao mesmo tempo, não deve deixar transparecer a complexidade de processamento que existe durante a interação com a grelha.

O Projecto

- Ser o mais portátil possível, ou seja, não deve depender de módulos exteriores evitando, assim, a necessidade de instalações adicionais.
- Ser simples de usar, ou seja, não ser necessário conhecer em detalhe o modo de funcionamento da grelha mas antes, apenas os parâmetros de acesso ao servidor.
- Estar devidamente estruturada e documentada, proporcionando ao programador facilidade de utilização.
- Deve ser transparente e flexível, ou seja, o programador deverá poder executar as suas tarefas localmente e, de seguida, passar para uma execução distribuída com relativa facilidade.
- Deve ser capaz de se auto-diagnosticar tornando-se, assim, fácil para o programador desvendar a causa de qualquer anomalia.

O sistema de processamento, do qual a *API* irá depender, é fisicamente constituído por vários computadores ligados entre si em rede *LAN*². Espera-se que este sistema computacional possua as seguintes características:

- Ser facilmente configurável em todos os aspectos, desde a própria constituição da grelha, até à configuração individual dos elementos que a constituem.
- Deve ser capaz de receber pedidos de processamento provenientes de vários clientes diferentes em simultâneo e distribuir as tarefas pelos diversos recursos disponíveis na rede.
- Quando do termo de cada tarefa os resultados devem ser entregues ao cliente que fez o pedido de processamento.
- Ser capaz de detectar e recuperar de uma situação de *deadlock*, que pode ocorrer quando da execução de tarefas submetidas.
- Reagir devidamente quando da ausência inesperada de um cliente, e caso este tenha submetido tarefas para execução, esta execução deve ser cessada o quanto antes.
- Reagir devidamente quando da ausência inesperada de uma unidade de processamento. E caso esta esteja de momento a processar alguma tarefa, deve existir uma tentativa de re-alocação da tarefa ou tarefas em causa.
- O nó central deve manter o bom funcionamento independentemente do comportamento dos clientes que com ele interagem, procurando assim garantir disponibilidade e qualidade de serviço.

²**LAN:** *Local Area Network*.

2.3 Abordagem

Para o desenvolvimento do projecto procurou-se seguir um conjunto de passos que constituíram o seu planeamento. Com estes passos pretendeu-se rentabilizar o tempo disponível e garantir um resultado satisfatório. O conjunto de passos seguidos foi então:

- Análise da especificação e requisitos fornecidos.
- Análise tecnológica.
- Definição da arquitectura.
- Implementação da *framework*.
- Teste da *framework*.
- Desenvolvimento da aplicação de demonstração.
- Teste da aplicação de demonstração e análise de resultados.
- Documentação final.

2.4 Planeamento

Para que fosse possível a conclusão do projecto com êxito foi necessário planear o seu desenrolar, tendo sempre em atenção os prazos inicialmente estipulados. O planeamento proposto pode observar-se na figura 2.2, acabando por traduzir-se nas seguintes tarefas:

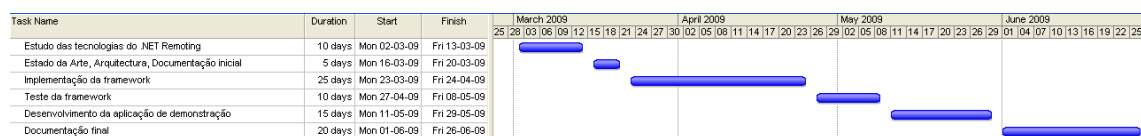


Figura 2.2: Planeamento do projecto

• De 02 de Março a 13 de Março

Estudo das Tecnologias do .NET Remoting - Estudo aprofundado das tecnologias do .NET Remoting para ser possível definir devidamente a arquitectura. A análise tecnológica apenas incidiu nas características do .NET Remoting pois, já estaria decidido à partida, que o projecto usaria esta tecnologia como base.

• De 16 de Março a 20 de Março

Estado da arte, arquitectura, documentação inicial - Efectuar um estudo inicial do estado da arte. De seguida, e com base no que fora investigado até agora, definir a arquitectura. E por fim, documentar tudo o que foi descoberto e produzido até ao momento.

- **De 23 de Março a 24 de Abril**

Implementação da framework - Implementação de tudo o que foi projectado na fase de arquitectura, usando as tecnologias inicialmente testadas, e efectuando testes incrementais de cada módulo desenvolvido.

- **De 27 de Abril a 08 de Junho**

Teste da framework - Testar o funcionamento da *framework* como um todo. Visto ser uma aplicação distribuída e fortemente concorrente, as diversas interações conduzem a uma infinidade de sequências de execução. Como tal, é de extrema importância testar o maior número possível de situações.

- **De 11 de Junho a 29 de Junho**

Desenvolvimento da aplicação de demonstração e análise de resultados -

Implementação de dois aplicativos que servem como interfaces facilitadoras, para melhorar a interacção com os componentes da *framework*. De seguida, implementar um aplicativo de demonstração que use as potencialidades da *framework*, que se encontre no contexto das actividades do INEB e que, de preferência, seja visualmente mais agradável. Por fim, testar todas as componentes em ambiente distribuído, e proceder à respectiva análise de resultados.

- **De 01 de Julho a 26 de Julho**

Documentação final - Com base em tudo o que foi feito, rever e refinar a documentação inicial para, a partir desta, elaborar o relatório de dissertação final do projecto.

Capítulo 3

Estado da Arte

Antes de dar início ao desenvolvimento da *framework*, foi necessário atender a várias questões. Em qualquer projecto de *software* é essencial escolher a tecnologia que melhor se adapta à realidade da aplicação a desenvolver. Neste caso em particular, a tecnologia já foi previamente escolhida, restando apenas realizar um estudo, aos diversos mecanismos que a constituem. Esta realidade não impede a existência de uma pequena análise às características de outras tecnologias. Para além da essencial análise tecnológica, é importante examinar o panorama actual, no que diz respeito à computação paralela e distribuída. A análise de projectos relacionados com o tema e respectivas plataformas de suporte é igualmente importante. Todos estes tópicos são abordados neste capítulo.

3.1 Visão geral

Actualmente o tema "Computação Paralela e Distribuída" [Onl09a] assume proporções consideráveis, indo desde as aplicações de processamento paralelo residentes em apenas uma máquina, até à integração de sistemas diversificados à escala mundial. Os motivos pelo qual se interligam diversas máquinas são variados e podem ir desde a necessidade de garantir uma certa disponibilidade ou qualidade de serviço, até ao aumento do poder de processamento. Ao longo do tempo, foram-se desenvolvendo diversos conceitos, relativamente ao modo como esta interligação é construída. Uma das filosofias que actualmente assume maior importância é a de *Grid Computing* [cGp09a]. Esta é facilmente escalável, podendo resumir-se a um conjunto de computadores (*tightly coupled*¹) ligados

¹*tightly coupled*: ou "fortemente ligados". A definição de sistema *tightly coupled* abrange os sistemas em que os *CPUs* estão de tal forma conectados que se um elemento falha, pode comprometer o funcionamento do sistema inteiro. Neste tipo de sistemas é habitual a partilha de memória e de recursos de *I/O*.

entre si, através de uma rede *LAN*, ou pode ser globalmente abrangente, interligando diferentes redes e organizações geograficamente dispersas, heterogêneas e *loosely coupled*². Tem-se assistido a uma rápida disseminação de aplicações de software, que vão desde aplicações e serviços distribuídos ao *middleware*³ de suporte, tornando cada vez mais real este conceito[20009][cGp09b][Gri09a].

Quando do aparecimento de plataformas adequadas (*middleware*[Gri09b]), começaram a surgir imensos projectos [Gri09c]. São exemplos deste tipo de plataformas a *Folding@home* [ISU09] (projecto dedicado à dobragem de proteínas e outras operações moleculares) e o *BOINC*⁴. Tendo esta última sido inicialmente desenvolvida para suportar apenas o projecto *SETI@home*⁵, tornou-se profícua para outras aplicações distribuídas nas mais diversas áreas, permitindo aos investigadores usufruir de um enorme poder de processamento. Actualmente, existem vários projectos de plataformas para computação distribuída (como por exemplo o *BEinGRID*[BEi09], o *GridsForE-Sci*[EGE09], o *Interactive European Grid Project*[Gri09e], e o *Euro Grid*[Gri09d] a nível europeu ou *Worldwide LHC Computing Grid*[Gri09f] a nível mundial) apresentando-se estes nas mais diversas áreas. Com a utilização crescente destas arquitecturas consegue-se uma optimização de recursos, diminuindo a complexidade, e consequente o custo dos recursos computacionais, permitindo aos cientistas e investigadores usufruírem de um poder de processamento abundante e barato, favorecendo em larga escala as suas actividades de pesquisa. Por estes motivos, a sua utilização constituiu uma forte alternativa ao uso dos designados "super computadores". O cenário em estudo remete-se para a situação em que o *grid computing* se resume a um conjunto relativamente pequeno e concentrado de computadores, ligados em rede através de uma rede local de alta velocidade, não dedicada exclusivamente ao processamento paralelo e distribuído de tarefas. Numa primeira instância, este será o ambiente de desenvolvimento, teste e utilização. Numa situação futura a aplicação será possivelmente instalada num ambiente de *cluster computing* [And09].

²*loosely coupled*: ou "fracamente ligados". Os elementos constituintes dos Sistemas *loosely coupled* são heterogêneos e habitualmente encontram-se dispersos uns relativamente aos outros. Para que a interacção seja possível, é necessário recorrer a mecanismos específicos.

³*middleware*: *software* que interliga componentes de *software* ou aplicações.

⁴*BOINC*: *Berkeley Open Infrastructure for Network Computing*, trata-se de um sistema não comercial de *middleware* para *grid computing*. Consultar [UoC09].

⁵*SETI@home*: *Search for Extra-Terrestrial Intelligence at home*, trata-se de um projecto de computação distribuída que usa computadores conectados através da Internet. Consultar [oC09].

3.2 Tecnologias específicas para computação paralela e distribuída

Ao longo dos anos foram surgindo diversas tecnologias e metodologias para possibilitar o desenvolvimento de sistemas com arquiteturas distribuídas. Uma das primeiras tecnologias com sucesso surgiu no início da década de 90 e denominava-se *CORBA*⁶. Tratando-se esta de uma especificação (definido pela *OMG*⁷.) para lidar com objectos distribuídos, foi sendo usada para construir aplicações distribuídas e heterogéneas, independentemente da plataforma ou linguagem de programação. Algum tempo depois, a Microsoft lançou a tecnologia *COM*⁸, sendo inicialmente esta de baixo nível para a comunicação entre processos (ambos na mesma máquina) e criação dinâmica de objectos. Mais tarde, surgiu o *DCOM*⁹, uma extensão do *COM* para ambientes distribuídos onde a comunicação local entre processos é substituída por um protocolo de rede, sendo a sua operação transparente para o programador, ou seja, todos os detalhes de baixo nível da comunicação em rede ficam escondidos. Mais tarde, a Microsoft viria a introduzir outra extensão à tecnologia *COM*, o *MTS*¹⁰. Esta, e uma série de outras extensões ao *COM*, foram denominadas de *COM+*¹¹. Esta nova tecnologia resume-se a um conjunto de serviços desenhados para simplificar a criação e uso de componentes de software, possibilitando uma extensa interoperabilidade entre componentes, independentemente da implementação, funcionando como que uma camada superior, permitindo uma interacção melhorada com o *COM*.

As tecnologias introduzidas pela Microsoft apenas funcionam em plataformas desenvolvidas pela mesma, e tendo em conta que os seus sistemas operativos são muito usados, o *COM+* acabou por ser indiscutivelmente mais utilizado. A maior diferença entre *COM* e *CORBA* [STE00, Bos00] diz respeito à natureza competitiva sobre a qual assentam, ou seja, a Microsoft pretendia generalizar o uso dos seus sistemas operativos (como servidores) em ambiente empresarial. O objectivo do *CORBA* seria conceber especificações com o objectivo de criar interoperabilidade e soluções *OO*¹² portáteis e distribuídas independentemente da plataforma. Ao contrário da Microsoft, os fornecedores da tecnologia *CORBA* não controlam um único sistema operativo, daí a necessidade de cobrir múltiplas plataformas.

A tecnologia previamente escolhida para este projecto, o *.NET Remoting*[Cor09h] foi lançada pela Microsoft em 2002, juntamente com a primeira versão do *framework .NET*

⁶*CORBA*: Common Object Requesting Broker Architecture, consultar [OMG09a].

⁷*OMG*: Object Management Group, trata-se de um consórcio, cujo objectivo inicial seria estabelecer especificações para sistemas distribuídos orientados a objectos, actualmente está focada em modelação e especificações baseados em modelos. Consultar [OMG09b]

⁸*COM*: Component Object Model, consultar [Cor09a].

⁹*DCOM*: Distributed Component Object Model, consultar [Cor09c].

¹⁰*MTS*: Microsoft Transaction Server, consultar [Cor09f].

¹¹*COM+*: Component Object Model plus, consultar [Cor09b].

¹²*OO*: Object Oriented.

[Cor09g]. Esta consiste numa *API* que possibilita a comunicação entre processos residentes na mesma máquina, ou máquinas diferentes. Os mecanismos por detrás do funcionamento do *.NET Remoting* são de facto complexos e poderosos. No entanto, a sua utilização em cenários normais é de todo acessível, e até fácil (uma das razões pela qual é usada). Esta característica permite ao programador distanciar-se dos detalhes de baixo nível de interacção entre processos, e dedicar-se mais à aplicação propriamente dita.

Uma outra tecnologia que vale a pena mencionar denomina-se *Java RMI*¹³, e trata-se de uma *API* Java¹⁴ *OO* para a realização de *RPCs*¹⁵, partilhando esta alguma semelhanças com o *CORBA*. Numa primeira versão esta tecnologia apenas podia ser executada em plataformas que possuam uma *JVM*¹⁶, o que constituía uma limitação apesar de ser independente do sistema operativo. No entanto, mais tarde e para eliminar esta limitação, foi lançada uma segunda versão, usando a tecnologia *CORBA*. Comparativamente ao *.NET Remoting* [Woo09] esta tecnologia consome menos recursos e tem um melhor desempenho, mas é significativamente menos flexível e um pouco mais complexa de implementar.

Este tipo de tecnologias, baseadas em componentes, funcionam muito bem em ambientes intranet. No entanto, o seu uso na Internet levanta outras questões e dificuldades. A primeira questão diz respeito à interoperabilidade, ou seja, apesar de todos lidarem com objectos, por vezes não há acordo nos detalhes relativos aos mesmos (por exemplo, ciclo de vida, construtores, herança). A segunda questão diz respeito ao facto do seu estilo *RPC* (de um modo geral) conduzir a sistemas *tyghtly coupled* construídos usando invocação explícita de métodos.

¹³ **Java RMI**: *Java Remote Method Invocation*, consultar [SM09b].

¹⁴ **Java**: linguagem de programação desenvolvida por James Gosling na Sun Microsystems como componente fundamental da *Java platform*. Primeira versão saiu em 1995, consultar [SM09a].

¹⁵ **RPCs**: *Remote Procedure Calls*.

¹⁶ **JVM**: *Java Virtual Machine*, trata-se de um conjunto de programas de software e estruturas de dados que usam um modelo baseados em máquinas virtuais para a execução de outros programas de computador. Consultar [Jav09].

Uma outra tecnologia com funcionalidade semelhante e de extenso uso denomina-se *Web Service* [Gro09], funcionando esta como que uma *API* remota, acessível através da rede (por exemplo a Internet), e executada no servidor remoto. Comparativamente ao *.NET Remoting* [DE02] (e ao *DCOM* e *CORBA*) esta tecnologia é baseada na *Web*, e opera usando o protocolo *HTTP*¹⁷ (e outros protocolos de Internet) para trocar mensagens *SOAP*¹⁸ o que faz com que seja *loosely coupled* e possua elevados índices de interoperabilidade. Apesar das suas características e do seu modo de operação, este tipo de arquitectura pode ser usado em todo o tipo de aplicações (baseadas na *Web* ou não). Os *Web Services* baseiam-se em standards (*HTTP*, *XML*¹⁹, *WSDL*²⁰, *SOAP*) para expor o seu modo de funcionamento e são independentes da linguagem de programação, plataforma e dispositivo. No entanto, acabam por perder um pouco em performance e funcionalidade face às restantes tecnologias.

¹⁷*HTTP*: Hypertext Transfer Protocol.

¹⁸*SOAP*: Simple Object Access Protocol, é uma especificação protocolar para a troca de informação estruturada na implementação de *Web Services* em rede de computadores.

¹⁹*XML*: Extensible Markup Language.

²⁰*WSDL*: Web Services Description Language. Linguagem baseada em *XML* que fornece um modelo para descrever *Web Services*.

Relativamente ao ambiente de utilização da aplicação há que analisar os seguintes parâmetros:

- As máquinas estão ligadas em ambiente *LAN* (ou seja intranet).
- Não está previsto que estejam mais de cinquenta máquinas na rede.
- Partilha da mesma plataforma desenvolvida pela Microsoft (Windows XP).
- Familiaridade dos futuros programadores com tecnologia *.NET*.
- São conhecidas as configurações de todos os computadores.
- Configurações de software e *hardware* semelhantes.
- Possibilidade de privilégios administrativos em todas as máquinas, sendo portanto possível ajustar configurações se necessário.
- Numa situação futura há possibilidade de utilização, em *cluster* computacional ("mais" *tightly coupled* que o actual).

Pode-se, então, concluir que, de entre todas as tecnologias que foram estudadas, a escolha iria recair sobre uma tecnologia *OO* adaptada o melhor possível a plataformas Microsoft, que fosse o mais poderosa e flexível possível e, ao mesmo tempo, simples de utilizar, de modo a que fosse possível ao programador distanciar-se dos detalhes de baixo nível, e focar-se no que é realmente importante. Posto isto, podemos concluir que a tecnologia *.NET Remoting* é de facto a mais indicada para o cenário de utilização pretendido.

3.3 Projectos relacionados

Nos laboratórios da Universidade de Melbourne (Austrália) foi recentemente desenvolvido um projecto denominado "Alchemi"[Aks09], consistindo este numa *framework* baseada em .NET para o desenvolvimento de software para *grid Computing*. Esta permite agregar o poder computacional de várias máquinas distribuídas pela rede. Após uma análise cuidada da informação relativa ao projecto, foi possível constatar que este possui algumas semelhanças com o projecto aqui descrito:

- **Escolhas tecnológicas.**
- **Paradigma Gestor-Trabalhador:** a ideia de ter apenas um gestor que aloca tarefas aos diversos trabalhadores.
- **Arquitectura:** a nível da composição, possui o mesmo tipo de componentes (gestor, trabalhador, cliente). A nível da interacção é semelhante na medida em que, não há comunicação directa entre o cliente (que faz o pedido) e o trabalhador (que executa o pedido), ou seja, toda a informação é serializada e enviada ao gestor, que de seguida a reencaminha para o trabalhador e após conclusão da tarefa, é realizado o caminho inverso.
- **Configuração:** gestor único, múltiplos executores e múltiplos clientes.
- **Ciclo de execução:** sequência de interacção entre os nós que compõem a rede semelhante.

Por outro lado, possui algumas diferenças, estando estas relacionadas com a escalabilidade e interoperabilidade. Este projecto usa um *clustering* baseado na *Web* para deste modo, ser facilmente escalável, permitindo a interligação e hierarquização de *clusters* através da Internet. Possui, ainda, uma interface de *Web Services* genérica para possibilitar a execução de tarefas, ainda que vindas do exterior, tornando-se assim independente da plataforma. Os motivos da existência destas diferenças estão relacionados com o ambiente em que será usada a *framework* (*LAN* vs *WAN*²¹.) e a flexibilidade que será necessário atribuir à *framework* (não faz parte dos objectivos iniciais a interoperabilidade com software externo).

Um outro projecto recente e relacionado denomina-se "DotGrid"[Ali09], consistindo este numa *API* para a implementação de aplicativos e *middleware* de suporte ao *grid computing*. Este projecto é significativamente diferente, residindo as únicas semelhanças na tecnologia *.NET* e no conceito (processamento paralelo e distribuído). Mesmo comparativamente com o Alchemi, este projecto parece possuir uma implementação mais elaborada (mais baixo nível, e sem usar *.NET Remoting*), o que leva a outros índices de performance. A distribuição de informação é feita de um modo mais aperfeiçoado, ao invés de serializar e passar a informação de nó em nó. Possui mecanismos de segurança mais sofisticados recorrendo inclusivé ao uso de criptografia. Estas e outras características fazem com que este projecto seja mais elaborado e aperfeiçoado que o projecto que o presente relatório descreve.

²¹ **WAN**: *Wide Area Network*, consultar [Onl09i]

3.4 Análise tecnológica (estudo do .NET Remoting)

Quando se procede à concepção de uma possível arquitectura, é necessário tomar diversas decisões. Uma vez que as tecnologias a serem usadas, para a concretização do projecto, se baseiam em *.NET*, resta determinar em concreto o que usar, e de que forma, procurando maximizar a relação entre facilidade/simplicidade e qualidade. O objectivo ao conceber a arquitectura será potenciar a estabilidade, segurança, transparência, concorrência, tolerância a falhas e desempenho da *framework*, bem como a sua facilidade de utilização, em detrimento da escalabilidade (irá operar apenas em ambiente *LAN*), heterogeneidade (assenta em tecnologias Microsoft), e interoperabilidade (não está prevista qualquer interacção com software externos).

As escolhas relativas à análise tecnológica são as que se seguem.

3.4.1 Tipos de remoting

Existem dois tipos de interacção remota entre cliente e servidor. Um deles usa o que se designa por "*ByValue Objects*"²², tratando-se estes, de uma serialização de objectos que são passados como cópia ao processo remoto. Qualquer método invocado sobre estes objectos será executado localmente, tendo o cliente de possuir a classe compilada, contendo a sua definição. O outro tipo de interacção usa os chamados "*MarshalByRefObjects*"²³, que residem sempre no servidor e aceitam invocações remotas por parte dos clientes. Para possibilitar a interacção o cliente detém uma referência para o objecto, no entanto não se trata de uma referência comum, pois em vez de conter um endereço de memória, contém o nome ou endereço de *IP*²⁴, que identifica não só o servidor como a informação relativa ao objecto específico, para que este possa ser distinguido dos restantes que estão alojados no servidor. Este segundo método adequa-se melhor ao cenário pretendido, pois deste modo, qualquer alteração é de imediato propagada para quem detém referências do mesmo, não sendo necessário ao cliente ter a definição da classe do objecto. Além disso, como se disse, o processamento é remoto e não local.

²²Consultar [RS02], pags. 25 e 26.

²³Consultar [RS02], pags. 26 a 46.

²⁴*IP*: Internet Protocol, consultar [Onl09h].

3.4.2 Tipos de activação

Existem dois tipos de activação dos objectos remotos: a primeira denomina-se "SAO"²⁵ e a segunda "CAO"²⁶. A diferença entre os dois reside na iniciativa de instanciação. No primeiro caso o cliente pede uma referência para o objecto remoto e a instanciação fica a cargo do servidor. Caso o servidor esteja configurado como *singleton*²⁷, irá fornecer uma única referência para todos os clientes que a pediram, sendo esta destruída apenas quando o servidor terminar. Se estiver configurado como *singlecall*²⁸ será criada uma referência nova para cada invocação e destruída assim que a execução terminar. É importante referir que o servidor poderá instanciar objectos *singleton* por iniciativa própria, que poderão ser acedidos pelos clientes, mediante configuração adequada. Nos CAO os objectos são instanciados no servidor por iniciativa do cliente, tendo este que guardar as respectivas referências para aceder aos objectos, ou seja o servidor poderá possuir vários objectos instanciados por iniciativa dos clientes, e poderá passar as referências a qualquer cliente²⁹.

Relativamente aos tipos de activação a solução escolhida será misturar SAO com CAO e usar o "Factory design pattern"³⁰. A interacção é a seguinte: no servidor reside um objecto instanciado pelo próprio (também conhecido como "Published Object"³¹.) e nesta instância irá residir informação de gestão e de todos os clientes que se encontram a interagir com o sistema. A mesma referência será enviada a todos os clientes, assim que estes invocarem métodos sobre o objecto remoto *singleton*, residente no servidor. De entre os métodos disponibilizados, irá haver um que após um processo de autenticação bem sucedido, fornecerá uma nova instância de um objecto representativo da ligação entre cliente e servidor. A partir deste momento, qualquer interacção entre cliente e servidor será feita através deste objecto, que representa como que o "login" do cliente.

²⁵SAO: *Server Activated Objects*, consultar [RS02], pags. 26 e 27.

²⁶CAO: *Client Activated Objects*, consultar [RS02], pags. 34 a 43.

²⁷Observar cenário 1 na figura 3.1, consultar [RS02], pag. 30.

²⁸Observar cenário 2 na figura 3.1, consultar [RS02], pags. 28,29 e 30.

²⁹Observar cenário 3 na figura 3.1.

³⁰Consultar [RS02], pags. 38 a 43.

³¹Consultar [RS02], pags. 32 e 33

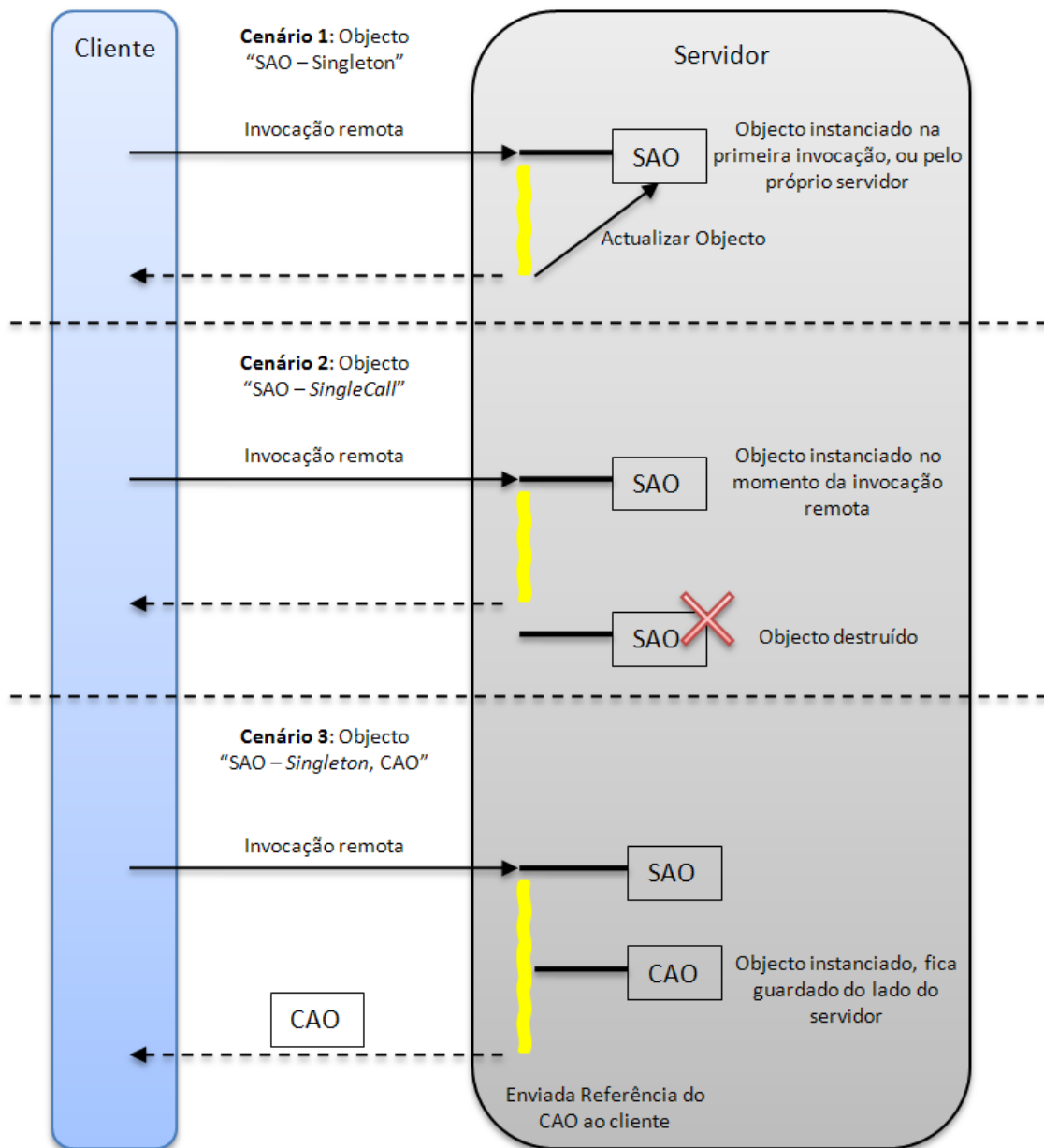


Figura 3.1: Tipos de activação

3.4.3 Tipos de invocação

Existem três tipos de invocação remota e todos elas possuem potencial para serem utilizadas, em situações específicas. O tipo mais utilizado será sem dúvida as chamadas síncronas ("Synchronous Calls")³². O cliente ao fazer este tipo de invocação bloqueia à espera do retorno e, caso ocorra alguma exceção do lado do servidor, esta é propagada para o cliente. O segundo tipo pode ser efectivamente útil se o cliente não quiser bloquear enquanto espera o retorno de uma invocação. Por este motivo a invocação assíncrona ("Asynchronous Calls")³³ comporta duas fases. O primeiro passo desencadeia o processo de execução, não bloqueando à espera do retorno ou recebendo qualquer exceção. Para verificar se a função já retornou, é necessário chamar outro método que verifica se o servidor já terminou de processar o pedido, ficando este método à espera do termo da execução, recebendo neste momento qualquer exceção, caso esta ocorra ou tenha ocorrido. O terceiro tipo de invocação denomina-se "One-Way Call"³⁴ e é diferente dos casos anteriores, não podendo ter parâmetros de saída ou valores de retorno. A *framework .NET* não garante a execução dos métodos, nem qualquer informação posterior à invocação. Na eventualidade do servidor estar indisponível, o cliente não é notificado. Este tipo de invocação não é muito desejável para a maioria das situações, mas poderá ter utilidade por exemplo, na escrita de *logs* de importância reduzida.

³²Observar cenário 1 na figura 3.2, consultar [RS02], pags. 47 a 51.

³³Observar cenário 2 na figura 3.2, consultar [RS02], pags. 51 a 55.

³⁴Observar cenário 3 na figura 3.2, consultar [RS02], pags. 55 a 58.

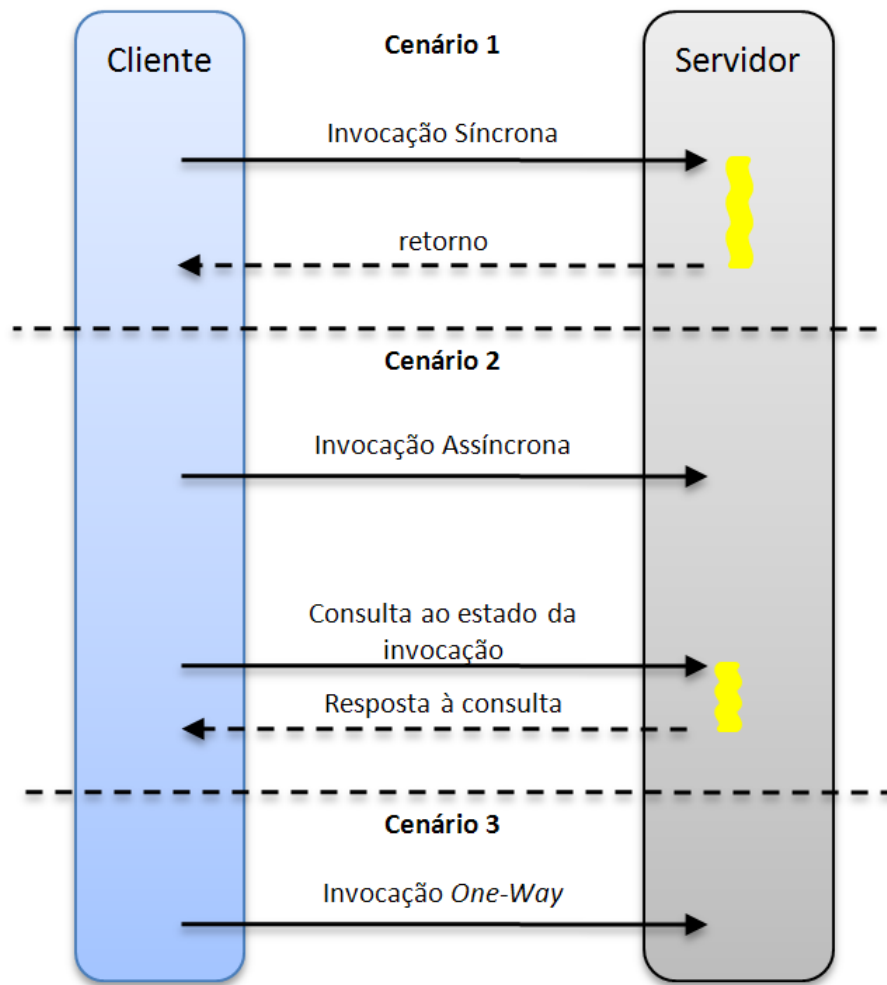


Figura 3.2: Tipos de invocação

No cenário da aplicação a desenvolver estamos perante funções de processamento pesado, onde é inevitável esperar pelo resultado das invocações, sem que a aplicação do cliente bloqueie. A solução poderia ser realizar a invocação de forma síncrona (ou colocar apenas a segunda fase da invocação assíncrona) numa execução (*thread*³⁵) à parte. No entanto, a melhor solução será a função retornar de imediato, e posteriormente ser o próprio servidor a notificar o cliente, evitando assim a existência de funções "pendentes".

³⁵*thread*: significa "fio" de execução. Um mesmo processo pode possuir várias *threads*, ou seja, pode possuir vários procedimentos a executar em simultâneo.

3.4.4 Partilha de *assemblies*

Nas aplicações *.NET Remoting* os servidores necessitam de partilhar com os clientes informação relativa aos serviços disponibilizados, e para o conseguirem existem diversos métodos.

Podem partilhar a implementação³⁶, ou seja, colocar a implementação do servidor no *assembly*³⁷ partilhado com o cliente. Embora esta opção fosse simples, não é com certeza das melhores, pois para além de violar os princípios de desenvolvimento de aplicações distribuídas, torna a implementação vulnerável a terceiros. Esta solução pode ser desejável na situação cujo objectivo seja alternar facilmente entre o panorama de execução local e distribuído, ou seja, podemos facilmente criar objectos dos tipos definidos no *assembly* partilhado, procedendo a invocações locais.

Um outro método será partilhar apenas as interfaces³⁸, que serão implementadas pelo servidor e usadas pelo cliente na invocação remota de métodos.

Ao invés de partilhar interfaces, poder-se-á especificar classes abstractas³⁹ no *assembly* partilhado. O objecto remoto alojado no servidor irá herdar os métodos da classe abstracta e implementar as funcionalidades que serão usadas pelo cliente. A desvantagem desta abordagem é a "poluição" da hierarquia de herança, com demasiados protocolos e construções específicas.

Uma outra possibilidade será utilizar a ferramenta "*SoapSuds.exe*", fornecida pela Microsoft, para extrair os meta dados⁴⁰ de um dado tipo, alojado num servidor em funcionamento, ou num *assembly*. A partir deste ponto, gera-se um novo *assembly* que irá apenas conter a meta informação extraída. Para que o cliente possa interagir com o servidor basta referenciar o *assembly* gerado. Com esta solução deixa de ser necessário usar *assemblies* intermédios. No entanto, é uma técnica antiga, pouco usada e mais complexa que as anteriores, não havendo propriamente necessidade de a usar.

Na fase de implementação serão usadas duas técnicas: de partilha de implementações e de interfaces. Tendo em conta que o *assembly* principal será partilhado por todos, o objectivo será "despi-lo" o mais possível de implementações desnecessárias, privilegiando a partilha de interfaces. Com esta metodologia consegue-se partilhar de forma eficiente os métodos necessários, mantendo a implementação do lado do servidor e escondida face ao cliente. Este cenário pode ser observado na figura 3.3.

³⁶Consultar [RS02], pag. 67.

³⁷**assembly**: mais concretamente *.NET Assembly*. Na *framework .NET* da Microsoft, um *assembly* é uma biblioteca de código pré-compilado, cujo objectivo é facilitar as actividades de instalação, segurança e controle de versões.

³⁸Consultar [RS02], pag. 67.

³⁹Consultar [RS02], pag. 67.

⁴⁰Consultar [RS02], pags. 68 a 73.

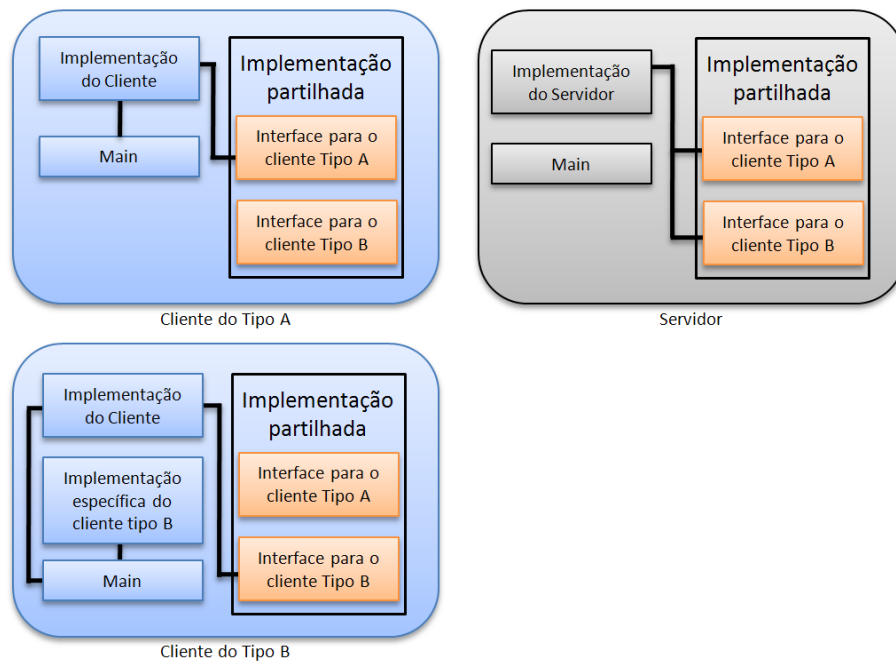


Figura 3.3: Cenário de partilha de *assemblies* e interfaces

3.4.5 Escolha do protocolo

Ao escolher o protocolo há que ter em consideração a localização, configuração das entidades comunicantes e o ambiente no qual estas vão operar[dn09]. A aplicação a desenvolver vai ser instalada em ambiente *LAN*, logo todas as entidades estão fisicamente acessíveis, e como tal, são facilmente configuráveis. Tendo em conta estas características, a solução mais eficiente consiste em usar o protocolo *TCP*⁴¹ directamente, ao invés do protocolo *HTTP*. Usar o protocolo *TCP* beneficia a eficiência e prejudica a escalabilidade. Caso se pretendesse expandir o sistema para além do ambiente *LAN*, o uso do protocolo *TCP* revelava-se um problema, devido ao facto de entre as extremidades comunicantes existirem normalmente *firewalls* ou *routers* que rejeitam tráfego, não originário ou destinado à porta 80.

⁴¹*TCP*: *Transmission Control Protocol*, consultar [Onl09h].

3.4.6 Configurações multi-servidor

Este tipo de configurações ⁴² pode ter vários fins, por exemplo para garantir disponibilidade ou qualidade de serviço. Nesta situação irão existir vários servidores a alojar o mesmo tipo de objecto, podendo o cliente escolher o melhor. Este tipo de configuração pode ainda servir para balancear a carga entre servidores, na medida em que cada um disponibiliza um serviço diferente. Poderia, por exemplo, existir um servidor a alojar um objecto *singleton* e durante as invocações remotas por parte dos clientes iria recorrer a outro servidor que estaria a disponibilizar um serviço do tipo *singlecall*. Para o cenário da aplicação a desenvolver, o servidor apenas se irá encarregar de encaminhar os pedidos dos clientes para outros clientes, responsáveis pelo processamento, e realizar o fluxo inverso na recolha de resultados. Tendo em conta que o processamento pesado apenas se irá realizar nos clientes, e que não haverá muitos clientes na rede, não há necessidade de existirem vários servidores. Uma das possibilidades de existência de vários servidores, pode ser a de interligação de vários *clusters*, na situação onde o servidor é tratado como um cliente de processamento face a outro servidor (ver *figura 3.4*).

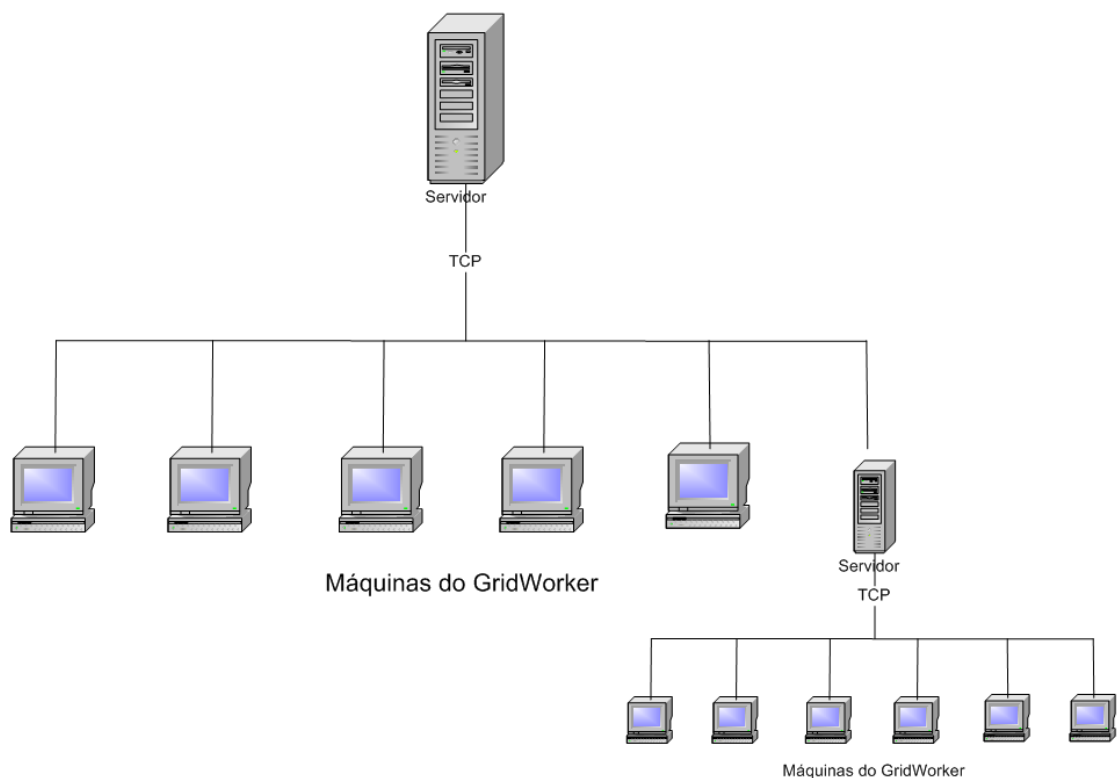


Figura 3.4: Cenário multi-servidor

⁴²Consultar [RS02], pags. 59 a 66.

3.4.7 Configuração do .NET Remoting

Para que seja possível a comunicação entre os diversos componentes de *.NET Remoting* é necessário uma configuração prévia para acordar endereços de acesso, protocolos de comunicação e configurações de segurança. A vantagem de usar ficheiros de configuração⁴³, face à configuração programática, é tornar possível a alteração dos parâmetros de configuração sem recompilar o código. A desvantagem é que sempre que se iniciar a aplicação, o ficheiro de configuração terá de estar presente. No caso do serviço *.NET Remoting* ser alojado no *IIS*⁴⁴, o ficheiro constitui o único modo de configuração.

Para se proceder à configuração da aplicação que será desenvolvida, irá usar-se uma espécie de método híbrido. Internamente, a ideia será usar configuração programática, para que tanto o programador que irá usar a *framework*, como o responsável pela instalação dos restantes componentes, não necessitem de saber pormenores de configuração usados. No entanto, terão sempre de existir ficheiros de configuração específicos contendo apenas os campos que necessitam mesmo de ser parametrizáveis, como por exemplo configurações de acesso ao servidor e configurações de segurança. No caso do servidor, este terá de ter pelo menos o par endereço/porta (que irá escutar enquanto estiver activo), e a lista de clientes que irão aceder ao servidor. O ficheiro de configuração dos clientes terá de ter, pelo menos as configurações de acesso ao servidor, e a respectiva identidade para permitir a autenticação.

⁴³Consultar [Cor09d] e [RS02], pags. 76 e 85 a 100.

⁴⁴*IIS*: *Internet Information Services*, também chamado de *Internet Information Server*, trata-se de um conjunto de serviços baseados em Internet destinado a servidores. Foi desenvolvido pela Microsoft para usar juntamente com os seus sistemas operativos. Actualmente é o segundo mais popular servidor *Web*. Consultar [Cor09e].

3.4.8 Instalação

Existem três modos de proceder à instalação da aplicação. A primeira consiste em associar a aplicação a uma *"Console application"*⁴⁵. Este tipo tem como vantagens a fácil visualização de saídas, iniciar e terminar o programa bem como *debug* directo usando o *IDE*⁴⁶ (*VS*⁴⁷ 2008), o que corresponde ao cenário mais fácil, para quem pretende testar algumas tecnologias. No entanto, o ambiente de desenvolvimento difere do ambiente de utilização final, não sendo razoável a aplicação ter de ser iniciada cada vez que o sistema é iniciado. De forma, a resolver uma destas questões pode-se utilizar os chamados *"Windows Services"*⁴⁸. Este é um serviço que é iniciado com o sistema operativo, o que faz com que deixe de ser necessário um iniciar explícito. No entanto, o *debug* e a recolha de saídas constituem uma dificuldade. Para resolver este problema, basta definir o arranque do programa como sendo híbrido, bastando para isso definir um argumento de entrada no método *"main()"*, que será responsável por definir o programa como sendo uma *"Console Application"* ou *"Windows Service"*. Tal como no *"Windows Service"*, ao alojar os componentes no *IIS*⁴⁹, o *debugging*⁵⁰ e a recolha de saídas podem revelar-se uma dificuldade. Mas, por outro lado, resolve todos problemas da *"Console Application"*, pois para além de arrancar com o sistema, possui mecanismos integrados que facilitam a autenticação e o uso de canais seguros. Devido a estas características, o *IIS* é o indicado por especialistas como a melhor solução de alojamento. Sem dúvida que para uma aplicação empresarial, que disponibiliza serviços a um leque alargado de clientes num ambiente *WAN*, se trata da escolha acertada. Mas tendo em conta que o ambiente de utilização da aplicação a desenvolver será uma *LAN* e que o *IIS* só usa o protocolo *HTTP* (o que conduziria a uma perda de eficiência, e impediria o uso de eventos), a solução de alojamento escolhida será o *"Windows Service"* para uma utilização final. Assim, sabendo que a conversão de *"Console Application"* para *"Windows Service"* é acessível, a ideia será usar uma *"Console Application"* durante a fase de desenvolvimento e demonstração.

3.4.9 Segurança

Desde a versão 2.0 da *framework .NET*⁵¹ que a infra-estrutura de segurança é parte integrante da *framework*. Para garantir a segurança da comunicação basta acrescentar umas propriedades à definição dos canais de comunicação. Embora em ambiente *LAN*, as

⁴⁵Consultar [RS02], pag. 108.

⁴⁶*IDE: Integrated Development Environment*, também conhecido como *"integrated design environment"* ou *"integrated debugging environment"*.

⁴⁷*VS: Microsoft Visual Studio*, trata-se de um *IDE* desenvolvido pela Microsoft.

⁴⁸Consultar [RS02], pags. 108 a 115.

⁴⁹Consultar [RS02], pags. 116 a 120.

⁵⁰Processo metódico de procura e redução do número de *bugs* ou defeitos num programa de *software*, transformando-o de modo a que o seu comportamento corresponda ao pretendido.

⁵¹Consultar [RS02], pags. 151-160.

preocupações de segurança não sejam as mesmas comparativamente a um ambiente *WAN*, a segurança não pode ser negligenciada. Muito simplificada, a ideia será configurar os canais de comunicação para que usem a infra-estrutura de segurança disponibilizada pela *framework .NET*, garantindo assim o acordo de protocolos e a consequente encriptação da informação. Relativamente à autenticação, o servidor irá manter uma lista de pares de *usernames* e *passwords* (encriptadas usando *MD5*⁵²), correspondendo esta à lista de utilizadores cujo acesso é autorizado. Para se autenticar o utilizador terá de conhecer o seu *username* e *password* (formando estes dois elementos a sua identidade). De seguida esta informação será passada ao servidor devidamente escondida através do "*Call Context*"⁵³, que ficará encarregue de a validar, autorizando ou não o acesso aos serviços por ele disponibilizados.

⁵²**MD5**: *Message-Digest algorithm 5*, trata-se de uma função de *hash* criptográfica amplamente usada. Uma função de *hash* por sua vez é um procedimento determinístico que transforma um bloco arbitrário de informação numa *string* de *bits* de tamanho fixo, correspondendo isto ao valor de *hash* do bloco de informação inicial. Consultar [WC309].

⁵³Consultar [Cor09j] e [RS02], pags. 209 a 212.

3.4.10 Notificações

Os eventos⁵⁴ constituem um método do tipo *publish-subscribe* para uma iniciativa de comunicações em sentido inverso ao "normal", ou seja, do servidor para o cliente⁵⁵. Este tipo de fluxo de informação torna-se relevante quando o servidor necessita de notificar o cliente. Para o fazer o servidor necessita de manter uma variável que irá guardar os clientes que a inscreveram (associaram um *handler*⁵⁶) ao respectivo evento, e quando for oportuno basta invocar (publicar) a variável do tipo *event* com os respectivos argumentos, que todos os subscritores serão notificados (o *handler* de cada um será chamado). Esta estratégia de notificação não deve ser usada em ambientes WAN, pois as latências da rede, o número elevado de clientes, o seu estado actual (é desconhecido, podendo estar indisponíveis) e configuração (*firewall* ou portas de acesso) levam a que seja inviável. Este tipo de solução deve apenas ser empregue para comunicação entre aplicações residentes na mesma máquina e, se bem definida, é possível estender esta solução a um ambiente LAN condicionado, ou seja, onde o número de máquinas não é elevado, e onde o estado e configuração de todas as máquinas é conhecido⁵⁷. Existem alternativas para a notificação embora não sejam tão adequadas ao cenário pretendido. Poder-se-ia, por exemplo, usar *UDP*⁵⁸, no entanto esta alternativa para além de ser assíncrona, não garante ordem nas notificações, não é fiável (não garante que os pacotes sejam entregues), e cada pacote enviado pode ser recebido várias vezes (um por cada interface de rede). Esta alternativa obrigaria a uma série de mecanismos auxiliares para garantir o bom funcionamento. Uma outra alternativa que corrige alguns aspectos do *UDP* é o *MSMQ*⁵⁹, que para além de garantir a ordem, e ser fiável, não recebe mensagens repetidas. O facto de ser assíncrona obriga a que o cliente possua mecanismos auxiliares para consultar periodicamente a fila de mensagens ou que configure os "triggers" associados à recepção de mensagens (funcionalidade associada ao serviço de *MSMQ*). Embora este serviço faça parte do sistema operativo, não é instalado por defeito, o que conduz a uma instalação extra, caso se pretenda usar.

⁵⁴Consultar [RS02], pags. 213 a 224.

⁵⁵Consultar cenário 1 na figura 3.5.

⁵⁶No contexto da programação, diz respeito ao procedimento específico para lidar com uma dada situação, como por exemplo o despoletar de um evento.

⁵⁷Consultar [RS02], pag. 277.

⁵⁸*UDP: User Datagram Protocol*, trata-se de um dos protocolos principais pertencentes ao conjunto de protocolos usados na Internet. Consultar cenário 3 na figura 3.5.

⁵⁹*MSMQ: Microsoft Message Queuing*, trata-se de uma implementação de sistema de filas de mensagens desenvolvido pela Microsoft e disponível em grande parte dos seus sistemas operativos. Consultar [Pet09], e [RS02], pags. 284-286 e ver ainda cenário 3 na figura 3.5

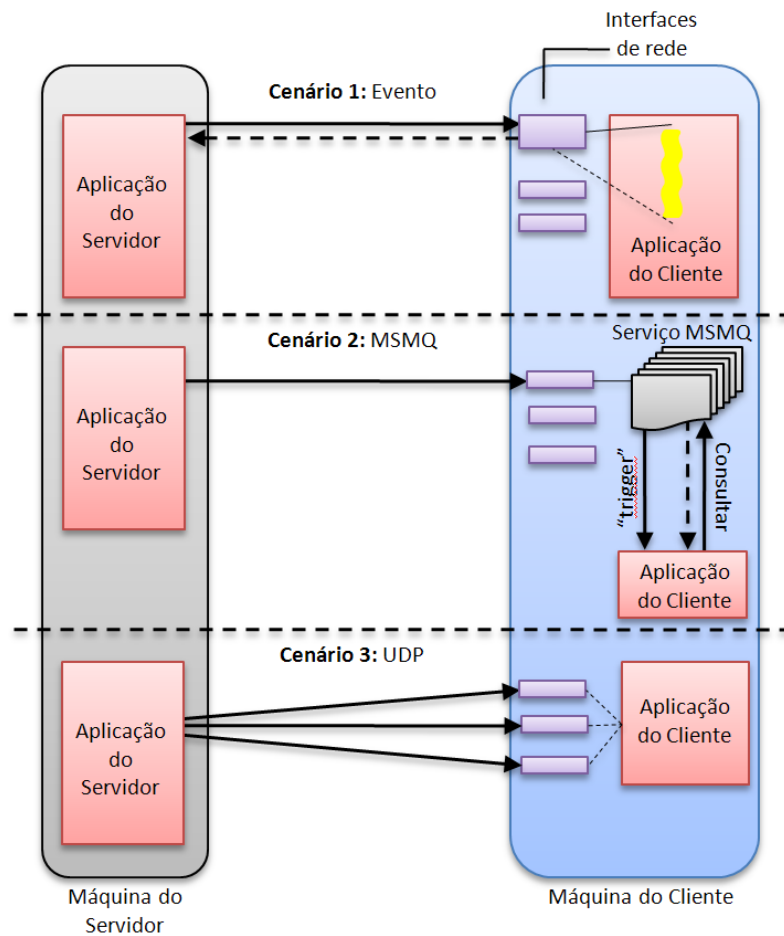


Figura 3.5: Diferentes cenários de notificação

Tendo em conta que o ambiente da aplicação a desenvolver se trata de um *cluster* de computadores ligados entre si através de uma rede *LAN*, e onde o sincronismo e a eficiência assumem um papel mais importante face à escalabilidade, usar eventos como forma de notificação corresponde à melhor alternativa.

3.4.11 *Assemblies*

Para que seja possível decifrar o conteúdo dos *assemblies*, será usado um método de reflexão incluído na *framework .NET* (denominada de "reflection"⁶⁰), que permite efetuar interrogações acerca da estrutura e conteúdos próprios. Esta técnica será usada para determinar quais os *assemblies* (e os seus dependentes) que estão definidos do lado do cliente, e que necessitam de ser enviados ao servidor quando da submissão de pedidos. Serve ainda para que os clientes executores consigam invocar métodos de um *assembly* cujo conteúdo é apenas conhecido em tempo de execução.

3.4.12 *Transferência de assemblies*

Para que seja possível ao cliente submeter o seu código no servidor, e para que este seja executado por terceiros, é necessário escolher um modo de transferir os *assemblies* de um lado para o outro. Em primeiro lugar, será necessário converter em *bytes* o respectivo *assembly*, para que seja possível o seu envio através da rede. A partir deste ponto, resta escolher o modo de transporte de uma extremidade comunicante para a outra. Tendo em conta que já existe um canal de comunicação estabelecido em *.NET Remoting*, a ideia será usá-lo para a transferência de *assemblies*. O servidor após receber o *assembly* ficará encarregue de o distribuir, pelos demais clientes executores. O executor terá, então, de carregar o respectivo *assembly* quando da execução de métodos que com ele estejam relacionados.

⁶⁰Consultar [[Ric02](#)] e [[Cor09i](#)].

3.4.13 Comunicação entre domínios aplicativos e injeção de *assemblies*

A necessidade de criação de um *AppDomain*⁶¹ diferente surge da necessidade de carregar e descarregar os *assemblies*. Infelizmente, a *framework* não permite descarregar um *assembly*, ou seja, para descarregar um *assembly*, o que de facto é preciso fazer é descarregar o *AppDomain* onde esse *assembly* foi carregado. O procedimento consiste em criar um novo *AppDomain*, injectar o *assembly* (ou *assemblies*), instanciar e obter uma referência para um dado objecto definido no *assembly*, invocar os métodos necessários e recolher os resultados. No final, será necessário descarregar o *AppDomain*.

Um dos problemas encontrados foi que, ao obter a referência de um objecto instanciado num segundo *AppDomain*, o *assembly* que contém a classe que define esse mesmo objecto era igualmente carregado no *AppDomain* principal. A forma de resolver esta questão consistiu em usar um *assembly* auxiliar, que fosse sempre carregado em ambos os *AppDomains*, e seria este *assembly* auxiliar que ficaria encarregue de executar métodos de outros *assemblies*. Desta forma, o *AppDomain* principal não lida directamente com os restantes *assemblies*.

⁶¹*AppDomain*: *application domain*. No contexto da *framework .NET*, trata-se de mecanismo (similar a um processo no contexto dos sistemas operativos) usado para isolar a execução de aplicações de *software*, de modo a que não se afectem umas às outras.

3.4.14 Gestão do tempo de vida dos objectos

Qualquer objecto do tipo *CAO* residente no servidor, tem de possuir um tempo de vida associado, sendo tal característica dependente da gestão do ciclo de vida⁶². Se tal gestão não existisse, todos os objectos residentes no servidor representativos do registo dos clientes iriam ficar a ocupar espaço de memória no servidor por tempo indeterminado, o que conduziria a longo prazo ao esgotamento da mesma.

Cada aplicação que funcione como servidor em *.NET Remoting* possui a correr, em segundo plano, uma outra aplicação denominada "*Lease Manager*"⁶³, cuja função é verificar o tempo de vida (*TTL*⁶⁴) de cada objecto residente no servidor. Se o tempo de vida do objecto tiver expirado, este é removido do servidor. De forma a controlar o tempo de vida do objecto, evitando que este seja precocemente removido, é necessário definir o tempo de vida do objecto como infinito (se essa for a solução mais indicada), ou implementar um mecanismo de renovação do tempo de vida do objecto. A primeira hipótese reside na implementação de um mecanismo baseado em *sponsors*⁶⁵. Muito simplificada, um *sponsor* é um objecto residente no servidor (possuindo também este um tempo de vida) que poderá ter a ele associado vários objectos pelo qual é responsável. Cada objecto remoto apenas pode estar associado a um *sponsor* mas este pode ter vários objectos dele dependentes. Usando *sponsors* a gestão do ciclo de vida é feita da seguinte forma: assim que o *Lease Manager* detecta que o *TTL* de um dado objecto expirou, este verifica se o objecto possui algum *sponsor*, e em caso afirmativo, caberá ao mesmo decidir se expande, ou não, o tempo de vida do objecto, e por quanto tempo.

⁶²Consultar [RS02], pags. 185 a 209.

⁶³Consultar [RS02], pags. 186 e 187.

⁶⁴*TTL: Time To Live*, é um limite expresso num período de tempo ou numero de iterações ou transmissões (no contexto das tecnologias das redes de computadores ou computadores) que uma dada unidade de informação ou entidade (computacional) pode experienciar antes de ser descartada.

⁶⁵Consultar [RS02], pags. 196 a 209 e cenário 1 na figura 3.6.

Posto isto, a ideia seria definir um *sponsor* para os clientes e outro para os executores, e à medida que fossem expirando, o respectivo *sponsor* era contactado, o seu registo era analisado, e um novo tempo de vida restante seria retornado. O problema foi que apesar de se ter implementado um mecanismo para criar distinção entre os vários objectos residentes no servidor, assim que um *sponsor* renovava um tempo de vida, o tempo de vida de todos os objectos que dele dependiam eram renovados. Tal seria de todo indesejável, pois iria conduzir à situação em que seriam renovados tempos de vida de objectos já expirados. Analisando a situação, chegou-se à conclusão que seria necessário criar um *sponsor* distinto para cada objecto⁶⁶, embora tal seja uma solução pouco viável, apesar de existirem poucos objectos em simultâneo. Tendo em conta as dificuldades encontradas, na implementação dos mecanismos de gestão do tempo de vida, usando *sponsors*, a alternativa foi implementar um mecanismo de renovação explícito do tempo de vida⁶⁷.

De referir que, faz parte da configuração do servidor, um parâmetro que define o tempo de vida dos objectos representativos do registo do cliente. Este parâmetro é passado ao cliente, no momento do registo, para que este consiga determinar de quanto em quanto tempo deve proceder a uma renovação explícita. Assim, evita-se que o objecto residente no servidor, correspondente ao seu registo, expire precocemente.

⁶⁶Consultar cenário 2 na figura 3.6.

⁶⁷Consultar cenário 3 na figura 3.6.

Estado da Arte

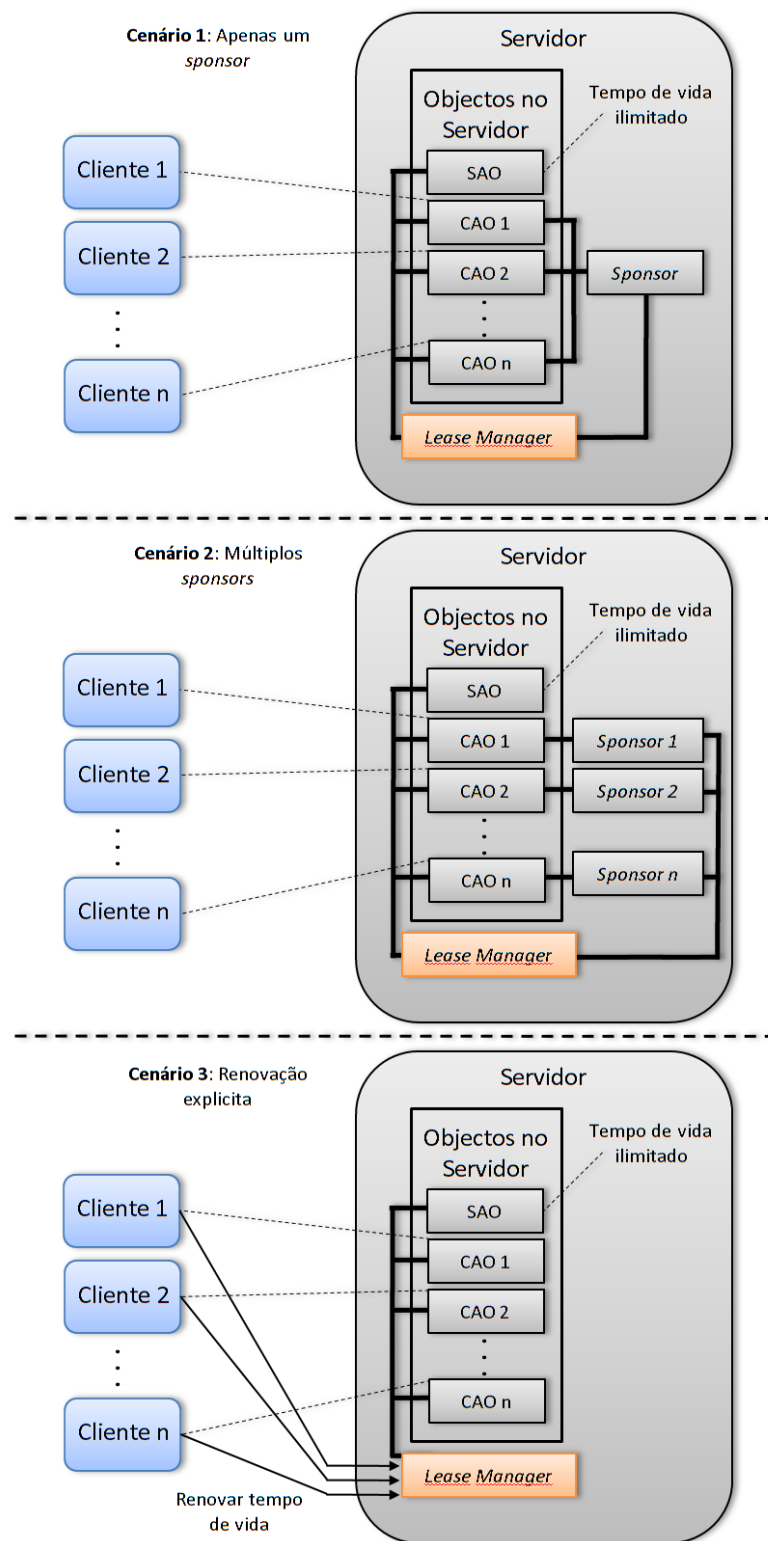


Figura 3.6: Diferentes cenários de gestão do ciclo de vida

Capítulo 4

Arquitectura

Até este ponto foi possível compreender um pouco mais acerca deste projecto. Desde um estudo do conceito em que este se insere a projectos que com ele estão relacionados, bem como uma análise tecnológica detalhada. Tendo como base descrições iniciais, requisitos propostos e escolhas tecnológicas foi elaborada a arquitectura do projecto. Esta será explicitada recorrendo a diagramas complementados com exposições sucintas dos pontos julgados pertinentes.

4.1 Arquitectura física

O cenário físico em que os diversos componentes da aplicação vão ser instalados é composto por três tipos de elementos ligados entre si em ambiente *LAN* usando o protocolo *TCP* (consultar figura 4.1). Irá existir um servidor central "GridMaster"¹ que deverá atender aos pedidos dos clientes e enviar notificações (quando necessário) para os restantes elementos da rede. Para além do servidor existem os clientes (pelo menos um) responsáveis pelo processamento e denominados "GridWorkers", ficando estes encargos da execução de tarefas. Estes dois tipos de elementos constituem a grelha ("Grid") computacional, cujos recursos estão ao dispor do programador mediante o uso de uma *API*.

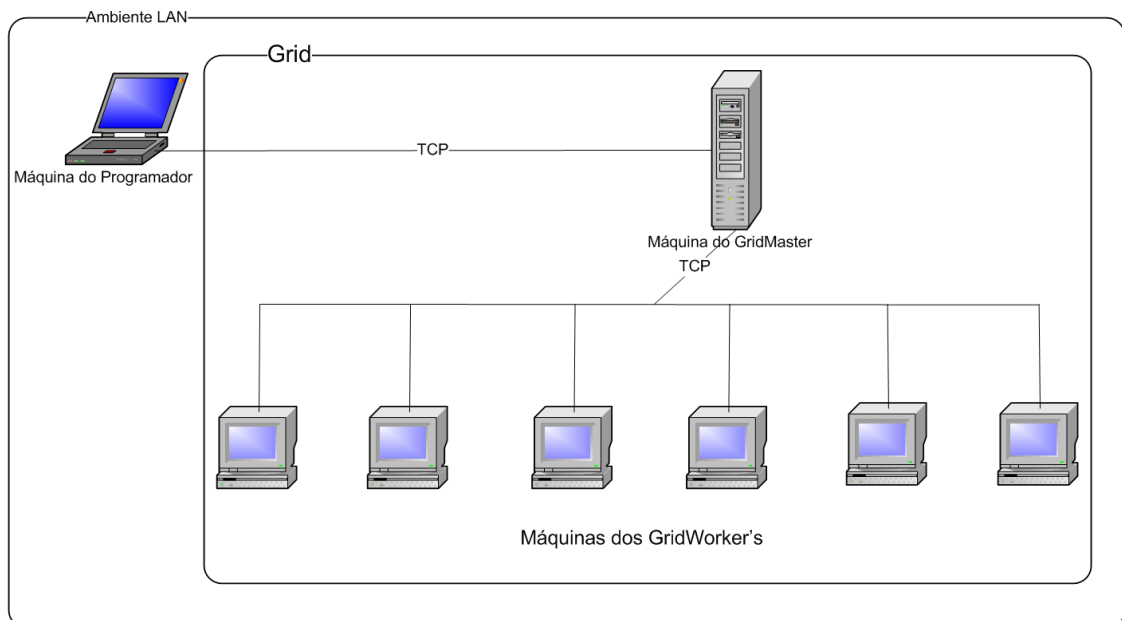


Figura 4.1: Arquitectura física do projecto

¹É importante referir que as entidades que representam os componentes do sistema, apesar de se tratarem estrangeirismos, não vão aparecer em itálico

4.2 Arquitectura lógica

A arquitectura lógica do projecto é composta por quatro componentes fundamentais (consultar figura 4.2). A componente GridMaster representa o servidor central da grelha e é responsável por todas as actividades de gestão (gestão dos clientes, pedidos e resultados). O GridWorker representa as unidades responsáveis pelo processamento. Estes dois últimos tipos de componentes formam a Grid. O programador, aquando a sua interacção com a Grid, é representado pela componente API, pois é através desta que é possível a submissão de pedidos e recolha de resultados. A última componente denomina-se Framework, e tratando-se de uma "biblioteca" que é partilhada por todas as outras componentes.

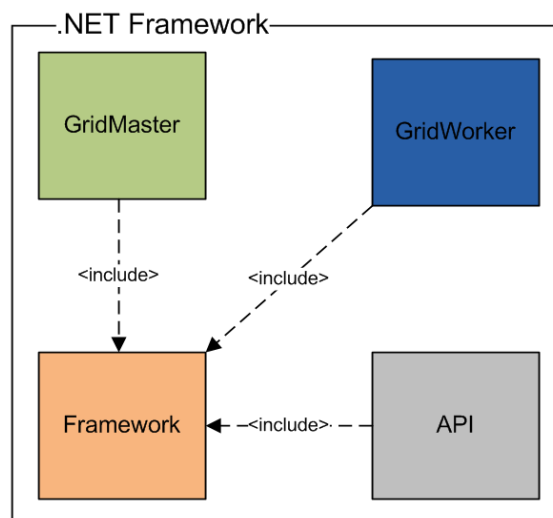


Figura 4.2: Diagrama de composição geral

Exceptuando a Framework, as restantes componentes são fortemente interactivas. A interacção entre os diversos componentes distribuídos assim como as consequências que esta interacção causa em cada um, constitui um dos principais inconvenientes durante as fases subsequentes do projecto. Deste modo, é extremamente importante conceber um modelo lógico que sirva de base de apoio para as restantes fases de desenvolvimento. Tal pode verificar-se no seguinte diagrama da figura 4.3. Este procura representar em simultâneo a composição dos componentes bem como a interacção entre os elementos que constituem esses mesmos componentes.

As descrições que se seguem procuram ajudar a explicar melhor o diagrama de arquitectura lógica (figura 4.3). Estas vão relatar o modo como reagem e funcionam os componentes, perante as diversas situações de interacção. Estas podem existir desde que o programador inicia a sua aplicação, contendo métodos que espera que sejam executados, de forma paralela e distribuída, até à respectiva recepção de resultados. As situações de interacção são, então, as seguintes:

Arquitectura

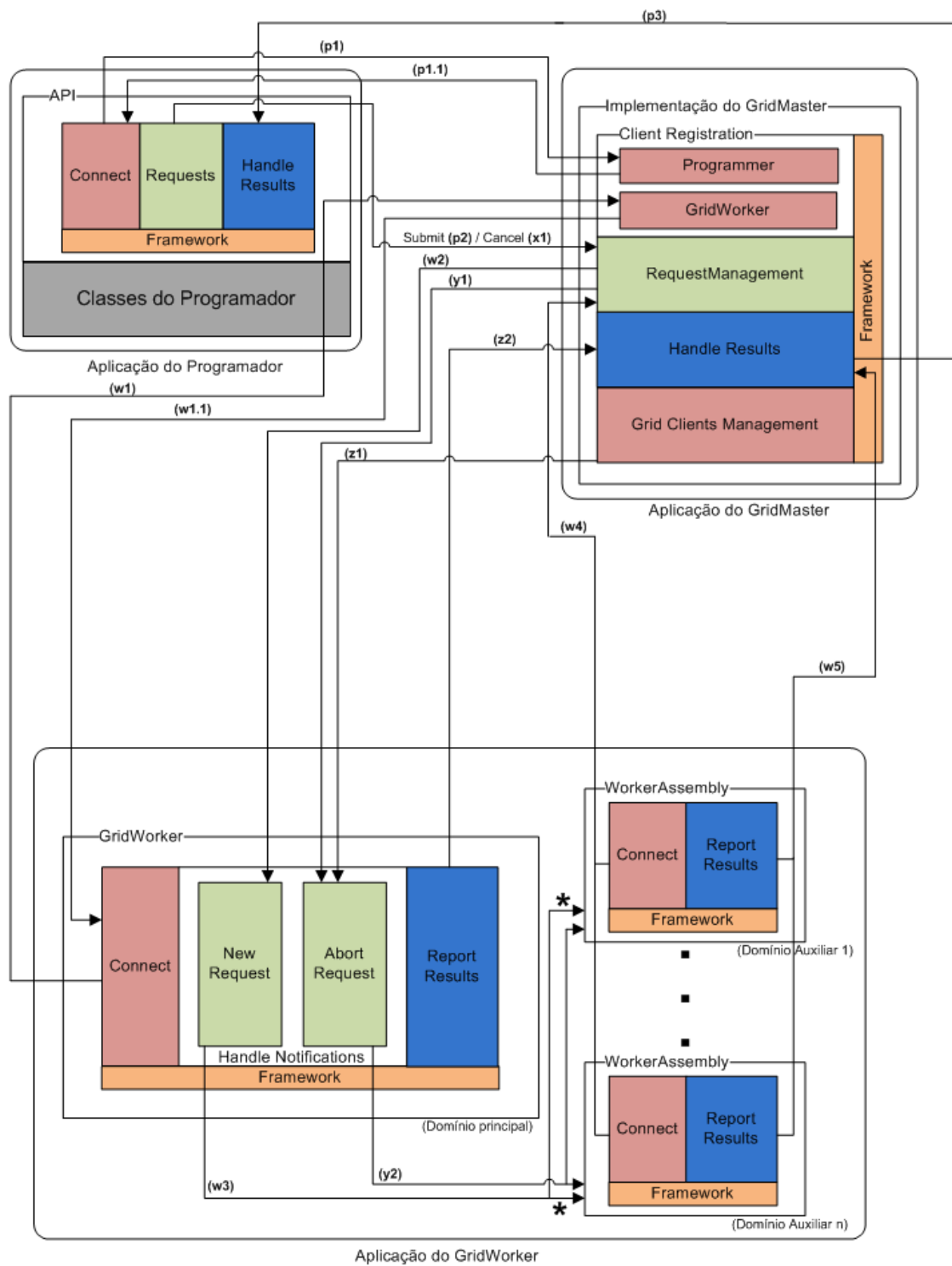


Figura 4.3: Arquitectura lógica do projecto

4.2.1 Registo dos clientes

Para que os clientes (API ou GridWorker) possam interagir com o servidor é necessário um registo prévio. Para o conseguir é necessária uma conexão ao servidor GridMaster de forma a obter uma referência para o objecto *singleton* nele residente. De seguida, basta proceder a uma invocação remota de um método que corresponde à autenticação ². Se esta for bem sucedida, o servidor devolve ao cliente uma instância de um objecto, que irá conter toda a informação necessária, para a todas as interacções futuras³ (ficando esta instância denominada "instância de registo"). A partir do momento que o cliente detém a posse desta instância, subscreve um *handler* para recepção de notificações e o processo de registo estará completo.

4.2.2 Construção da Grid

Para que se inicie a "construção" da Grid é necessário iniciar o servidor GridMaster com as devidas configurações. Seguidamente, vão-se iniciando os clientes executores, GridWorker, nas diversas máquinas espalhadas pela rede. Após a conexão e respectivo registo dos clientes executores, estes irão passar a fazer parte constituinte da Grid, aumentando a sua capacidade global de processamento, mas permanecendo à espera até que os seus serviços sejam solicitados pelo GridMaster.

4.2.3 Ciclo de execução

A partir do momento em que a aplicação do cliente (programador) inicia, a API irá tentar registar-se junto do GridMaster, embora seja necessário assegurar, em primeiro lugar, que todas as configurações estão correctas, e que o código do programador respeita a estrutura pretendida. Após a obtenção da respectiva instância de registo ⁴ será necessário analisar as tarefas que o programador definiu, convertendo-as em pedidos ("*Requests*"). Esta conversão é necessária, pois um pedido engloba mais informação que uma tarefa ("*Task*"), sendo vital para a sua distribuição pela Grid. Quando da conversão, é essencial determinar que *assemblies* serão necessários anexar ao pedido, antes da sua transferência para o GridMaster. Assim, terão de ser enviados os *assemblies* (e seus dependentes) que contenham classes correctamente definidas para execução distribuída. Após a correcta definição dos pedidos, o processo normal será submetê-los ao GridMaster e aguardar os resultados ⁵. O servidor, assim que recebe um pedido, analisa os recursos disponíveis e procede (se possível) à respectiva alocação. Para o fazer o GridMaster coloca toda a informação que o GridWorker irá necessitar para o processamento do pedido na sua

²Ver transições p1 e w1 na figura 4.3.

³Ver transições p1.1 e w1.1 na figura 4.3.

⁴Ver transição p1.1 na figura 4.3.

⁵Ver transição p2 na figura 4.3.

instância de registo, notificando-o, de seguida, através de um evento ⁶. Após a recepção da notificação (execução do *handler*), o GridWorker consulta a sua instância de registo para obter os *assemblies* que precisa de carregar. Posteriormente (e numa *thread* à parte) define um novo domínio de aplicação (*AppDomain*), injectando para lá os *assemblies* necessários, juntamente com um *assembly* auxiliar, denominado "WorkerAssembly"⁷. Este último *assembly*, após iniciar a sua execução, começa por estabelecer uma conexão com o GridMaster (apesar de não efectuar registo, autentica-se com as credencias do GridWorker que o criou) com o objectivo de recolher informação relativa ao pedido que irá processar ⁸. Se tudo estiver de acordo com o esperado, inicia a execução do pedido, e no final reporta ao GridMaster o resultado da execução ⁹. Em boa medida, é possível afirmar que a componente WorkerAssembly está para o GridWorker como o GridWorker está para o GridMaster. O WorkerAssembly irá correr numa *thread* e domínio diferente da instância principal do GridWorker, e tem como objectivo principal a execução de pedidos e respectivo envio de resultados. Já a instância principal do GridWorker tem como objectivo a gestão dos vários WorkerAssembly's (dependendo do número de núcleos de computação da máquina onde está alojado o GridWorker), ficando responsável pela sua criação, e terminação quando necessário. Sempre que o GridMaster é notificado por um GridWorker, este analisa o conteúdo da notificação, a fim de determinar, em concreto, quem foi o notificante (GridWorker). Posteriormente, este procede à reconfiguração da instância de registo, do GridWorker, e à notificação do cliente (API) ¹⁰. Após o cliente (API) ser notificado da disponibilidade dos resultados consulta a sua instância de registo. Seguidamente, procede à reconversão do pedido (Request) em tarefa (Task), antes de invocar os métodos que foram explicitamente definidos pelo programador, para a recepção de resultados.

⁶Ver transição w2 na figura 4.3.

⁷Ver transição w3 na figura 4.3.

⁸Ver transição w4 na figura 4.3.

⁹Ver transição w5 na figura 4.3.

¹⁰Ver transição p3 na figura 4.3.

4.2.4 Cancelamento de pedidos

O programador pode proceder a um cancelamento explícito de tarefas, que foram anteriormente submetidas para execução. Quando isto acontece a API notifica o GridMaster que pretende proceder ao cancelamento de pedidos (um ou mais) ¹¹. Para cada pedido a cancelar, o GridMaster determina se esse pedido já foi alocado para execução. Caso tal não aconteça, basta dar o pedido como cancelado, caso contrário é necessário emitir uma ordem de cancelamento ao GridWorker, que estará a executar o pedido ¹². O GridWorker ao receber uma ordem deste tipo, termina a *thread* associada à execução do pedido, e notifica o GridMaster, para que este possa proceder à actualização da sua instância de registo ¹³.

4.2.5 Ausências inesperadas

Tendo em conta que estamos perante um sistema distribuído é difícil garantir que todos os clientes, que interagem com o servidor, estejam sempre a funcionar correctamente. Logo, um ponto importante a levar em consideração, quando do desenvolvimento do servidor central, será torná-lo resistente à ausência inesperada dos clientes, que com ele interagem. Para o conseguir há que definir mecanismos que actuam quando ocorre alguma situação inesperada. A título de exemplo, se um GridWorker que se encontrava a processar um pedido se desliga da Grid de forma inesperada, é necessário proceder a uma re-alocação. Por outro lado, se for o cliente API a desligar-se de forma inesperada, o GridMaster terá de tentar abortar o processamento dos pedidos que foram submetidos pelo mesmo (pedido para abortar ¹⁴ e de seguida a confirmação ¹⁵).

4.2.6 Abortar processamento

A fim de garantir a disponibilidade da Grid, é necessário salvaguardar situações de *deadlock*. Estas situações podem ocorrer quando da execução de código submetido pelo cliente (API). Como tal, é fundamental criar mecanismos que previnam este tipo de situações. Uma das soluções será o GridMaster analisar periodicamente os pedidos submetidos, a fim de determinar se estes já excederam o tempo máximo permitido de processamento. Em caso afirmativo, a ideia será notificar o GridWorker para que aborte o processamento ¹⁶. Quando o fizer, o GridWorker notificará o GridMaster ¹⁷, que por sua vez notificará o cliente ¹⁸.

¹¹ Ver transição x1 na figura 4.3.

¹² Ver transição y1 na figura 4.3.

¹³ Ver transição z1 na figura 4.3.

¹⁴ Ver transição z1 na figura 4.3.

¹⁵ Ver transição z2 na figura 4.3.

¹⁶ Ver transição z1 na figura 4.3.

¹⁷ Ver transição z2 na figura 4.3.

¹⁸ Ver transição p3 na figura 4.3.

4.3 Diagramas de composição

Os diagramas que se seguem têm como objectivo expor a composição dos componentes constituintes do sistema, de modo mais pormenorizado.

4.3.1 Framework

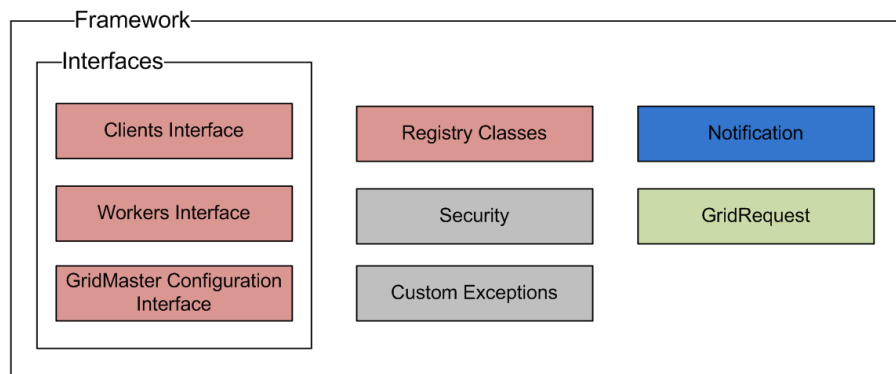


Figura 4.4: Composição do componente Framework

Descrição dos elementos da Framework (figura 4.4):

- **Registry Classes:** definem que informação guardar nas "instâncias de registo" dos clientes que interagem com a Grid.
- **Interfaces:** guardam informação relativa à interacção (protótipos dos métodos) que cada cliente tem com o servidor (GridMaster). A "ClientInterface" diz respeito apenas aos clientes "API". A "WorkerInterface" está direccionado para os Grid-Worker's. Por fim a "GridMasterConfigurationInterface" destina-se a qualquer entidade que pretenda configurar o GridMaster em tempo de execução.
- **Security:** guardam informação relacionada com os mecanismos necessários para garantir a encriptação (*MD5*) e a passagem segura de informação entre cliente e servidor.
- **Notification:** define que informação incluir em cada tipo de notificação, seja cliente-servidor ou servidor-cliente.
- **GridRequest:** define a informação que é necessária associar aos pedidos (Requests) que circulam na Grid.
- **Custom Exceptions:** guardam todas as excepções personalizadas, e como tal, adequadas a serem usadas dentro do ambiente da Grid pelos diversos componentes que a constituem, e que com ela interagem.

4.3.2 API

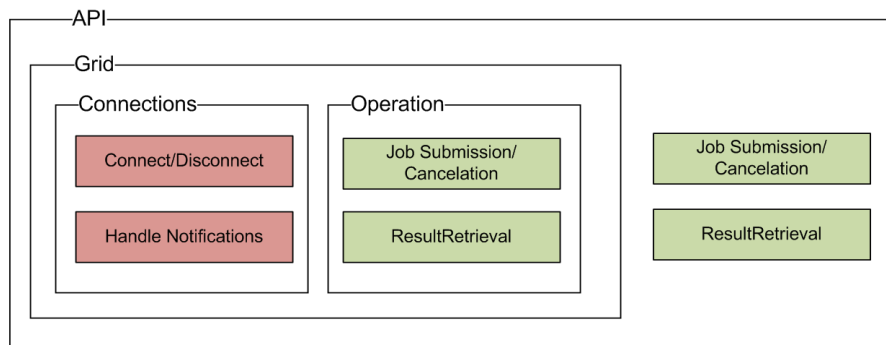


Figura 4.5: Composição do componente API

Descrição dos elementos da API (figura 4.5):

- **Grid-Connection:** engloba os mecanismos para estabelecer, manter e terminar a conexão com o servidor, e gestão de notificações.
- **Grid-Operation:** engloba todos os mecanismos necessários para tirar partido da conexão estabelecida com o servidor. Permite a submissão e cancelamento de pedidos ("JobSubmission" e "JobWaiting" respectivamente). Permite, ainda, a recolha de resultados.
- **Task:** serve para que o programador defina as tarefas que irão ser colocadas em execução paralela e distribuída. Tendo em conta que as Task's serão inevitavelmente convertidas quando da sua submissão, a sua existência apenas permite ao sistema esconder a complexidade associada às Request's. Assim, o programador é distanciado de detalhes desnecessários.
- **Job:** tem como finalidade melhorar a estruturação associada à definição de Task's do lado do programador. Servindo como contentor de Task's, este permite ao programador definir vários conjuntos de Task's a submeter, em momentos distintos, e associar a cada um, diferentes mecanismos de recolha de resultados.
- **Worker:** definição base, da qual todas as classes a incluir nas tarefas que poderão ser colocadas em execução paralela e distribuída, terão de derivar.

4.3.3 GridMaster

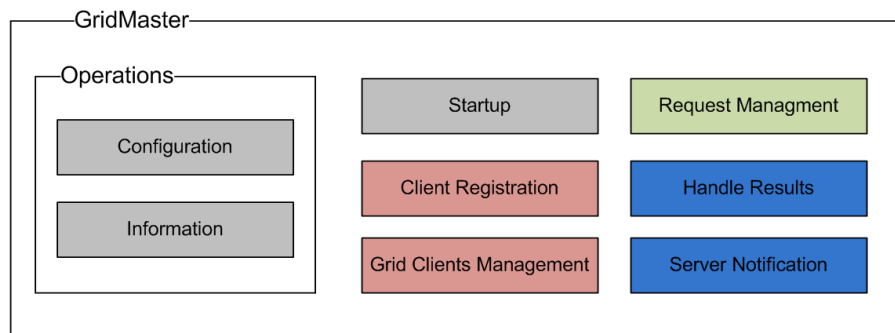


Figura 4.6: Composição do componente GridMaster

Descrição dos elementos do GridMaster (figura 4.6):

- **Startup** :componente de arranque do servidor (GridMaster), onde este carrega as definições provenientes de um ficheiro de configuração e inicia os serviços necessários.
- **Client Registration**: interage com o registo dos clientes.
- **Grid Clients Management**: componente que lida com situações anómalas e inesperadas, no que diz respeito à gestão de clientes. Estas situações ocorrem quando da detecção de *deadlock's* e ausência inesperada de clientes.
- **Request Management**: destinada a gerir todo o fluxo de pedidos que circulam na Grid. Desde a alocação, após submissão por parte do cliente API, até à re-alocação, aquando da falha de um cliente executor (GridWorker).
- **Handle Results**: tem como finalidade encaminhar os resultados para o respectivo cliente API assim que o cliente executor GridWorker terminar o processamento.
- **Server Notification**: módulo que será usado pelos clientes API ou GridWorker para notificar o servidor.
- **Operations**: componente usada por outro tipo de cliente, com missão diferente dos do tipo API ou GridWorker. Usada para configurar o servidor em tempo de execução (Módulo Configuration), ou obter informação apenas conhecida pelo servidor (Módulo Information). Esta componente pode vir a ser usada, por exemplo, por clientes cuja função seja monitorizar e/ou configurar o servidor em tempo de execução.

4.3.4 GridWorker

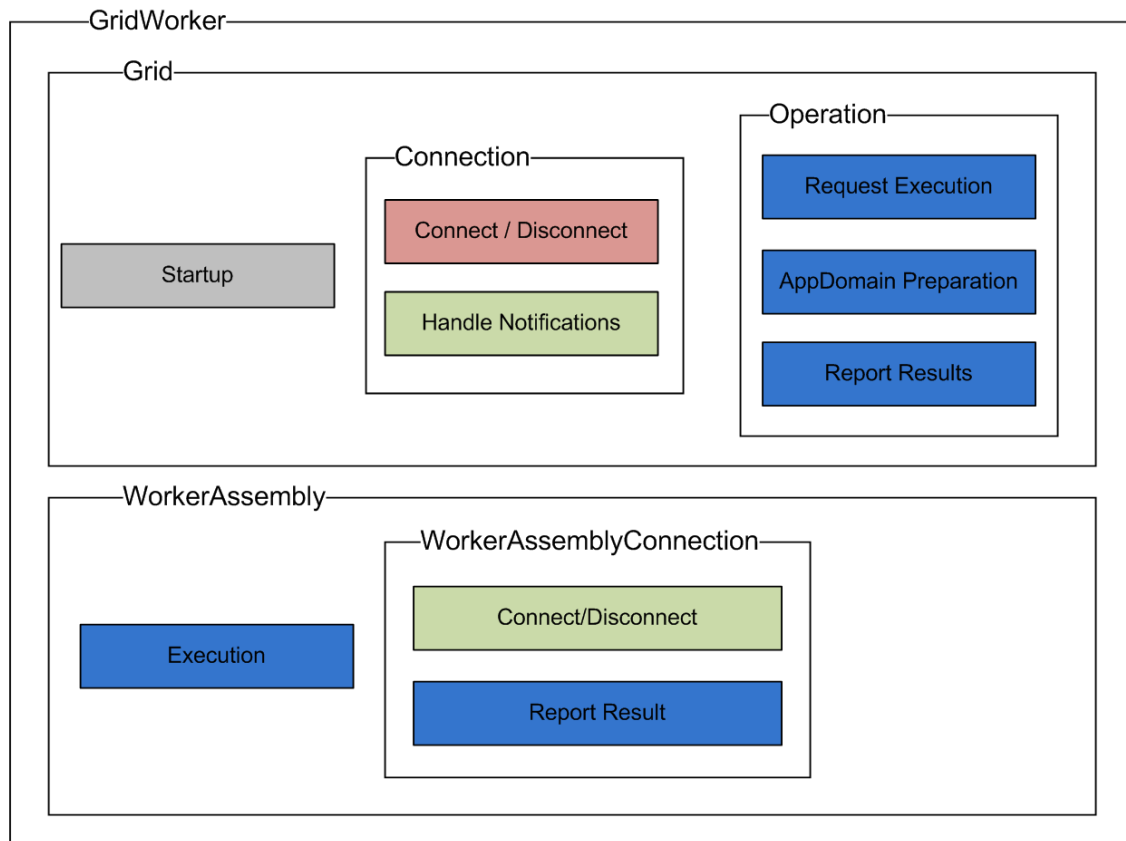


Figura 4.7: Composição do componente GridWorker

A componente GridWorker divide-se em dois módulos. O principal denomina-se "GridWorker" e tem como finalidade a gestão dos módulos do tipo "WorkerAssembly" (segundo módulo).

Descrição dos elementos do GridWorker (figura 4.7):

- **GridWorker-Startup:** componente de arranque da componente, onde esta carrega as definições provenientes de um ficheiro de configuração e inicializa os serviços necessários.
- **GridWorker-Connection:** engloba os mecanismos para estabelecer, manter e terminar a conexão com o servidor, e gestão de notificações.
- **GridWorker-Operation:** contém todos os mecanismos de gestão necessários para controlar a execução de pedidos ("Request Execution") e, consequentemente, o "WorkerAssembly".

O módulo auxiliar **WorkerAssembly** tem como objectivo a execução efectiva dos pedidos, estando para isso dotado dos seguintes módulos:

- **WorkerAssemblyConnection**: engloba os mecanismos para estabelecer a conexão ao servidor com o objectivo de recolher informação, relativa ao pedido a processar, e respectiva transmissão de resultados.
- **Execution**: módulo que define o mecanismo de execução concreto. Usando a tecnologia *.NET Reflection* invoca métodos previamente definidos pelo programador antes da submissão do pedido.

4.4 Diagramas de interacção

Os diagramas que se seguem têm como objectivo detalhar as sequências de interacção mais importantes, envolvendo as componentes principais do sistema. Embora estes diagramas sejam auto-explicativos, cada um será complementado com uma breve descrição, abarcando os pontos principais e mais relevantes.

4.4.1 API e GridMaster

Tendo em conta que esta componente é uma API, o modo como a aplicação final interage com o servidor está dependente da implementação do programador. O diagrama apresentado (consultar figura 4.8) corresponde a uma sequência normal de procedimentos, que se espera que o programador siga, para que consiga colocar um conjunto de tarefas em execução distribuída.

De forma a iniciar a comunicação com o servidor é necessário configurar o acesso ao mesmo. Esta configuração passa pela definição de IP (porta de acesso) credenciais e restantes parâmetros¹⁹. Para além disso, existe uma tentativa de comunicação com o servidor cuja finalidade corresponde à autenticação²⁰. O servidor (GridMaster), após analisar as credenciais recebidas, comunica o resultado ao cliente²¹. Este resultado pode corresponder a uma mensagem de erro, se o processo de autenticação falhar, ou pode corresponder à "instância de registo" do cliente. Após a criação dos *Jobs* e *Tasks*, representativos das tarefas que o programador pretende distribuir, a API irá proceder à sua conversão para *Request's* (pedidos de processamento adequados às necessidades da Grid) antes da sua submissão junto do GridMaster²². A partir deste ponto apenas se irá ficar à espera da conclusão dos pedidos submetidos²³. À medida que os pedidos forem sendo concluídos, cabe ao servidor encaminhar os resultados para o cliente API²⁴. Quando da recepção de resultados, a API precisa de transportar o pedido novamente para o contexto do programador, ou seja, transformar o *Request* que contém o resultado em *Tasks* e *Jobs*²⁵. Assim que recebe o resultado de todos os pedidos, a ideia será notificar o servidor do termo das comunicações²⁶. Este último passo não é obrigatório. O cliente pode simplesmente abandonar a interacção, e esperar que o servidor se aperceba disso. No entanto, um desconectar explícito é desejável sob o ponto de vista da eficiência.

¹⁹Ver transição a) na figura 4.8.

²⁰Ver transição b) na figura 4.8.

²¹Ver transição c) na figura 4.8.

²²Ver transição e) na figura 4.8.

²³Ver transição f) na figura 4.8.

²⁴Ver transição g) na figura 4.8.

²⁵Ver transição h) na figura 4.8.

²⁶Ver transição i) na figura 4.8.

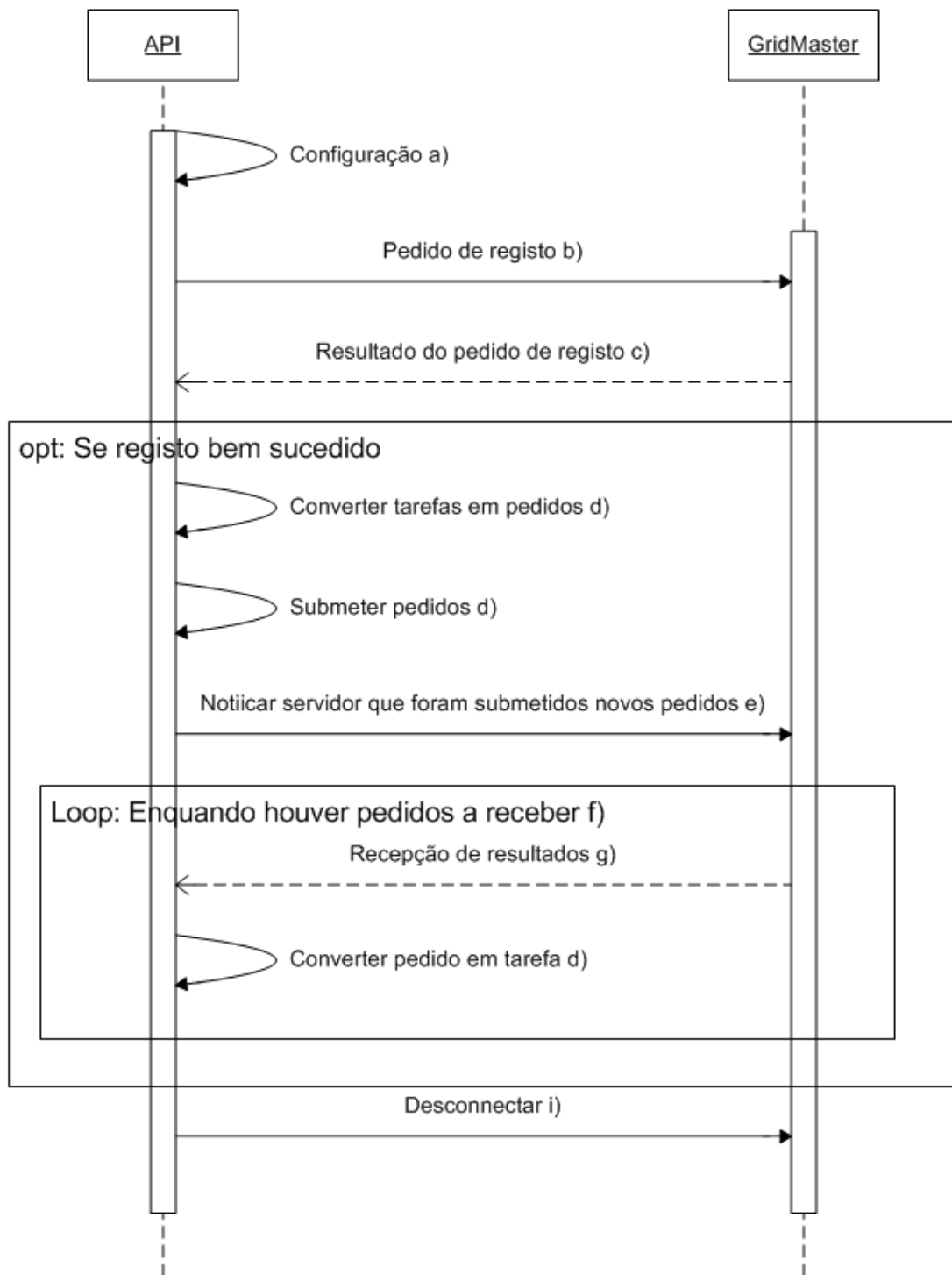


Figura 4.8: Interacção entre API e GridMaster

4.4.2 GridWorker e GridMaster

A interacção entre GridWorker e GridMaster é sempre a mesma, independentemente da implementação do programador (consultar figura 4.9). Apenas serão explicitadas as interacções mais importantes.

Seguem-se os detalhes relativos à figura 4.9:

- **Ciclo infinito**²⁷: a componente GridWorker não termina por si só, nem está previsto que o faça. Para que termine, tem de haver uma ordem explícita para o efeito. Essa ordem pode vir do GridMaster (no pedido²⁸), ou de uma outra aplicação cujo objectivo é controlar o GridWorker. Assim que houver um desconectar explícito cabe ao GridWorker abortar o processamento das componentes WorkerAssembly que dele dependem²⁹.
- **Tratamento de pedidos**: os pedidos recebidos pelo GridWorker³⁰ podem ser de dois tipos. O primeiro tipo³¹ desencadeia uma acção de preparação para o processamento³², sendo o processamento efectuado pelo WorkerAssembly. O segundo tipo³³ desencadeia um mecanismo para abortar o processamento de um dado WorkerAssembly³⁴, notificando de seguida o servidor³⁵.

²⁷Ver transição a) na figura 4.9.

²⁸Ver transição b) na figura 4.9.

²⁹Ver transição h) na figura 4.9.

³⁰Ver transição b) na figura 4.9.

³¹Ver transição c) na figura 4.9.

³²Ver transição d) na figura 4.9.

³³Ver transição e) na figura 4.9.

³⁴Ver transição f) na figura 4.9.

³⁵Ver transição g) na figura 4.9.

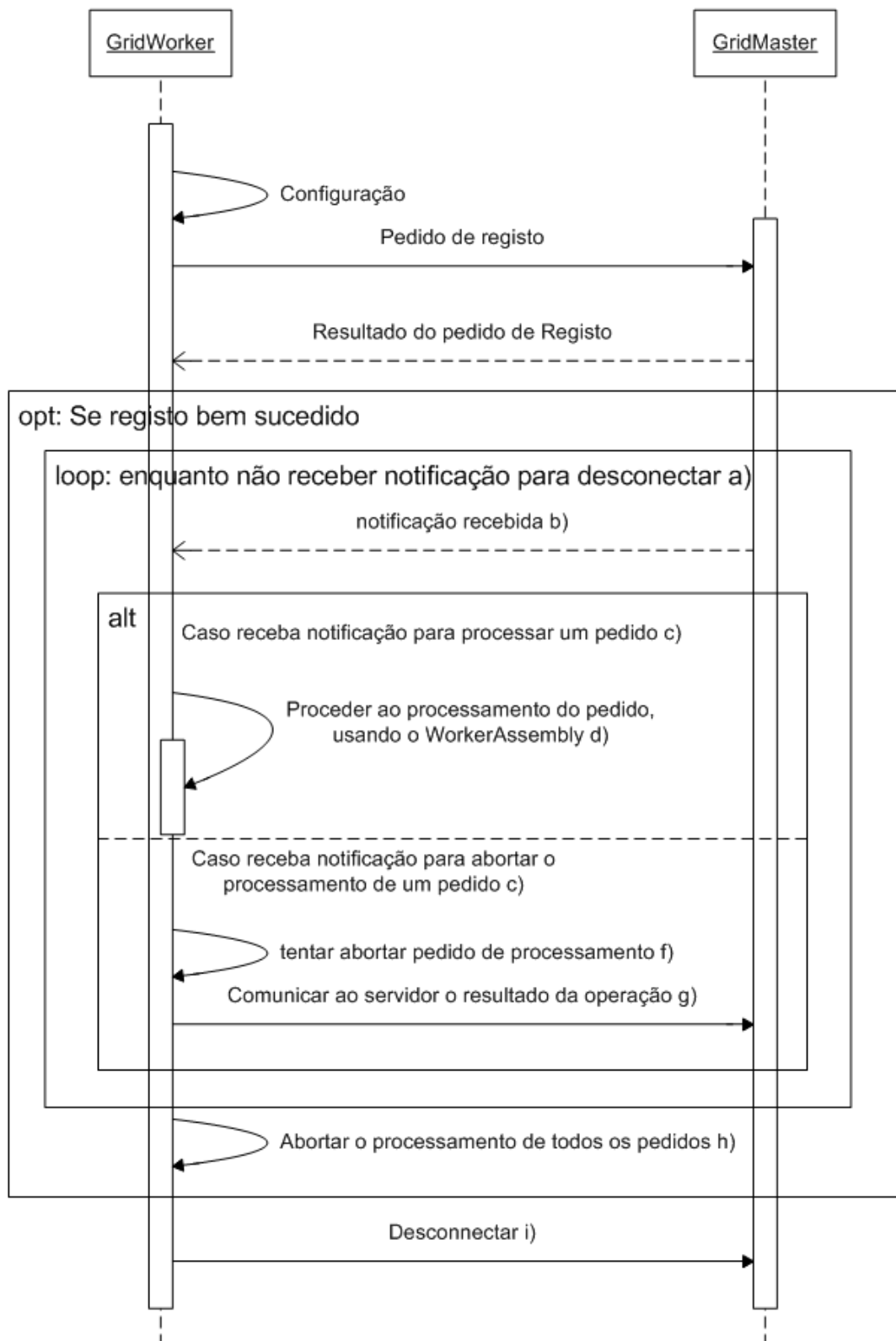


Figura 4.9: Interação entre GridWorker e GridMaster

4.4.3 WorkerAssembly e GridMaster

O diagrama que se segue (figura 4.10) corresponde a um dos passos ³⁶ do diagrama anterior, e diz respeito à interacção do módulo WorkerAssembly com o GridMaster.

Seguem-se os detalhes relativos à figura 4.10:

- **Obter informação relativa ao pedido a processar:** o módulo WorkerAssembly começa por configurar uma conexão ³⁷, apenas com o objectivo de obter informação relativa ao pedido que irá processar ³⁸. Se as credencias fornecidas forem correctas consegue obter a informação necessária para passar ao processamento ³⁹.
- **Processamento:** usando a informação consultada, invoca o método supostamente de processamento pesado associado ao pedido ⁴⁰, notificando o servidor quando a operação terminar ⁴¹.

³⁶Ver transição d) na figura 4.10.

³⁷Ver transição a) na figura 4.10.

³⁸Ver transição b) na figura 4.10.

³⁹Ver transição c) na figura 4.10.

⁴⁰Ver transição d) na figura 4.10.

⁴¹Ver transição e) na figura 4.10.

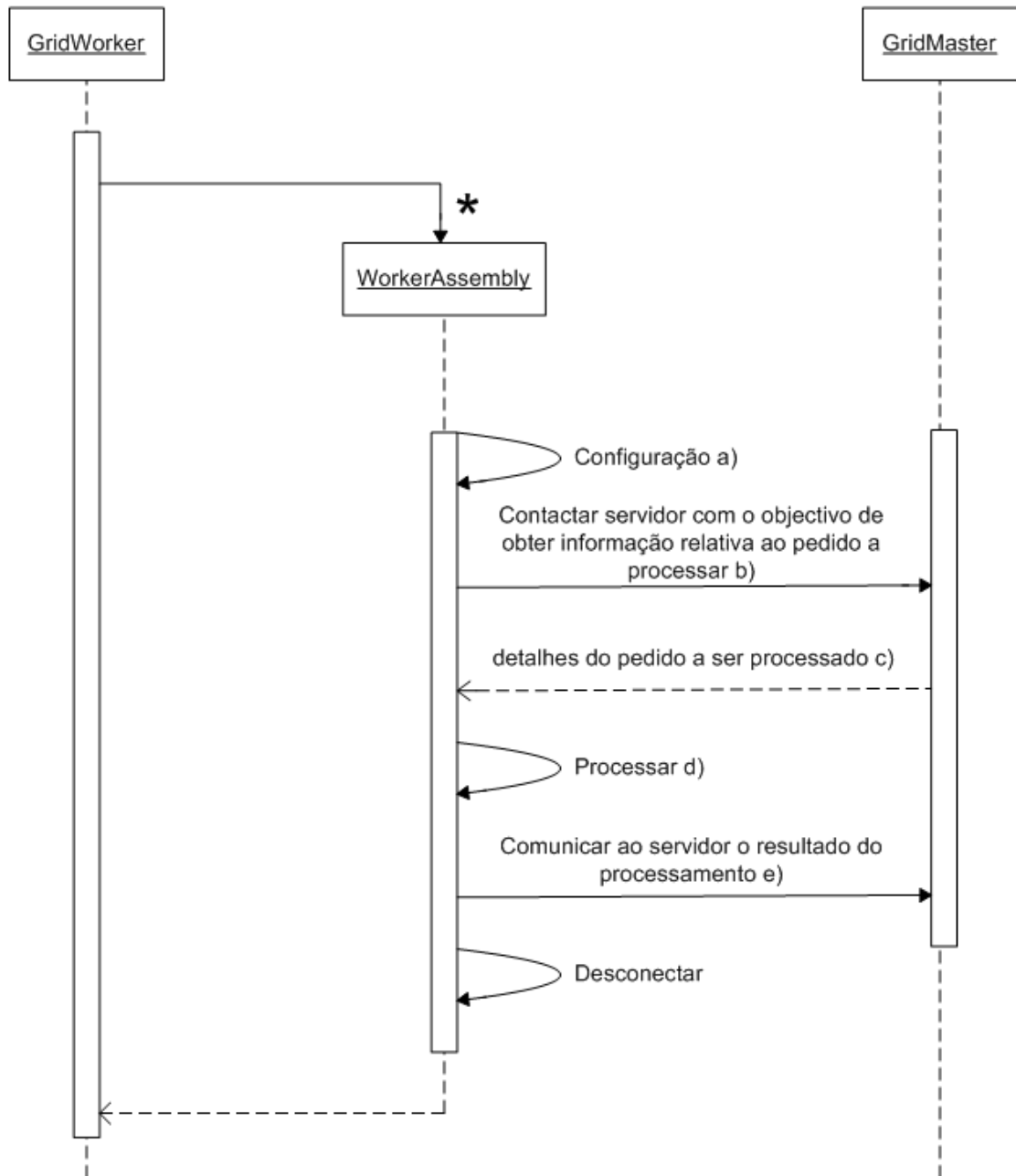


Figura 4.10: Interação entre WorkerAssembly e GridMaster

Capítulo 5

Implementação

Os objectivos principais deste capítulo são dar a conhecer o modo como ficou estruturado o sistema após a fase de implementação, explicar como é que o utilizador pode interagir com cada componente, dar a conhecer a metodologia dos testes efectuados ao sistema e indicar algumas das dificuldades sentidas durante a realização desta fase do trabalho. Na figura 5.1 é possível observar a distribuição do tempo dispendido durante a fase de implementação.

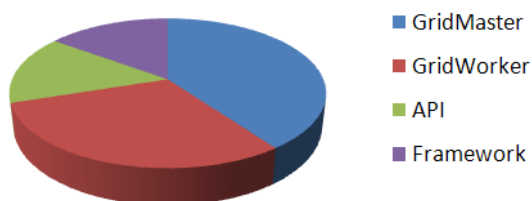


Figura 5.1: Divisão percentual do tempo durante a fase de implementação

5.1 GridMaster

Na figura 5.2 é possível observar os projectos que constituem a solução para a entidade GridMaster. A natureza da comunicação (fortemente concorrente) e a diversidade de interacção entre os elementos constituintes da Grid fazem com que seja necessária uma programação muito cuidada. Relevante é, também, o facto de que a qualquer instante pode haver um cliente que deixe de responder, levando a um erro de comunicação. Não há forma de determinar a actividade actual do GridMaster no momento do erro, querendo isto dizer que qualquer linha de código pode eventualmente lançar uma excepção. Sendo assim, é obrigatório que o servidor esteja preparado para reagir convenientemente, independente da origem, ou causa da excepção. No anexo A é possível observar um dos métodos pertencentes ao projecto "GridMasterImplementation" que melhor espelha o cenário aqui descrito (consultar anexo A.1).

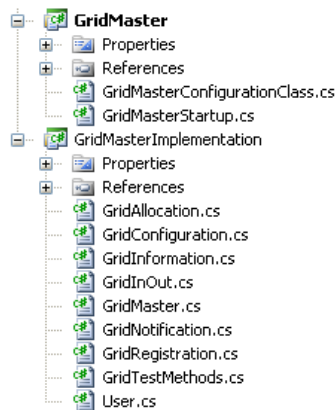


Figura 5.2: Árvore de ficheiros da solução GridMaster

5.1.1 Configuração

Inicialmente, o único meio para o utilizador interagir com o GridMaster corresponde ao uso de um ficheiro de configuração em XML (consultar figura 5.3).

```
<?xml version="1.0" encoding="utf-8"?>
<GridMasterConfigurationClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
"http://www.w3.org/2001/XMLSchema">
  <BindTo>127.0.0.1</BindTo>
  <Port>60045</Port>
  <EnsureSecurity>true</EnsureSecurity>
  <ClientsRenewSeconds>60</ClientsRenewSeconds>
  <WorkersRenewSeconds>60</WorkersRenewSeconds>
  <TimeoutPoolingSeconds>5</TimeoutPoolingSeconds>
  <DefaultTimeoutInSeconds>30</DefaultTimeoutInSeconds>
  <MaxRequestReallocation>1</MaxRequestReallocation>
  <RegistryToleranceInSeconds>60</RegistryToleranceInSeconds>
  <Users>
    <User>
      <UserName>bob</UserName>
      <PassDigest>9f9d51bc70ef21ca5c14f307980a29d8</PassDigest>
    </User>
    <User>
      <UserName>john</UserName>
      <PassDigest>527bd5b5d689e2c32ae974c6229ff785</PassDigest>
    </User>
  </Users>
</GridMasterConfigurationClass>
```

Figura 5.3: Ficheiro de configuração da aplicação GridMaster

O ficheiro de configuração contém vários campos devidamente enquadrados em etiquetas *XML* que se pretendem auto-explicativas (figura 5.3):

- **BindTo e Port:** endereço e porta em que o servidor irá escutar.
- **EnsureSecurity:** indica ao servidor se este deverá usar os procedimentos necessários para garantir uma comunicação segura.
- **ClientsRenewSeconds e WorkersRenewSeconds:** indica a periodicidade em segundos em que os clientes do servidor devem proceder a uma renovação explícita do tempo de vida.
- **TimeoutPoolingSeconds:** define a periodicidade com que é feita a análise ao tempo que cada pedido está a processar.
- **DefaultTimeoutInSeconds:** valor por defeito que define por quanto tempo um pedido pode ser processado, sem que seja abortado.
- **MaxRequestReallocation:** indica o número máximo de vezes que um pedido pode ser alocado.
- **RegistryToleranceInSeconds:** período de tempo, após registo, que deve ser concedido ao cliente, para que fique completamente operacional.
- **UserName e PassDigest:** nome do utilizador e palavra passe encriptada usando o algoritmo *MD5*.

5.2 Framework

A biblioteca principal do sistema (Framework) não sofreu alterações a nível estrutural durante a fase de implementação e teste. Apenas a definição de algumas classes foi gradualmente se modificando, à medida que se foi corrigindo defeitos e introduzindo novas funcionalidades. A figura 5.4 corresponde à versão final da árvore de ficheiros.

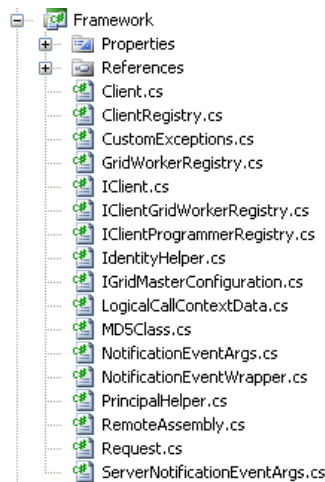


Figura 5.4: Árvore de ficheiros da solução Framework

5.3 GridWorker

Ambos os projectos que constituem a solução para a entidade GridWorker (consultar figura 5.5) vieram a sofrer alterações ao longo das fases de implementação e teste. A dificuldade associada à invocação remota de métodos e à modificação estrutural, para acomodar o processamento de vários pedidos em simultâneo, constituíram os principais contratempos e desafios.

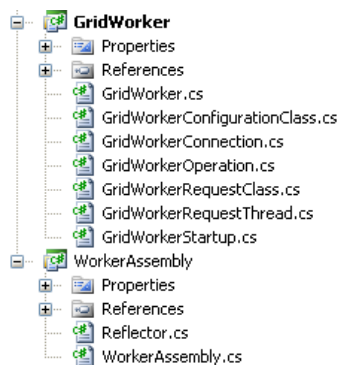


Figura 5.5: Árvore de ficheiros da solução GridWorker

5.3.1 Configuração

O ficheiro de configuração do GridWorker (ver figura 5.6) contém as credenciais de acesso ao servidor (GridMaster) e alguns parâmetros de configuração. Antes de surgir a interface dedicada à interacção com o GridWorker, esta constituía a única forma de configuração.

```
<?xml version="1.0" encoding="utf-8"?>
<GridWorkerConfigurationClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
  <BindTo>127.0.0.1</BindTo>
  <HostToBind>127.0.0.1</HostToBind>
  <Port>60045</Port>
  <UserName>bob</UserName>
  <PassDigest>9ff9d51bc70ef21ca5c14f307980a29d8</PassDigest>
  <EnsureSecurity>true</EnsureSecurity>
  <KeepAlivePoolingSeconds>59</KeepAlivePoolingSeconds>
  <NumberOfCores>1</NumberOfCores>
</GridWorkerConfigurationClass>
```

Figura 5.6: Ficheiro de configuração da aplicação GridWorker

Cada um dos campos do ficheiro de configuração (figura 5.6) tem o seguinte significado:

- **BindTo**: endereço *IP* ao qual o aplicativo GridWorker se vai associar para ficar à escuta de pedidos provenientes do servidor.
- **HostToBind/Port**: endereço *IP* e Porta ao qual o GridWorker se deve ligar para estabelecer conexão com o servidor.
- **UserName**: nome do utilizador que é fornecido ao servidor no momento do registo.
- **PassDigest**: palavra passe encriptada recorrendo ao algoritmo *MD5*. É fornecida ao servidor no momento do registo.

5.4 API

Foi o componente que mais estável se manteve ao longo de toda a fase de desenvolvimento, relativamente à sua concepção original. Sofreu apenas algumas alterações quando da introdução da possibilidade de cancelar pedidos. A figura 5.7 descreve o projecto que corresponde à implementação desta componente.

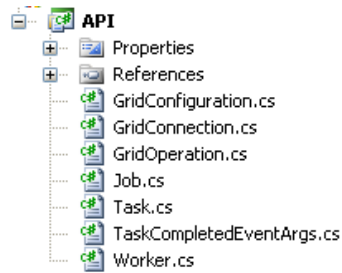


Figura 5.7: Árvore de ficheiros da solução API

5.4.1 Metodologia para uso da API

O objectivo da presente secção será apresentar a metodologia para o uso da API.

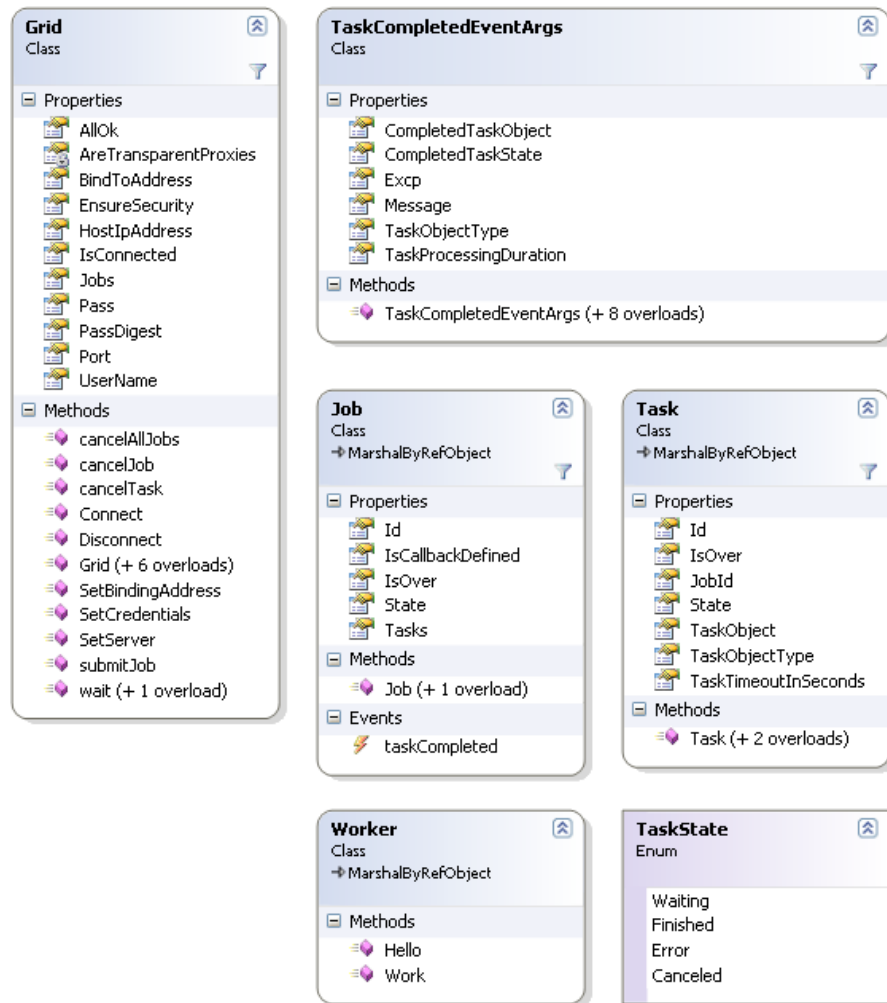


Figura 5.8: Funcionalidades disponibilizadas pela API ao programador

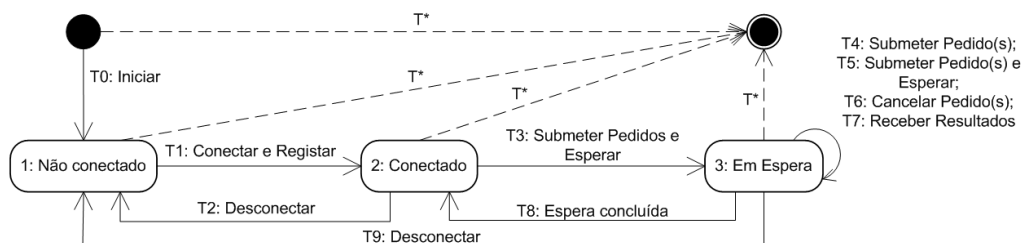


Figura 5.9: Estados possíveis de uma aplicação que use as funcionalidades disponibilizadas pela API

São disponibilizadas ao programador um conjunto de funcionalidades (observar figura 5.8), podendo estas ser utilizadas de diferentes formas (observar figura 5.9), obtendo-se assim resultados diversos. Seguem-se as principais operações a ser efectuadas por um programa que utilize a *API* para processamento paralelo e distribuído..

5.4.1.1 Conexão e Registo

Após o início da aplicação¹, pretende-se estabelecer uma conexão com o servidor². Para o fazer basta instanciar um objecto do tipo "Grid", configurar devidamente o acesso (no construtor, ou usando os "Set's"), e efectivar a conexão através do uso da função "Connect". Se tudo correr bem (tal pode ser verificado através das propriedades: "IsConnected" e "AllOk"), a conexão com o servidor é estabelecida e o registo é efectuado. A partir deste ponto a aplicação fica pronta para iniciar a interacção com o servidor.

5.4.1.2 Submissão de Pedidos

Em primeiro lugar, é necessário definir devidamente o conjunto de tarefas ("Job") a submeter bem como as tarefas ("Task") que compõem o conjunto. De seguida, será associada uma função para a recepção de resultados ("taskCompleted"). De referir ainda que para definir uma tarefa ("Task") é necessário fornecer uma instância de uma classe descendente da classe "Worker" e opcionalmente, definir o tempo máximo de processamento da tarefa. O método de processamento pesado, presente na classe descendente da classe "Worker" deve corresponder à redefinição do método "Work".

Posteriormente, será usada a instância da classe "Grid", obtida anteriormente, para invocar o método "submitJob"³. As estrutura anteriormente definidas são convertidas em pedidos (os chamados "Requests"), para que seja possível o seu envio para a Grid, iniciando-se deste modo a interacção.

Embora seja desencadeada uma notificação quando existem resultados disponíveis, é possível a qualquer momento consultar as propriedades de qualquer um dos objectos envolvidos na interacção ("Grid", "Job" e "Task").

5.4.1.3 Espera

Após a submissão de pedidos, a ideia será invocar o método "wait" de forma a bloquear a aplicação, obrigando-a a esperar pela conclusão de uma tarefa ("Task"), ou conjunto de tarefas ("Job"), antes de desconectar⁴. De outra forma não é garantida a recepção de todos os resultados. O simples facto da aplicação "estar em espera"⁵ indica que está a

¹Ver transição T0 e estado 1 na figura 5.9.

²Ver transição T1 e estado 2 na figura 5.9.

³Ver transição T3 na figura 5.9.

⁴Ver transição T3 e estado 3 na figura 5.9.

⁵Ver estado 3 na figura 5.9.

decorrer um processo de interacção com Grid. Assim, não é conveniente que a aplicação termine ou se desconecte⁶ da Grid. A situação ideal⁷ seria apenas terminar quando todos os pedidos que foram submetidos para processamento forem concluídos ou cancelados⁸. A consequência de outro tipo de terminação será um pequeno desperdício a nível dos recursos computacionais da Grid⁹.

5.4.1.4 Cancelamento

A funcionalidade de cancelamento permite anular submissões efectuadas anteriormente, através da invocação dos métodos "cancelTask" (cancelar apenas uma tarefa), "cancelJob" (cancelar um conjunto de tarefas) e "cancelAllJobs" (cancelar todas as submissões efectuadas até ao momento)¹⁰. Esta funcionalidade representa uma ferramenta importante para a economização de recursos computacionais da Grid, pois evita o processamento desnecessário de pedidos.

5.4.1.5 Recepção de Resultados

Quando da recepção de resultados¹¹, o respectivo método ("taskCompleted") associado à tarefa é invocado, notificando deste modo a aplicação. A informação que lhe é passada neste momento está definida na classe "TaskCompletedEventArgs". As propriedades desta contêm toda a informação necessária para o tratamento do resultado.

5.4.1.6 Termo da conexão

O modo como a conexão é terminada traz diferentes consequências para a Grid. O ideal seria proceder a um desconectar explícito, apenas quando não estivesse a decorrer nenhuma interacção¹². Um desconectar explícito durante a interacção irá forçar um cancelamento de todas as submissões¹³, havendo neste caso (como em qualquer cancelamento) um pequeno desperdício. O pior cenário corresponde a esta situação, onde a aplicação termina de forma inesperada¹⁴. Nesta situação os pedidos anteriormente submetidos (caso existam) não são imediatamente cancelados, originando um desperdício de recursos computacionais da Grid. No caso da conexão ainda não ter sido estabelecida a Grid não sofre qualquer consequência.

Coloca-se no anexo [A.2](#) um exemplo do procedimento habitual do uso da *API*.

⁶Ver transição T9 na figura [5.9](#).

⁷Ver transição T8 na figura [5.9](#).

⁸Ver transição T6 e T7 na figura [5.9](#).

⁹Ver transição T9 e T* na figura [5.9](#).

¹⁰Ver transição T6 na figura [5.9](#).

¹¹Ver transição T7 na figura [5.9](#).

¹²Ver transição T2 na figura [5.9](#).

¹³Ver transição T9 na figura [5.9](#).

¹⁴Ver transição T* na figura [5.9](#).

5.5 Interface com o utilizador

Devido à sua natureza, o sistema desenvolvido não possui propriamente uma interface. Até ao momento o único meio de interacção seria o ficheiro de configuração inicial. No entanto, mais tarde surgiu a ideia de conceber interfaces para facilitar a interacção entre o utilizador e a Grid, introduzindo-se a possibilidade de configuração e monitorização em tempo de execução. Por conseguinte, foram concebidas interfaces "facilitadoras", para interagir com os aplicativos representativos das entidades GridWorker e GridMaster.

5.5.1 GridWorkerManager

A figura 5.10 corresponde à interface facilitadora do GridWorker. Esta permite configurar (antes de iniciar), iniciar e terminar a aplicação:

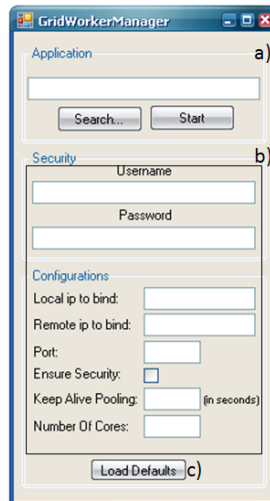


Figura 5.10: Interface da aplicação GridWorkerManager

As descrições que se seguem dizem respeito à imagem 5.10:

- **Controlo da aplicação** ¹⁵: permite iniciar e terminar a aplicação, sendo previamente necessário definir a sua correcta localização.
- **Dados do ficheiro de configuração** ¹⁶: possibilita definir os parâmetros de arranque da aplicação GridWorker. Esta funcionalidade apenas se limita a escrever para o ficheiro de configuração, ou seja, o arranque da aplicação não dispensa o uso do ficheiro.
- **Restaurar valores por defeito** ¹⁷: para cada parâmetro de arranque da aplicação GridWorker, são restituídos os valores por defeito.

¹⁵Ver indicação a) na figura 5.10

¹⁶Ver indicação b) na figura 5.10

¹⁷Ver indicação c) na figura 5.10

5.5.2 GridMasterManager

A figura 5.11 corresponde à interface facilitadora do GridMaster. Esta possui funcionalidades a nível da configuração e monitorização em tempo de execução.

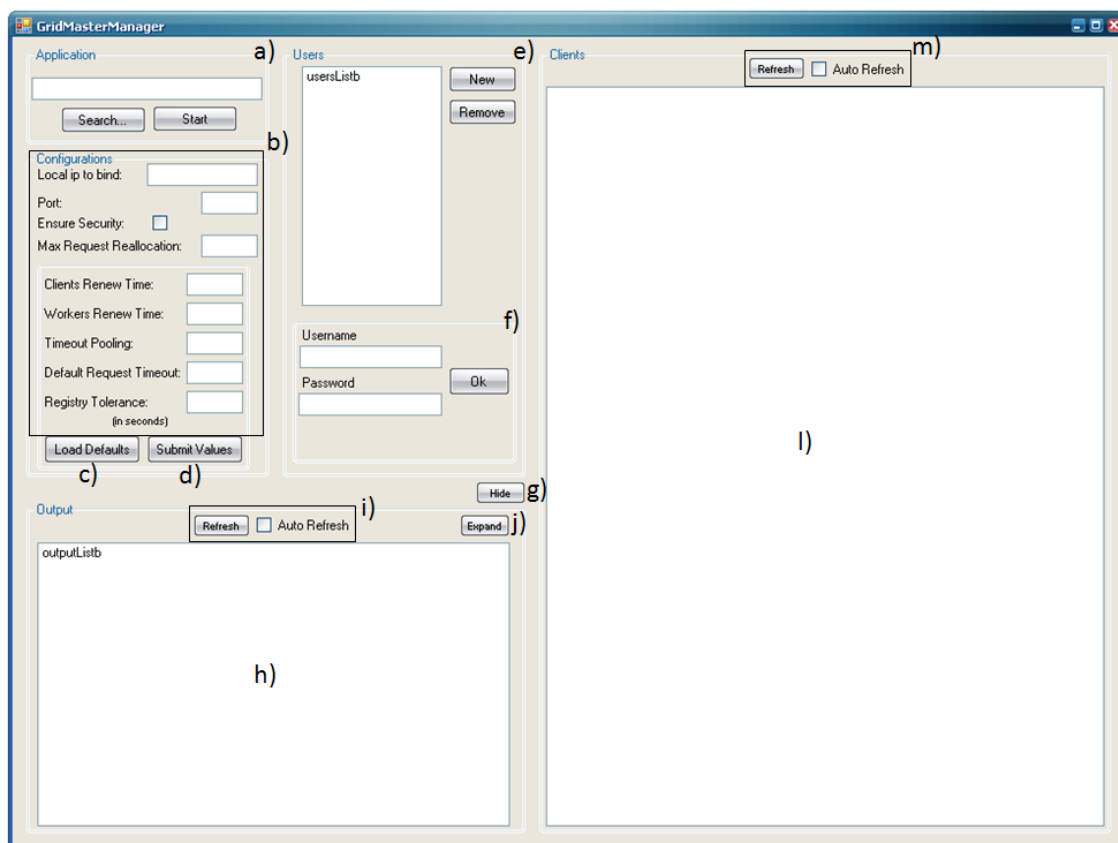


Figura 5.11: Interfac da aplicação GridMasterManager

As descrições que se seguem dizem respeito à imagem 5.11:

- **Controle da aplicação**¹⁸: após a correcta definição da localização, é permitido iniciar e terminar o aplicativo.
- **Dados do ficheiro de configuração**¹⁹: permite a definição dos parâmetros de arranque da aplicação GridMaster.
- **Restaurar valores por defeito**²⁰: para cada parâmetro de arranque da aplicação GridMaster, são restituídos os valores por defeito.

¹⁸Ver indicação a) na figura 5.11

¹⁹Ver indicação b) na figura 5.11

²⁰Ver indicação c) na figura 5.11

- **Submeter novos parâmetros configuração** ²¹: submete em tempo de execução, os novos parâmetros de configuração do servidor.
- **Gestão de utilizadores** ²²: local dedicado à gestão de utilizadores, possibilitando a inserção e remoção dos mesmos, antes e depois de iniciar a aplicação.
- **Adicionar utilizador** ²³: possibilita a inserção de novos utilizadores.
- **Botão para ocultar detalhe extra** ²⁴: diminui o tamanho da janela ocultando a zona onde é exibida a informação de saída do servidor, e também a zona onde é mostrada a informação dos clientes actualmente a interagir com a Grid.
- **Informação de saída do servidor** ²⁵: toda a informação de saída emitida pelo servidor (impressões que habitualmente iriam para a consola) surge nesta zona. A actualização pode ser feita de forma manual ou automática.
- **Actualização da informação de saída** ²⁶: o utilizador pode optar por uma actualização manual (mais eficiente), ou definir que a actualização da informação, de saída do servidor, será feita de forma automática (menos maçadora para o utilizador).
- **Botão de expansão** ²⁷: aumenta o tamanho da zona responsável pela exibição da informação de saída do servidor.
- **Informação dos clientes actualmente a interagir com a Grid** ²⁸: mostra informação detalhada de todos os clientes que se encontram de momento a interagir com a grelha.

²¹Ver indicação d) na figura 5.11

²²Ver indicação e) na figura 5.11

²³Ver indicação f) na figura 5.11

²⁴Ver indicação g) na figura 5.11

²⁵Ver indicação h) na figura 5.11

²⁶Ver indicação i) na figura 5.11

²⁷Ver indicação j) na figura 5.11

²⁸Ver indicação l) na figura 5.11

5.6 Testes

A natureza do sistema e a diversidade de interacções levou a que a fase de testes fosse a mais complexa e maçadora. O número de alterações e remodelações que foram necessárias efectuar, foi de tal ordem elevado, que se tornou difícil delinear uma fronteira clara entre a fase de implementação e a fase de testes. A maior parte das alterações incidiu nas funções de gestão da grelha residentes no servidor. As figuras 5.13 e 5.12 mostram estimativas percentuais do tempo dispendido durante a fase de testes.

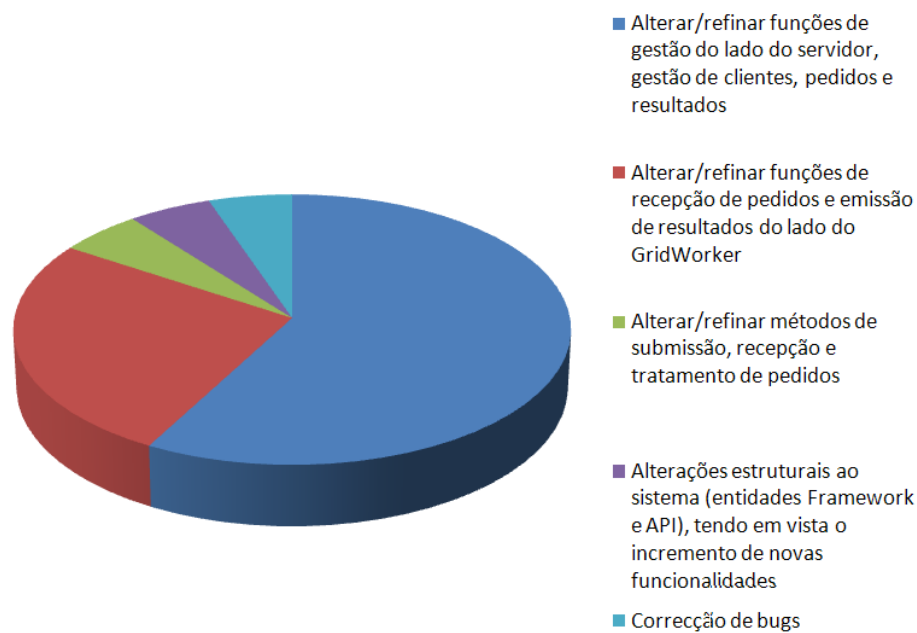


Figura 5.12: Divisão percentual do tempo durante a fase de testes, orientada às tarefas

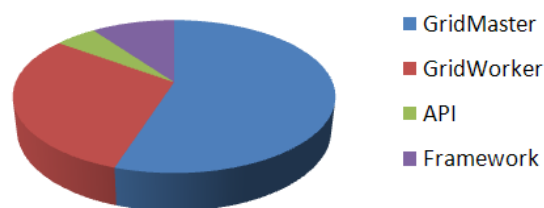


Figura 5.13: Divisão percentual do tempo durante a fase de testes, orientada às entidades

Capítulo 6

Aplicação de Demonstração e Análise de Resultados

6.1 Introdução

6.1.1 Contextualização

A aplicação de demonstração enquadra-se no contexto da análise numérica [Onl09e]. Mais precisamente em métodos numéricos para o cálculo de um conjunto particular de pontos, no plano complexo, segundo um algoritmo específico. O *fractal* formado pelos limites deste conjunto corresponde ao conjunto "Mandelbrot"[Onl09d]. Os estudos iniciais deste conjunto já datam de vários anos atrás. No entanto, desde as primeiras investigações até aos dias de hoje, existiram imensos matemáticos a contribuir para a compreensão deste conjunto numérico. A sua história, assim como as suas demais propriedades não serão aqui explicitadas, pois não vai ao encontro daquilo que é pretendido.

6.1.2 O Algoritmo

A definição do conjunto de Mandelbrot, juntamente com as suas propriedades básicas, sugere um simples algoritmo para desenhar uma imagem, representativa de uma região do conjunto. A ideia por detrás do mecanismo consiste em subdividir a região do plano complexo escolhida num certo número de *pixels*. Depois resta computar segundo um certo método (consultar algoritmo no anexo [A.3](#)) a fim de determinar a cor que aparecerá na imagem final. O que torna este algoritmo interessante e computacionalmente pesado é o cálculo para determinar a cor do *pixel*. A cor a atribuir ao pixel irá variar consoante o número de iterações que foram necessárias para determinar se a região representada pelo *pixel*, pertence ou não ao conjunto. O que acontece habitualmente é atribuir a cor branca a pontos que não pertencem, e a cor preta a pontos cujo processamento excedeu o número máximo de iterações (previamente definido), e que por isso provavelmente pertencem ao conjunto. Aos pontos intermédios será atribuída uma cor de acordo com o número de iterações que foram necessárias calcular para determinar O resultado final será uma imagem cujo conteúdo corresponde a um *fractal* composto por diferentes cores (ver figura [6.1](#)).

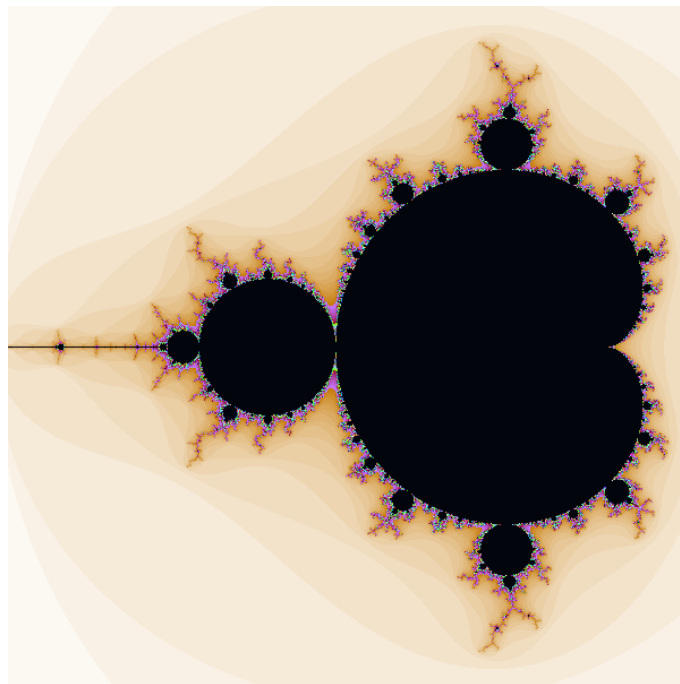


Figura 6.1: Exemplo de *fractal* representativo do conjunto de *Mandelbrot*

6.1.3 Motivação e objectivos

Ao escolher uma aplicação para demonstrar as potencialidades da *framework*, era preciso levar em consideração dois aspectos. O primeiro seria a quantidade de processamento que a sua utilização envolve, ou seja, a ideia seria desenvolver uma aplicação que mantivesse o processador "ocupado" por longos períodos de tempo. O segundo aspecto seria que o algoritmo fosse facilmente divisível em tarefas independentes. Ao implementar uma aplicação cujo objectivo é produzir imagens do conjunto de Mandelbrot, vamos ao encontro dos objectivos pretendidos. Para determinar a cor correspondente a certas regiões do conjunto são necessárias bastantes iterações (por vezes chegando ao limite, ficando o *pixel* com a cor preta), levando até o mais moderno dos processadores a demorar algum tempo a completar a operação. Para além disto, a cor a atribuir a cada *pixel* não está dependente das cores dos restantes *pixels* que compõem a imagem. Logo, é fácil dividir a imagem em fatias, afectando cada fatia uma tarefa diferente e independente. Outro aspecto que favorece a utilização em ambiente distribuído reside no facto da entrada ser mínima, em relação ao tempo de processamento (em certas situações), fazendo com que o *overhead*, de encaminhamento do pedido, seja mínimo em relação ao tempo que o executor dispensa para o processar. Ao implementar este aplicativo de demonstração foi possível evidenciar a facilidade de utilização da *framework* e, ao mesmo tempo, comprovar as suas potencialidades e níveis de eficiência.

6.1.4 Descrição da aplicação

A aplicação de demonstração consiste numa interface visualizadora de regiões do conjunto de Mandelbrot. Como tal, esta é munida de diversas opções, que permitem ao utilizador definir com facilidade regiões específicas. A aplicação é flexível, ao ponto de permitir que o processamento necessário para gerar a imagem seja local ou distribuído. No caso do processamento ser local, serão lançadas várias *threads* (habitualmente em número igual ao número de núcleos da máquina), e cada *thread* ficará encarregue de processar uma fatia da imagem. À medida que se forem obtendo resultados, estes serão transpostos para a interface. No caso do processamento ser distribuído, a ideia será dividir a imagem em fatias, associando cada uma à entrada de uma tarefa. O próximo passo consiste em usar a *framework* para submeter as tarefas para processamento distribuído. À medida que os resultados forem chegando, a interface é actualizada, com a fatia correspondente. Esta aplicação é especialmente útil pois permite estabelecer uma boa comparação entre o processamento local e o distribuído (usando a *framework*). É importante referir que uma leitura das páginas iniciais do documento [Dev08] antecedeu a fase de concepção do aplicativo de demonstração.

6.2 Arquitectura

Após uma breve introdução ao aplicativo de demonstração desenvolvido, segue-se uma sumária descrição da sua arquitectura. Os diagramas que se seguem têm como objectivo mostrar a arquitectura ao nível da composição e comportamento interno das componentes que a constituem.

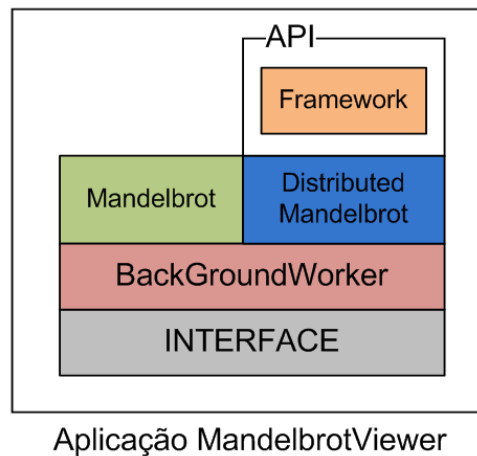


Figura 6.2: Arquitectura da aplicação de demonstração "MandelbrotViewer"

Na figura 6.2 é possível observar que a arquitectura se divide em camadas funcionais. A "Interface" corresponde ao *front-end* e as restantes correspondem ao *back-end*. É importante salientar que cada camada apenas interage directamente com as camadas que lhe são adjacentes.

6.2.1 Diagrama de composição

Na figura 6.3 é possível observar a composição detalhada da aplicação de demonstração.

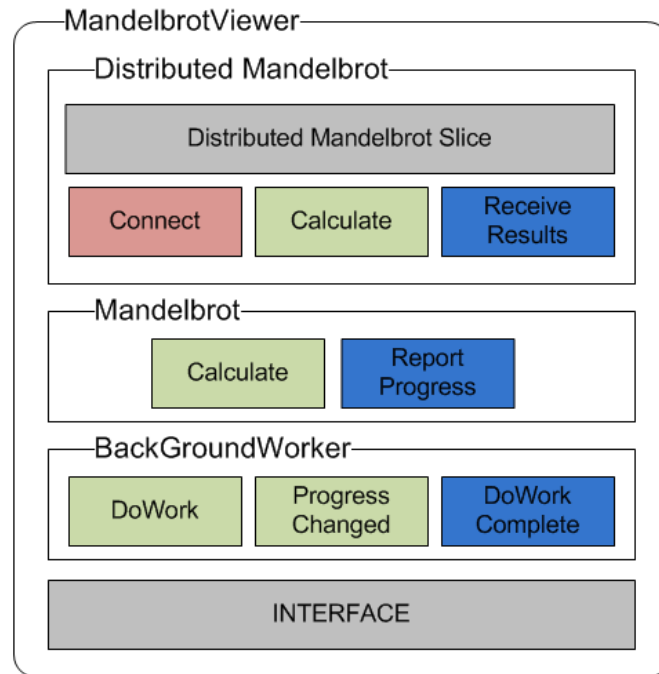


Figura 6.3: Composição da aplicação de demonstração "MandelbrotViewer"

Como complemento à figura 6.3 apresentam-se as seguintes descrições:

- **Interface:** engloba todo o *front-end* da aplicação. Contém um leque diversificado de elementos, a fim de interagir com o utilizador e com a camada que lhe é adjacente.
- **BackGroundWorker:** componente *back-end* que interage directamente com a interface. Os módulos "ProgressChanged" e "DoWorkComplete" têm como finalidade desencadear mecanismos que vão actualizar a interface à medida que os resultados vão estando disponíveis. O módulo "DoWork" é responsável pelo processamento e consequente produção de resultados. Esta componente encontra-se fortemente ligada às componentes "Mandelbrot" e "Distributed Mandelbrot", funcionando como ponte de ligação entre elas e a interface.
- **Mandelbrot:** componente dedicada ao cálculo local de imagens representativas de regiões do conjunto de Mandelbrot.
- **DistributedMandelbrot:** componente que possui estrutura e mecanismos devidamente definidos para tornar possível o uso da *API*, e consequente interacção com a Grid, conseguindo-se, desta forma, distribuir o processamento.

6.2.2 Diagrama de interacção

Na figura 6.4 é possível observar a interacção detalhada entre as componentes "Interface" e "BackGroundWorker".

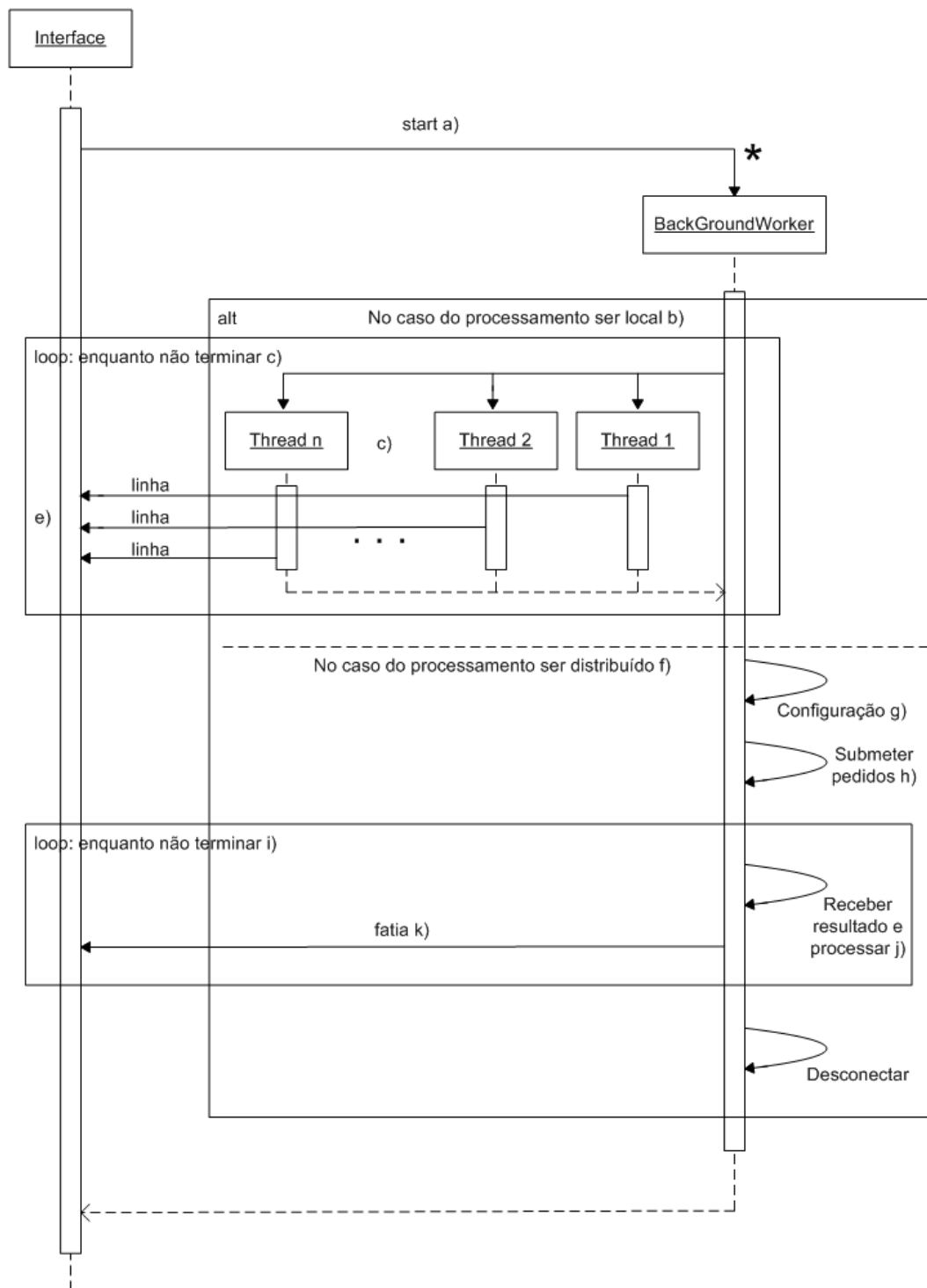


Figura 6.4: Interacção entres as componetes "Interface" e "BackGroundWorker"

Na figura 6.4 a entidade "Interface" após recolher e interpretar o input do utilizador, cria e lança em execução uma nova instância do chamado "BackGroundWorker" . Se o processamento for local ¹, serão criadas várias *threads* ², sendo o objectivo de cada uma, o cálculo de uma porção da imagem final. À medida que cada *thread* for obtendo resultados, estes serão imediatamente transpostos para a interface ³ (linha a linha). A geração da imagem termina quando todas as *threads* terminarem ou quando o utilizador cancelar o processo ⁴. No caso do processamento ser distribuído ⁵, a ideia será estabelecer uma conexão com o servidor e preparar os pedidos para submissão ⁶. De seguida, os pedidos são submetidos ⁷, tendo o aplicativo de ficar a aguardar pela sua conclusão, ou por um cancelar explícito do utilizador ⁸. À medida que os pedidos forem concluídos e os respectivos resultados forem chegando ⁹, a interface é actualizada ¹⁰. Cada resultado contém uma fatia da imagem.

6.3 Implementação

O objectivo desta secção consiste em explicitar os detalhes de implementação julgados pertinentes. Na figura 6.5 é possível observar a árvore de ficheiros do projecto MandelbrotViewer. É importante referir que o ficheiro "Mandelbrot Class" e o mecanismo de interacção entre a "Interface" e o "BackGrondWorker" foram adaptados do código anexo ao documento [Dev08]. Durante a fase de implementação a classe "MandelbrotClass.cs" foi modificada para que fosse possível a sua interacção com a Grid, dando origem à classe "DistributedMandelbrotClass.cs". Para além disso, foi produzida uma interface com extensas funcionalidades, com o objectivo de ser agradável de usar e, ao mesmo tempo, suficientemente completa e versátil para demonstrar as potencialidades da *framework*.

¹Ver transição b) na figura 6.4

²Ver transição d) na figura 6.4

³Ver transição e) na figura 6.4

⁴Ver transição c) na figura 6.4

⁵Ver transição f) na figura 6.4

⁶Ver transição g) na figura 6.4

⁷Ver transição h) na figura 6.4

⁸Ver transição i) na figura 6.4

⁹Ver transição j) na figura 6.4

¹⁰Ver transição k) na figura 6.4

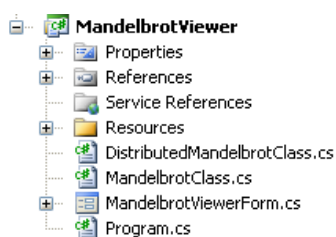


Figura 6.5: Árvore de ficheiros da solução MandelbrotViewer

6.3.1 Interface com o utilizador

As figuras que se seguem correspondem à interface do aplicativo de demonstração. A primeira figura diz respeito ao painel principal do aplicativo (figura 6.6), onde o utilizador define as entradas necessárias para a geração de imagens. A segunda figura (figura 6.7) descreve um segundo separador, que apenas está activo quando o processamento se realiza de forma distribuída, servindo apenas para monitorar o progresso.

Relativamente à figura 6.6 podemos descrever as seguintes zonas:

- **Painel¹¹**: zona onde aparecerá a imagem resultante do processamento (local ou distribuído).
- **Botão de início¹²**: as entradas ¹³ do utilizador são analisadas, e se estiverem de acordo com as especificações, o processamento inicia-se.
- **Configurações¹⁴**: local onde o utilizador pode definir vários parâmetros, relativos ao comportamento do aplicativo durante o processamento.
- **Input¹⁵**: local onde é definida a entrada a fornecer ao algoritmo que irá produzir a imagem.
- **Controlo¹⁶**: estes botões permitem deslocar a região que está actualmente a ser visualizada, produzindo uma nova imagem. Uma outra funcionalidade interessante, que não é imediatamente visível, diz respeito ao facto de ser possível usar o rato para definir uma nova região a ser visualizada (consultar figura B.2). Estas funcionalidades apenas estão disponíveis após a geração da primeira imagem, funcionando apenas como modificadores da entrada¹⁷.

¹¹Ver indicação a) na figura 6.6

¹²Ver indicação b) na figura 6.6

¹³Ver indicações c) e d) na figura 6.6

¹⁴Ver indicação c) na figura 6.6

¹⁵Ver indicação d) na figura 6.6

¹⁶Ver indicação e) na figura 6.6

¹⁷Ver indicação d) na figura 6.6

- **Guardar imagem** ¹⁸: permite guardar, para ficheiro, a imagem (produzida), representativa de uma região do conjunto de Mandelbrot.
- **Desenhar grelha** ¹⁹: desenha um referencial que se sobrepõe à imagem actual.
- **Barra de progresso** ²⁰
- **Tempo decorrido** ²¹

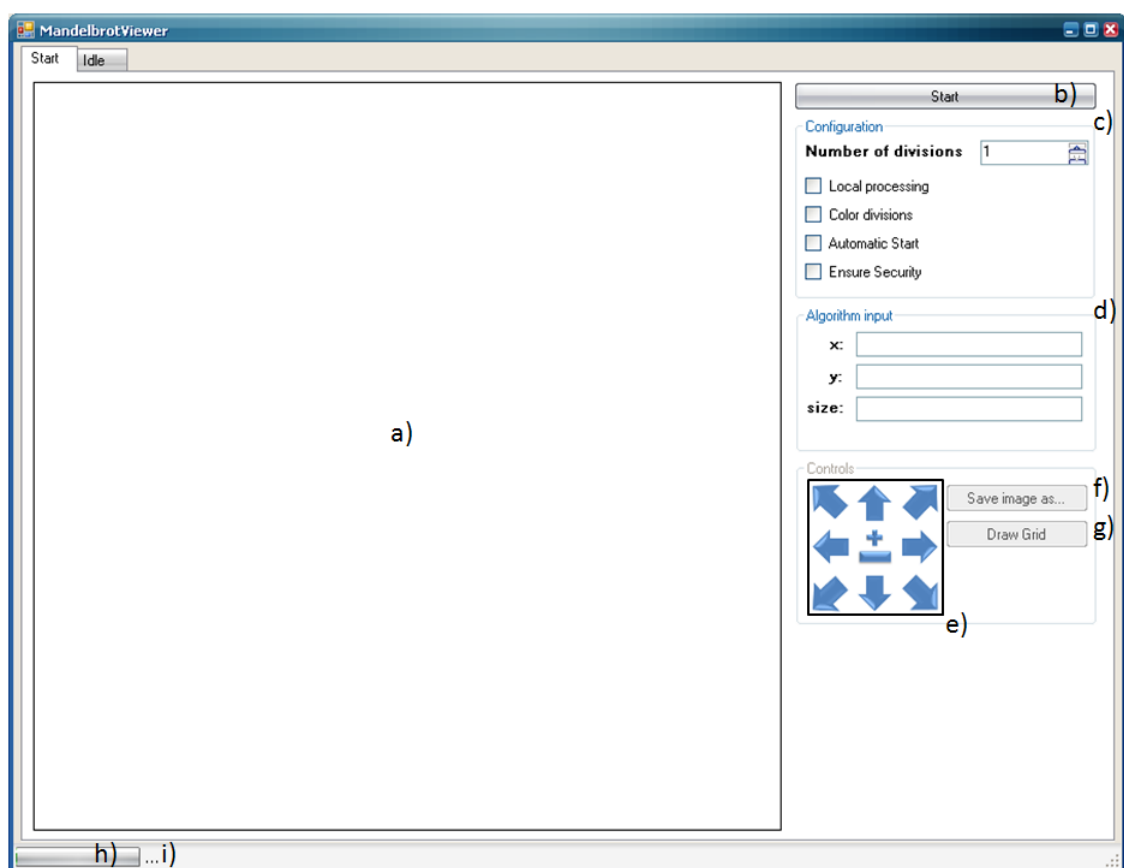


Figura 6.6: Interface da aplicação de demonstração "MandelbrotViewer" (tab 1)

¹⁸Ver indicação f) na figura 6.6

¹⁹Ver indicação g) na figura 6.6

²⁰Ver indicação h) na figura 6.6

²¹Ver indicação i) na figura 6.6

No separador mostrado na 6.7 podemos identificar as seguintes zonas:

- **Título do separador**²²: corresponde ao estado actual do processamento distribuído.
- **Output**²³: zona onde serão impressas mensagens enquanto decorre o processamento distribuído.
- **Botão para limpar o output**²⁴.
- **Estatísticas**²⁵: zona onde são exibidos dados estatísticos relativos às tarefas submetidas para processamento.

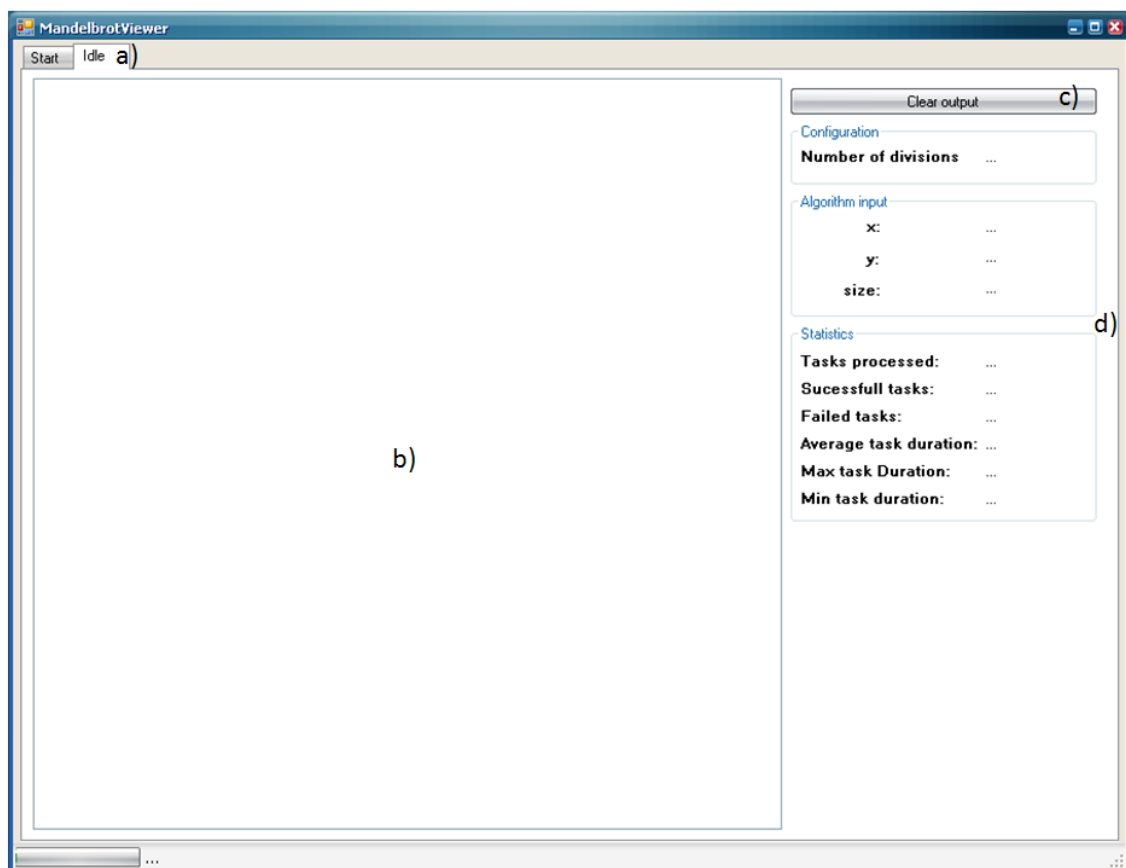


Figura 6.7: Interface da aplicação de demonstração "MandelbrotViewer" (tab 2)

²²Ver indicação a) na figura 6.7

²³Ver indicação b) na figura 6.7

²⁴Ver indicação c) na figura 6.7

²⁵Ver indicação d) na figura 6.7

6.3.2 Exemplos de execução

De seguida serão apontados alguns exemplos de execução, com o objectivo de comprovar o funcionamento do aplicativo de demonstração. No primeiro caso a aplicação foi configurada para que o processamento envolvido no cálculo da imagem fosse local (consultar anexo [B.1](#)). Na primeira figura [B.1](#) é possível observar o decorrer do processamento e respectiva geração da imagem. Na figura seguinte [B.2](#) o processamento foi concluído e pode-se observar que a imagem gerada está completa. No segundo caso a aplicação foi configurada para que o processamento fosse inteiramente distribuído. No anexo [B.2](#) é possível observar uma sequência de imagens que corresponde a este mesmo cenário. As imagens iniciais correspondem ao aspecto do aplicativo de demonstração ([B.3](#) e [B.4](#)) e das interfaces gestoras dos elementos constituintes da Grid ([B.5](#) e [B.6](#)), durante o decorrer do processamento. Por sua vez, as imagens [B.7](#) e [B.8](#) correspondem ao estado do aplicativo de demonstração quando do termo do processamento.

6.4 Análise de resultados

A partir do momento em que o aplicativo de demonstração se encontrava a funcionar em pleno, tanto no cenário local como no cenário distribuído, começou-se a proceder à recolha e análise de resultados. Durante a recolha de resultados, o objectivo foi estabelecer uma comparação entre o processamento local e o distribuído. Para isso, foram efectuadas várias medições com diversos cenários e configurações. As descrições das máquinas envolvidas nos ensaios estão contidas na tabela 6.1. A nomenclatura usada na descrição da configuração foi a seguinte: "(quantidade)(tipo)[(número de núcleos)]". Por exemplo que "3EXE[2]" corresponde a um grupo de três executores com dois núcleos de processamento cada um.

Tabela 6.1: Descrições dos computadores

nome	descrição	CPU	número de núcleos
NOTEBOOK	computador portátil pessoal ASUS A7JA	T700 2.0 Ghz	2
PC	computador pessoal	Q6600 @ 3.0	4
EOLO	computador do INEB	P4HT @ 3.4	1
CALIPSO	computador do INEB	P4HT @ 3.2	1
NOTUS	computador do INEB	PD @ 3.0	2
CLOTO	computador do INEB	P4HT @ 3.0	1
EUROS	computador do INEB	P4HT @ 3.2	1

6.4.1 Ensaio 1

Este ensaio (tabela 6.3) serviu para estabelecer a primeira comparação entre o processamento local e o distribuído. Foi usada a entrada²⁶ que permite uma visualização total do conjunto de Mandelbrot. A configuração usada nos testes distribuídos do presente ensaio, pode ser consultada na tabela 6.2.

Tabela 6.2: Configuração distribuída

nome	aplicativo instalado
EOLO	GridMaster, Client
CALIPSO	GridWorker
NOTUS	GridWorker
CLOTO	GridWorker
EUROS	GridWorker

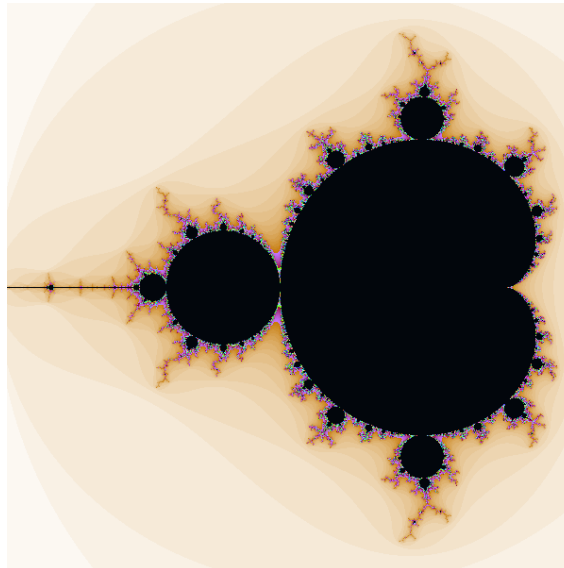


Figura 6.8: Exemplo de *fractal* representativo do conjunto de *Mandelbrot*

O input escolhido gera uma imagem onde o conjunto de Mandelbrot é visível na sua totalidade (figura 6.8). Após observar os resultados obtidos, e também a imagem gerada, pode-se constatar que a divisão não é equilibrada, existindo fatias que exigem um tempo de processamento significativamente superior às restantes. Estas fatias são as que possuem uma maior área a preto, correspondendo portanto às fatias do meio. Este desequilíbrio é facilmente constatável, analisando os testes número 3 e 4 da tabela 6.3.

Apesar dos processadores das máquinas onde foi realizado o ensaio possuírem apenas um núcleo, existe uma ligeira vantagem na atribuição de duas tarefas em simultâneo. O

²⁶ $x=-0.70, y=0, size=2.5$

motivo deste acontecimento está relacionado com o facto do processador possuir *hyper-threading*²⁷, e com o modo como o sistema operativo faz o escalonamento. Embora a diferença seja mínima, para se tirar total partido de um processador com *hyper-threading* é de facto necessário colocá-lo a processar dois pedidos em simultâneo (ver testes 6 e 7, 12 e 15, 13 e 16 da tabela 6.3). Aumentando o número de divisões, as zonas de processamento mais intensivas são igualmente divididas, reduzindo o desperdício de recursos através de uma melhor distribuição do esforço de processamento, reduzindo desta forma o tempo total de processamento.

No caso dos testes 13 e 14, o aumento do número de divisões veio piorar o desempenho (aumentando o tempo final em 7.5 segundos). Tal acontecimento está relacionado com o aumento do *overhead* associado à duplicação dos pedidos, podendo-se concluir que irá sempre existir um ponto a partir do qual deixa de compensar, aumentar o número de divisões.

Tabela 6.3: Ensaio 1

nrº	tipo de teste	configuração	número de divisões	tempo decorrido (secs)
1	local	NOTEBOOK[2]	1	81.0320
2	local	NOTEBOOK[2]	2	43.7820
3	local	PC[4]	1	54.1090
4	local	PC[4]	2	27.4530
5	local	PC[4]	4	25.9530
6	distribuído	2EXE[1]	2	64.1250
7	distribuído	2EXE[2]	4	60.2030
8	distribuído	2EXE[2]	5	58.0930
9	distribuído	2EXE[2]	8	43.5940
10	distribuído	2EXE[2]	10	47.5940
11	distribuído	3EXE[1] + 1EXE[2]	5	61.7500
12	distribuído	3EXE[1] + 1EXE[2]	8	44.2035
13	distribuído	3EXE[1] + 1EXE[2]	16	35.5790
14	distribuído	3EXE[1] + 1EXE[2]	32	43.0780
15	distribuído	4EXE[2]	8	42.8600
16	distribuído	4EXE[2]	16	25.5940

²⁷*hyper-threading*: Tecnologia desenvolvida pela Intel com o objectivo de inovar a computação paralela. Um processador que possua apenas um núcleo, e tenha a *hyper-threading* activa é tratado pelo sistema operativo como se tivesse efectivamente dois núcleos físicos, sendo o peso de processamento dividido pelos dois

6.4.2 Ensaio 2

Este segundo ensaio (tabela 6.4) pretende ser mais equilibrado, estabelecendo uma comparação mais fiel entre o processamento local e distribuído. A única diferença face ao ensaio anterior reside nos dados de entrada²⁸. A região que corresponde a uma imagem equilibrada está localizada precisamente no meio da imagem anterior (ver figura 6.9). Esta imagem embora pouco "atraente" corresponde à situação ideal de teste.

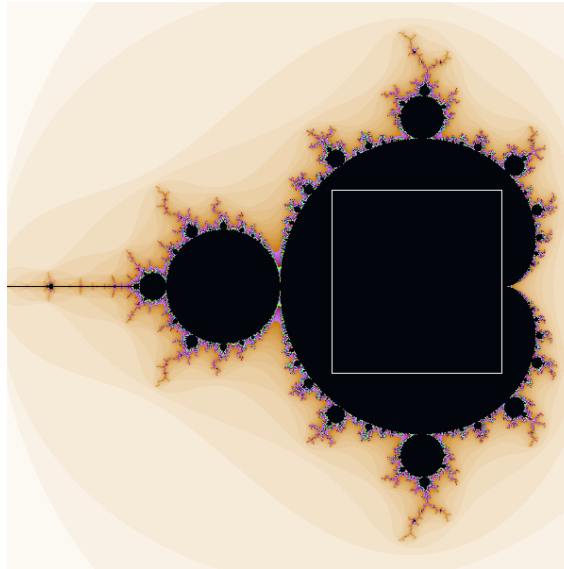


Figura 6.9: Exemplo de *fractal* representativo do conjunto de *Mandelbrot*, com uma região selecionada

Tabela 6.4: Ensaio 2

nr ^o	tipo de teste	configuração	número de divisões	tempo decorrido (segs)
1	local	NOTEBOOK[2]	1	591.5790
2	local	NOTEBOOK[2]	2	302.4220
3	local	PC[4]	1	224.5620
4	local	PC[4]	2	112.5320
5	local	PC[4]	4	58.3290
6	distribuído	4EXE[2]	8	80.5780
7	distribuído	4EXE[2]	16	89.0940
8	distribuído	4EXE[2]	32	93.0780

Devido ao input escolhido torna-se perfeitamente visível a diferença entre o uso de 2 e 4 núcleos (testes número 4 e 5 da tabela 6.4). Para além disto é possível constatar mais cedo a perda de eficiência associada ao aumento do número de divisões (testes 6,7 e 8 da tabela 6.4)

²⁸Sendo que agora corresponde a: $x=-0.175$, $y=-0.008$, $size=0.762$

6.4.3 Ensaio 3

A finalidade do terceiro ensaio consiste em recolher amostras que comprovem a diminuição do tempo de processamento, através do aumento do número de executores. Para obter valores mais precisos, procedeu-se a uma reconfiguração (consultar tabela 6.5). E manteve-se a entrada do ensaio anterior.

Tabela 6.5: Configuração distribuída

nome	aplicativo instalado
EOLO	GridWorker, Client
CALIPSO	GridWorker
NOTUS	GridMaster
CLOTO	GridWorker
EUROS	GridWorker

Tabela 6.6: Ensaio 3

nrº	tipo de teste	configuração	número de divisões	tempo decorrido (secs)	tempo ideal (secs)
1	distribuído	4EXE[1]	1	486.9220	486.9220
2	distribuído	4EXE[1]	2	243.6100	243.4600
3	distribuído	4EXE[1]	3	166.1090	162.3100
4	distribuído	4EXE[1]	4	124.0940	121.7300

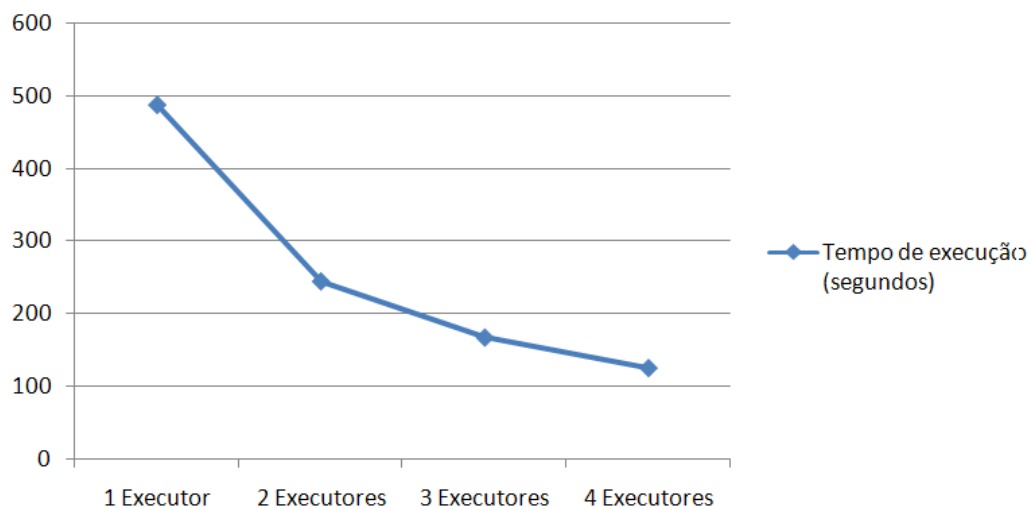


Figura 6.10: Gráfico de variação do tempo em função do número de executores

Analisando o gráfico (figura 6.10), correspondente ao ensaio da tabela 6.6 pode-se facilmente constatar a diminuição do tempo de processamento, quase linear como consequência do aumento do número de executores idênticos (mas não exactamente iguais), como seria de esperar.

Capítulo 7

Conclusões e Trabalho Futuro

De um modo geral pode-se afirmar que o projecto satisfaz os requisitos inicialmente previstos e, como tal, cumpre em muito boa medida os objectivos. Surgiram apenas alguns problemas ocasionais de conectividade e uma pequena falha na autenticação. O problema da autenticação (detectado um pouco tarde), levou a que fosse necessário alterar uma propriedade do sistema operativo¹. Apesar de muito provavelmente não corresponder à solução ideal, decidiu-se a curto prazo, simplesmente alterar esta propriedade. Este pequeno problema nunca impediu o desenvolvimento e teste do sistema, não sendo, portanto, muito relevante.

Os resultados obtidos durante o teste (distribuído) do aplicativo de demonstração foram satisfatórios e corresponderam às expectativas. Assim, foi possível comprovar a eficácia e usabilidade da *framework*.

7.1 Acordo com o planeamento

O projecto decorreu praticamente como se tinha inicialmente previsto. A fase que se revelou um pouco mais complicada e morosa foi, sem dúvida, a fase de testes. Tal facto, vem indiscutivelmente associado à própria natureza do sistema distribuído, sempre complicado de testar e de se ter uma visão de conjunto.

¹ControlPanel\AdministrativeTools\LocalSecuritySettings: SecurityOptions, Set the "Network access: Sharing and security model for local accounts" to "classic - local users authenticate as themselves".

7.2 Trabalho futuro

Embora o projecto tenha cumprido os objectivos inicialmente propostos, a sua natureza tolera a existência de uma imensidão de alterações e melhoramentos. Os mais plausíveis de serem realizados a curto prazo incidem sobre as limitações actuais do sistema.

Actualmente, o sistema apenas funciona em máquinas com plataforma *.NET* e sistema operativo da Microsoft. Uma possibilidade seria migrar o sistema para a plataforma *Mono*², possibilitando assim o seu funcionamento em sistemas baseados em Linux, abrangendo deste modo, um maior número de plataformas

Uma outra limitação reside no modo como a informação associada aos pedidos é transmitida para a grelha. Toda a informação necessária para o processamento de um dado pedido é serializada, com o próprio, e enviado ao servidor, como um pacote só. Esta metodologia é relativamente simples, e viável de ser usada nas situações em que a quantidade de informação não é muito elevada. Se assim não for, a performance global do sistema sairá penalizada em grande parte devido às serializações e reconstruções que ocorrem, no encaminhamento dos pedidos, desde o cliente (API) até o executor (GridWorker). Uma forma de evitar o uso destes procedimentos será permitir o acesso directo ao sistema de ficheiros da máquina (cliente), onde foi submetido o pedido. No entanto, a concretização desta ideia não é propriamente simples. Para além disto, levantam-se algumas questões a nível da interoperabilidade e segurança. A situação ideal seria conceber algo semelhante ao sistema de ficheiros distribuído do projecto [Ali09]. Este tipo de metodologia apesar de mais sofisticada, é significativamente mais difícil de desenvolver e requer uma programação mais explícita, por parte do programador (quando do uso da *API*).

Uma funcionalidade que iria permitir que o projecto fosse aplicável a um maior número de situações, seria desenvolver um mecanismo para a troca de informação entre os clientes executores. Deste modo, o presente sistema poderia ser usado para distribuir o processamento de algoritmos que após subdivisão dão origem a tarefas dependentes. Em termos tecnológicos, a troca de mensagens é efectivamente possível, bastando para isso usar o mesmo mecanismo que o GridMaster usa para notificar o GridWorker. O maior entrave seria conceber uma metodologia (do lado da API) que permitisse ao programador definir (de forma simples e transparente) a comunicação entre tarefas, necessária em grande parte da vezes para sincronização.

²*Mono*: Conjunto de ferramentas compatíveis com *.NET* para sistemas operativos baseados em Linux

Referências

- [20009] BEinGRID 2008. Gridpedia, Março 2009. Consultado em <http://www.gridipedia.eu>.
- [Aks09] Akshay Luther and Rajkumar Buyya and Rajiv Ranjan and Srikumar Venugopal, Grid Computing and Distributed Systems (GRIDS) Laboratory and Department of Computer Science and Software Engineering and The University of Melbourne in Australia . Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids . Março 2009. Consultado em <http://www.gridbus.org/papers/Alchemi.pdf>.
- [Ali09] Alireza Poshtkuhi and Ali Haj Abutalebi and Leila Mahmoudi Ayough, Shaahin Hessabi and Islamic Azad University - Qazvin Branch and Microelectronics Research & Design Center of IRAN and Dept of Computer Engineering Sharif University of Technology. DotGrid: A .NET-based Infrastructure for Global Grid Computing . Maio 2009. Consultado em <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01630951>.
- [And09] Andrey Y Shevel. Cluster computing, Março 2009. Consultado em <http://www.clustergate.ru/>.
- [BEi09] BEinGRID. BEinGRID, Março 2009. Consultado em <http://www.beingrid.eu/about.html>.
- [Bos00] Guus Bosman. CORBA & COM+. Fevereiro 2000. Consultado em <http://www.guusbosman.nl/downloads/PaperOop.html>.
- [cGp09a] EC cofunded GridTalk project. Grid computing, Março 2009. Consultado em <http://www.gridcafe.org/what-is-the-grid.html>.
- [cGp09b] EC cofunded GridTalk project. Gridcafe, Maio 2009. Consultado em <http://www.gridcafe.org>.
- [Cor09a] Microsoft Corporation. COM, Junho 2009. Consultado em <http://www.microsoft.com/com/default.aspx>.
- [Cor09b] Microsoft Corporation. COM+, Junho 2009. Consultado em [http://msdn.microsoft.com/en-us/library/ms685978\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms685978(VS.85).aspx).
- [Cor09c] Microsoft Corporation. DCOM, Junho 2009. Consultado em <http://msdn.microsoft.com/en-us/library/ms809340.aspx>.

REFERÊNCIAS

- [Cor09d] Microsoft Corporation. Format for .NET Remoting Configuration Files, Março 2009. Consultado em http://msdn.microsoft.com/en-us/library/ms973907.aspx#remotingconfig_topic14x.
- [Cor09e] Microsoft Corporation. Internet information services, Junho 2009. Consultado em <http://learn.iis.net/>.
- [Cor09f] Microsoft Corporation. MTS, Junho 2009. Consultado em [http://technet.microsoft.com/pt-pt/library/cc751038\(en-us\).aspx](http://technet.microsoft.com/pt-pt/library/cc751038(en-us).aspx).
- [Cor09g] Microsoft Corporation. .NET Framework, Junho 2009. Consultado em [http://msdn.microsoft.com/pt-pt/library/zw4w595w\(en-us\).aspx](http://msdn.microsoft.com/pt-pt/library/zw4w595w(en-us).aspx).
- [Cor09h] Microsoft Corporation. .NET Remoting, Junho 2009. Consultado em [http://msdn.microsoft.com/en-us/library/kwdt6w2k\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx).
- [Cor09i] Microsoft Corporation. Reflection, Junho 2009. Consultado em <http://msdn.microsoft.com/en-us/library/f7ykdhsy.aspx>.
- [Cor09j] Microsoft Corporation. Remoting example: Callcontext, Março 2009. Consultado em [http://msdn.microsoft.com/en-us/library/6z5ezzh0\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/6z5ezzh0(VS.71).aspx).
- [DE02] Priya Dhawan e Tiim Ewald. ASP .NET Web Services or .NET Remoting How to Choose. Setembro 2002.
- [Dev08] Developed by Pluralsight LLC, Microsoft Corporation. Sequential to Parallel HPC Development. 2008.
- [dn09] dev newsgroups. .NET Remoting TCP or HTTP?, Março 2009. Consultado em <http://www.devnewsgroups.net/group/microsoft.public.dotnet.framework/topic8761.aspx>.
- [EGE09] EGEE Project. Enabling Grids for E-science, Março 2009. Consultado em <http://project.eu-egee.org/index.php?id=104>.
- [Gri09a] Distributed computing info, Março 2009. Consultado em <http://distributedcomputing.info>.
- [Gri09b] Distributed computing info, Março 2009. Consultado em <http://distributedcomputing.info/platforms.html>.
- [Gri09c] Distributed computing info, Março 2009. Consultado em <http://distributedcomputing.info/projects.html>.
- [Gri09d] Euro grid, Maio 2009. Consultado em <http://www.eurogrid.org/>.
- [Gri09e] Interactive european grid project, Maio 2009. Consultado em <http://www.lip.pt/computing/projects/i2g/>.

REFERÊNCIAS

- [Gri09f] Worldwide LHC Computing Grid, Maio 2009. Consultado em <http://lcg.web.cern.ch/LCG/>.
- [Gro09] WC3 Working Group. WS, Junho 2009. Consultado em <http://www.w3.org/TR/ws-arch/>.
- [Jav09] Java-Virtual-Machine.net. Java virtual machine, Junho 2009. Consultado em <http://java-virtual-machine.net/sun-java-virtual-machine.html>.
- [ISU09] Pande lab Stanford University. Folding, Março 2009. Consultado em <http://folding.stanford.edu/English/Main>.
- [oC09] University of California. Seti, Março 2009. Consultado em http://setiathome.berkeley.edu/sah_about.php.
- [OMG09a] OMG - Object Managment Group. CORBA, Junho 2009. Consultado em <http://www.omg.org/gettingstarted/corbafaq.htm>.
- [OMG09b] OMG - Object Managment Group. OMG, Junho 2009. Consultado em <http://www.omg.org/>.
- [Onl09a] Encyclopædia Britannica Online. Computação distribuída, Maio 2009. Consultado em <http://www.britannica.com/EBchecked/topic/1494997/distributed-computing>.
- [Onl09b] Encyclopædia Britannica Online. Compute, Maio 2009. Computing basics, Consultado em <http://www.britannica.com/EBchecked/topic/130429/computer>.
- [Onl09c] Encyclopædia Britannica Online. Computer, Maio 2009. Consultado em <http://www.britannica.com/EBchecked/topic/130429/computer>.
- [Onl09d] Encyclopædia Britannica Online. Mandelbrot set, Junho 2009. Consultado em <http://www.britannica.com/EBchecked/topic/958557/Mandelbrot-set>.
- [Onl09e] Encyclopædia Britannica Online. Numerical analysis, Junho 2009. Consultado em <http://www.britannica.com/EBchecked/topic/422388/numerical-analysis>.
- [Onl09f] Encyclopædia Britannica Online. Sistemas distribuídos, Maio 2009. Consultado em <http://www.britannica.com/EBchecked/topic/166154/distributed-system>.
- [Onl09g] Encyclopædia Britannica Online. Super computer, Maio 2009. Computing basics » Digital computers » Supercomputer, Consultado em <http://www.britannica.com/EBchecked/topic/130429/computer>.
- [Onl09h] Encyclopædia Britannica Online. TCP/IP, Maio 2009. Consultado em <http://www.britannica.com/EBchecked/topic/602945/TCPIP>.

REFERÊNCIAS

- [Onl09i] Encyclopædia Britannica Online. WAN, Maio 2009. Consultado em <http://www.britannica.com/EBchecked/topic/643173/wide-area-network>.
- [Pet09] Peter A. Bromberg. Message Queue over HTTP: Usage Scenarios , Março 2009. Consultado em <http://www.eggheadcafe.com/articles/20040620.asp>.
- [Ric02] Jeffrey Richter. *Applied Microsoft .NET Framework Programming*. Microsoft Press A Division of Microsoft Corporation One Microsoft Way Redmond Washington 98052-6399 , 2002.
- [RS02] Ingo Rammer e Mario Szpuszta. *Advanced .NET Remoting*. Microsoft Press A Division of Microsoft Corporation One Microsoft Way Redmond Washington 980526399 , 2002.
- [SM09a] Inc. Sun Microsystems. Java, Junho 2009. Consultado em <http://pt.sun.com/java/about/>.
- [SM09b] Inc. Sun Microsystems. Java RMI, Junho 2009. Consultado em <http://java.sun.com/docs/books/tutorial/rmi/overview.html>.
- [STE00] STEFANOS PETROU 2000/2001 UK, London. CORBA vs COM+. 2000.
- [UoC09] Berkeley University of California. Boinc, Maio 2009. Consultado em <http://boinc.ssl.berkeley.edu/>.
- [WC309] WC3 Working Group. Message-digest algorithm 5, Junho 2009. Consultado em http://www.w3.org/TR/1998/REC-DSig-label/MD5-1_0.
- [Woo09] Nathan Wood. Java RMI vs .NET Remoting. Janeiro 2009. Consultado em <http://www.geocities.com/giantstonesoldier/SE325Assignment2Nwoo050.pdf>.

Anexo A

Extractos de código

A.1 Método Timeout

O trecho de código que se segue pertence ao GridMaster.

```
(...)
private void timeoutThreadMethod()
{
try \\first try
{
Hashtable requestsToAbortIDs = null;
List<String> workersToPingIDs = null;
List<String> clientsToPingIDs = null;
while (true)
{
Thread.Sleep(timeoutPoolingSeconds*1000);
print("timeoutThreadMethod called");
//analyze if any processing request of any worker has timedout
lock (clients)
{
clientsToPingIDs = new List<String>();
workersToPingIDs = new List<String>();
requestsToAbortIDs = new Hashtable();
foreach (ClientRegistry client in clients.Values)
{
lock (client.Requests)
{
bool wentOff = false;
try \\second try
{
foreach (String key in client.Requests.Keys)
{
if (wentOff)
{
if (!requestsToAbortIDs.Contains(key))
```

Extractos de código

```
        requestsToAbortIDs.Add(key, "x");
        continue;
    }

    try \\third try
    {
        Request requestData = (Request)client.Requests[key];
        if (requestData.Dob == requestData.ProcessingStarted)
            continue;
        //see if request timed out
        DateTime now = DateTime.Now;
        DateTime timeoutDateTime = DateTime.Now;
        if (requestData.ProgrammerTimeoutInSeconds == -1)
            timeoutDateTime =
                requestData.ProcessingStarted.AddSeconds(
                    defaultTimeoutInSeconds);

        else
            timeoutDateTime =
                requestData.ProcessingStarted.AddSeconds(
                    requestData.ProgrammerTimeoutInSeconds);

        if (
            ((now >= timeoutDateTime) &&
             (requestData.State == RequestState.Processing))
            ||
            (requestData.State == RequestState.Canceling))
        {
            requestsToAbortIDs.Add(key, requestData.WorkerID);
        }
    } \\third try over
    catch (Exception)
    {
        wentOff = true;
        if (!requestsToAbortIDs.Contains(key))
            requestsToAbortIDs.Add(key, "x");
        clientsToPingIDs.Add(client.ClientID);
        print("timeoutThreadMethod EXCEPTION,
            request holder went off during timeout, will abort... ");
    }
}

} \\second try over
catch(Exception)
{
    print("timeoutThreadMethod EXCEPTION,
        request holder went off during timeout, will abort... ");
}
```

Extractos de código

```
    }
  }
  //ping necessary clients
  foreach (String idToPing in clientsToPingIDs)
  {
    if (clients.ContainsKey(idToPing))
      pingClient(idToPing);
  }

  foreach (String requestToAbortID in requestsToAbortIDs.Keys)
  {
    if (requestsToAbortIDs[requestToAbortID].ToString() != "x")
    {
      GridWorkerRegistry worker = null;
      try //fourth try
      {
        worker = ((GridWorkerRegistry)workers[
          requestsToAbortIDs[
            requestToAbortID]]);
        worker.SendNotification(
          new NotificationEventArgs(
            NotificationType.Abort,
            "ABORT",
            ClientType.GridWorker,
            requestToAbortID));
      } //fourth try over
      catch (Exception)
      {
        print("timeoutThreadMethod EXCEPTION worker
          holding the requests to abort went off");
        if (!workersToPingIDs.Contains(
          requestsToAbortIDs[requestToAbortID].ToString()))
          workersToPingIDs.Add(
            requestsToAbortIDs[requestToAbortID].ToString());
      }
    }
  }
  else
  {
    lock (workers)
    {
      foreach (String key in workers.Keys)
      {
        try //fifth try
        {
          GridWorkerRegistry worker =
            ((GridWorkerRegistry)workers[key]);
          worker.SendNotification(
            new NotificationEventArgs(
```

Extractos de código

```
NotificationType.Abort,
"ABORT",
ClientType.GridWorker,
requestToAbortID));
} //fifth try over
catch (Exception)
{
    print("timeoutThreadMethod EXCEPTION worker
    holding the requests to abort went off");
    if (!workersToPingIDs.Contains(key))
        workersToPingIDs.Add(key);
}
}
}
}
lock (workers)
{
    //ping necessary clients
    foreach (String idToPing in workersToPingIDs)
        if (workers.ContainsKey(idToPing))
            pingWorker(idToPing);
}
}
catch (Exception e)
{
    print("timeoutThreadMethod - UNEXPECTED EXCEPTION: " + e.Message);
}
}
(...)
```

A.2 Exemplo de uso da API

O trecho de código que se segue corresponde a um exemplo de uso da API.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Runtime.Serialization;
using AuxiliarProgrammerAssembly;
using AuxiliarProgrammerAssemblyAux;
using ProgrammerSpecial;

using API;

namespace ProgrammerProject
{

    public class ProgrammerClass : Worker
    {
        public int[] array;
        public int a;
        public int b;
        public int c;

        public ProgrammerClass()
        {
            array = new int[3];
            this.a = 2;
            this.b = 3;
            this.c = 1;
        }
        public ProgrammerClass(int a, int b, int c)
        {
            array = new int[3];
            this.a = a;
            this.b = b;
            this.c = c;
        }

        // "heavy" method
        public override void Work()
        {
            AuxiliarProgrammerAssembly.ProgrammerHelpClass help
            = new ProgrammerHelpClass();
            int result = help.sum(2500, 2500);

            // heavy function
        }
    }
}
```

Extractos de código

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("runing {0}", i);
    Thread.Sleep(1000);
}

c = a + b;
array[0] = a;
array[1] = b;
array[2] = c;
}
}

public class Program
{
    public class CallbackClass
    {
        public int taskCounter;
        public int taskTotal;

        public CallbackClass()
        {
            taskCounter = 0;
            taskTotal = 0;
        }

        //method to process the result
        public void myCallBack(TaskCompletedEventArgs e)
        {
            //count the number of results
            taskCounter++;
            Console.WriteLine();

            //check for error's, and react acoording to the content
            if (e.CompletedTaskState == TaskState.Error)
            {
                ProgrammerClass myResult =
                    (ProgrammerClass)e.CompletedTaskObject;
                Console.WriteLine("({6})Task Finished Bad -
a:{0} b:{1} c:{2} -> ({3}) [{4}/{5}]",
myResult.a, myResult.b, myResult.c,
e.Message, taskCounter, taskTotal,
e.TaskProcessingDuration.TotalSeconds.ToString("F0"));
            }
            else if (e.CompletedTaskState == TaskState.Waiting)
            {
                Console.WriteLine("CallBack Received but Task still
```

Extractos de código

```
        \ "Waiting\ "??");
    }
    else if (e.CompletedTaskState == TaskState.Finished)
    {
        if (e.TaskObjectType.Name == "ProgrammerClass")
        {
            ProgrammerClass myResult =
                (ProgrammerClass)e.CompletedTaskObject;
            Console.WriteLine("{6})Task Finished Ok -
a:{0} b:{1} c:{2} -> ({3}) [{4}/{5}]",
myResult.a, myResult.b, myResult.c,
e.Message, taskCounter, taskTotal,
e.TaskProcessingDuration.TotalSeconds.ToString("F0"));
        }
        else if (e.TaskObjectType.Name == "ProgrammerClassSpecial")
        {
            ProgrammerClassSpecial myResult =
                (ProgrammerClassSpecial)e.CompletedTaskObject;
            Console.WriteLine("{6})Task Finished Ok -
a:{0} b:{1} c:{2} -> ({3}) [{4}/{5}]",
myResult.a, myResult.b, myResult.c,
e.Message, taskCounter, taskTotal,
e.TaskProcessingDuration.TotalSeconds.ToString("F0"));
        }
    }
}

static void Main()
{
    //Configuration, connection and registry
    Console.WriteLine("ProgrammerClass Main Started");
    Grid grid = new Grid(
        "127.0.0.1", "127.0.0.1", "60045",
        "bob", "bob", true);
    grid.Connect();

    //Check if everything is Ok
    Console.WriteLine("grid.IsConnected: " + grid.IsConnected);
    Console.WriteLine("All Ok?: {0}", grid.AllOk);

    //Proceed only if all ok
    if (grid.AllOk)
    {
        //Job definition
        Job myJob = new Job();
        CallbackClass callClass = new CallbackClass();
    }
}
```

Extractos de código

```
myJob.taskCompleted += callClass.myCallBack;

//Task definition
callClass.taskTotal = 50;
for (int i = 0; i < callClass.taskTotal; i++)
    myJob.Tasks.Add(
        new Task(
            15,
            typeof(ProgrammerClass),
            new ProgrammerClass(i, 2, 1)));

Task taskToCancel = new Task(
    15,
    typeof(ProgrammerClass),
    new ProgrammerClass(i, 2, 1))

myJob.Tasks.Add(taskToCancel);

//Job submission and waiting
grid.submitJob(myJob);
Console.WriteLine("Programmer program Waiting");
grid.wait(myJob);

//Cancelation
//grid.cancelTask(taskToCancel); //task cancelation
//grid.cancelJob(myJob); //job cancelation grid.cancelAllJobs
//grid.cancelAllJobs(); //all jobs cancelation
}

//Disconnect
grid.Disconnect();
Console.WriteLine("Programmer program Over");
Console.ReadLine();

//PROGRAM OVER
}
}
}
```

A.3 Algoritmo Mandelbrot

O trecho de código que se segue corresponde ao algoritmo para o cálculo dos *pixels* constituintes de uma imagem representativa de uma região do conjunto de Mandelbrot. O código que se segue foi retirado dos projecto anexos ao documento [Dev08].

```
// Returns a color reflecting the value of the
//Mandelbrot set element at this position.

private int MandelbrotColor(
double yp, double xp, double y, double x, double size, int pixels)
{
    //compute pixel position:
    double ypos = y + size * (yp - pixels / 2) / ((double)pixels);
    double xpos = x + size * (xp - pixels / 2) / ((double)pixels);

    y = ypos;
    x = xpos;

    double y2 = y * y;
    double x2 = x * x;

    int color = 1;

    const int MAXCOLOR = 69887; // affects rendering color

    //This magic number happens to produce a colour
    //approximating black with my colour picker calculation.
    //It also makes things pretty slow, which is handy.

    //Repeat until we know pixel is not in Mandelbrot set,
    //or until we have reached max # of iterations,
    //in which case pixel is probably in the set.
    //In the latter, color will be black.

    while ((y2 + x2) <= 4 && color < MAXCOLOR)
    {
        y = 2 * x * y + ypos;
        x = x2 - y2 + xpos;

        y2 = y * y;
        x2 = x * x;

        color++;
    }

    return color;
}
```

Anexo B

Exemplos de execução

B.1 Exemplo de execução local

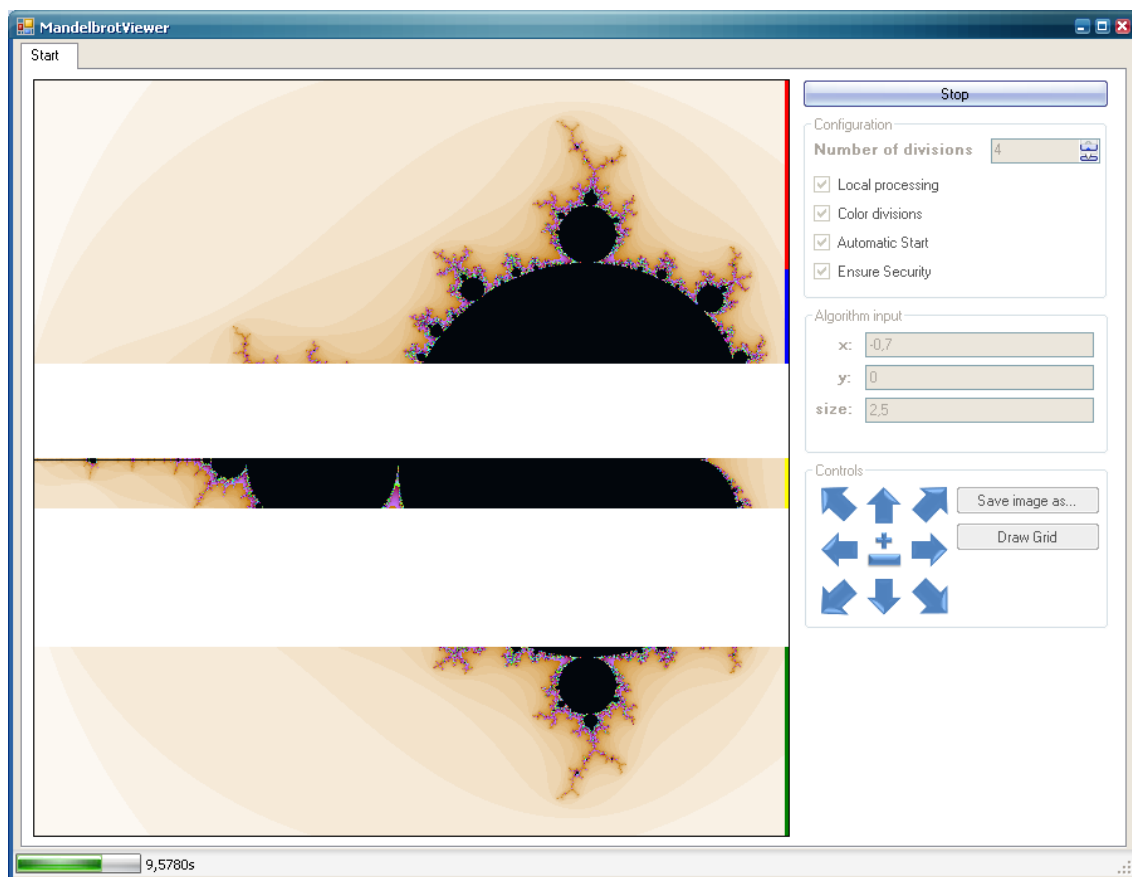


Figura B.1: Exemplo de execução da aplicação MandelbrotViewer aquando a geração da imagem

Exemplos de execução

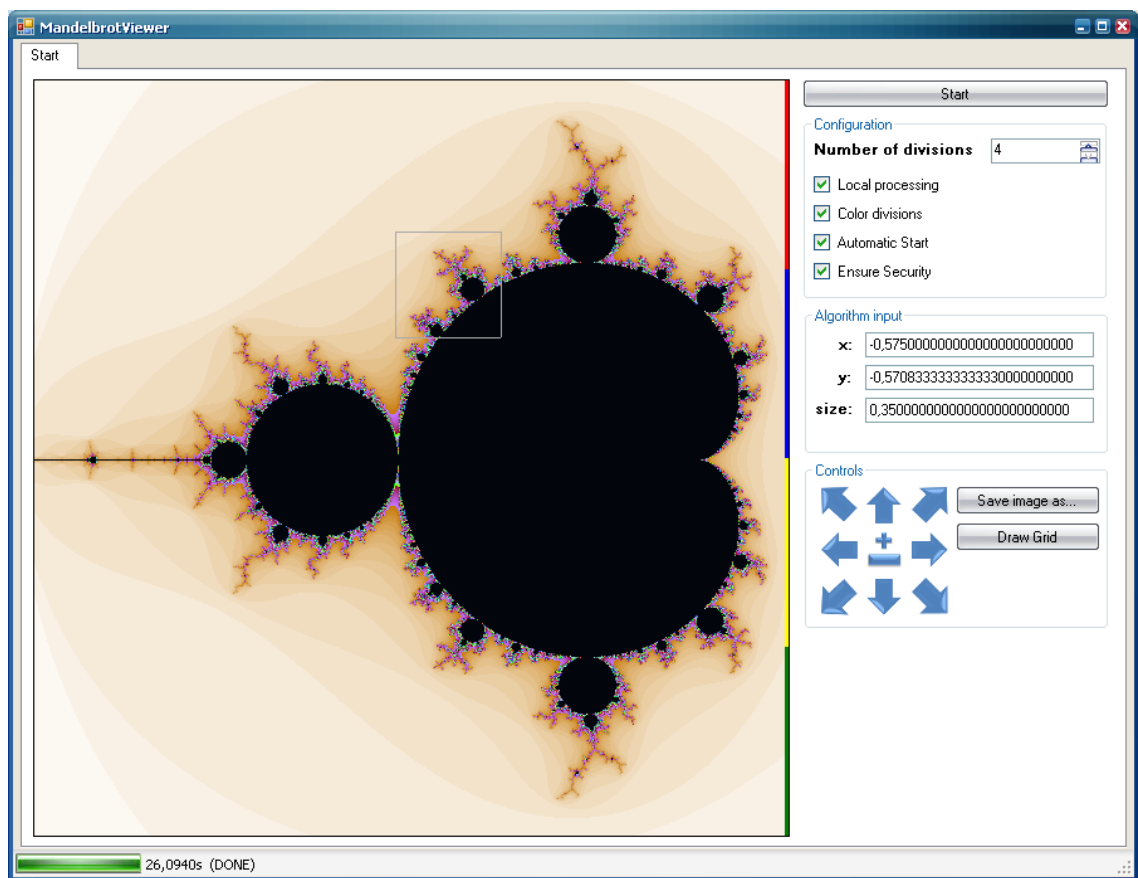


Figura B.2: Exemplo de execução da aplicação MandelbrotViewer após a geração da imagem. (A imagem contém um exemplo de selecção usando o rato)

B.2 Exemplo de execução distribuída

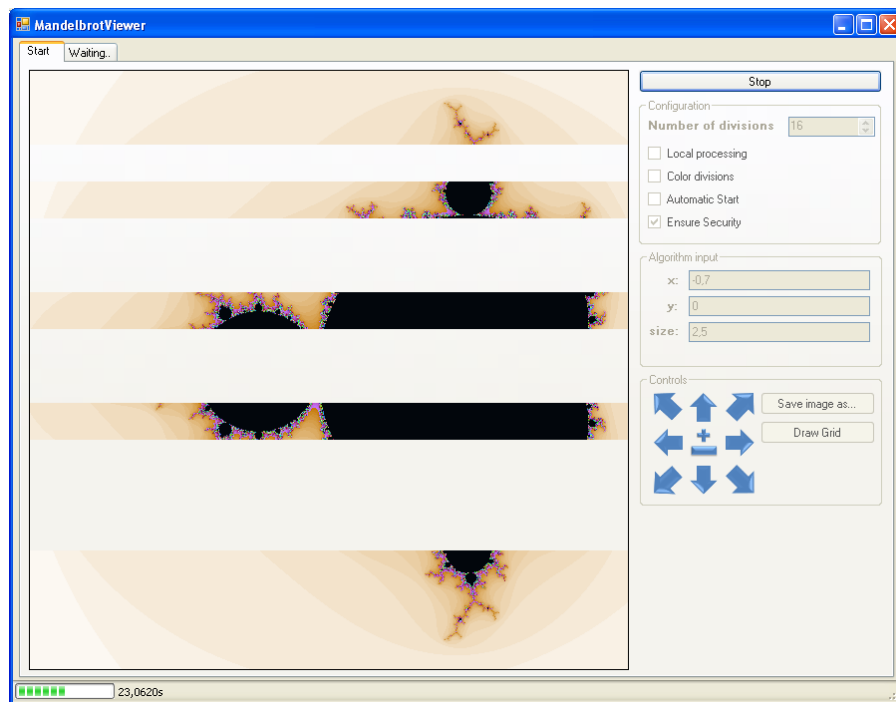


Figura B.3: Aspecto da aplicação MandelbrotViewer enquanto gera a imagem (tab 1)

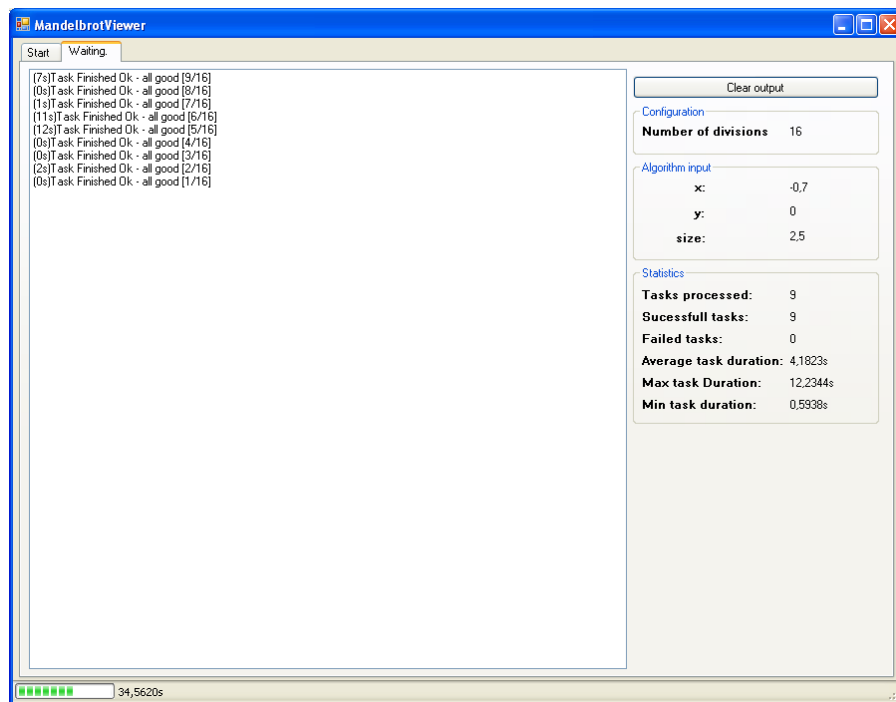


Figura B.4: Aspecto da aplicação MandelbrotViewer enquanto gera a imagem (tab 2)

Exemplos de execução

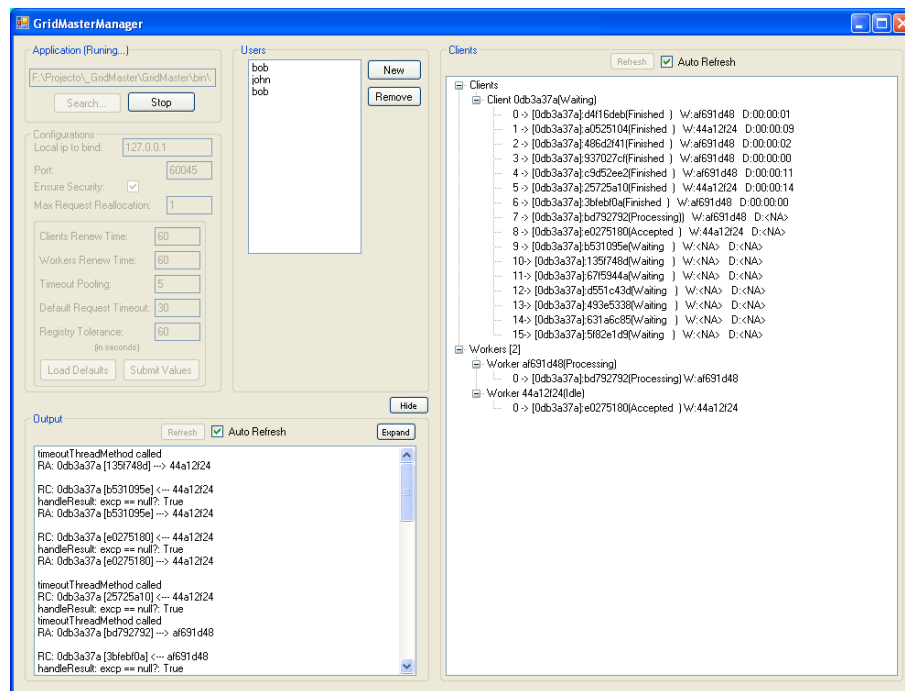


Figura B.5: Exemplo de execução da aplicação GridMasterManager

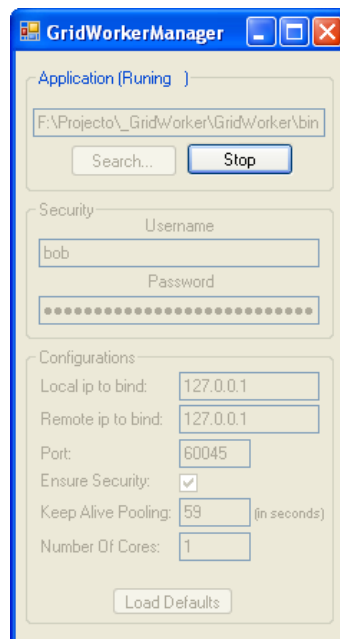


Figura B.6: Exemplo de execução da aplicação GridWorkerManager

Exemplos de execução

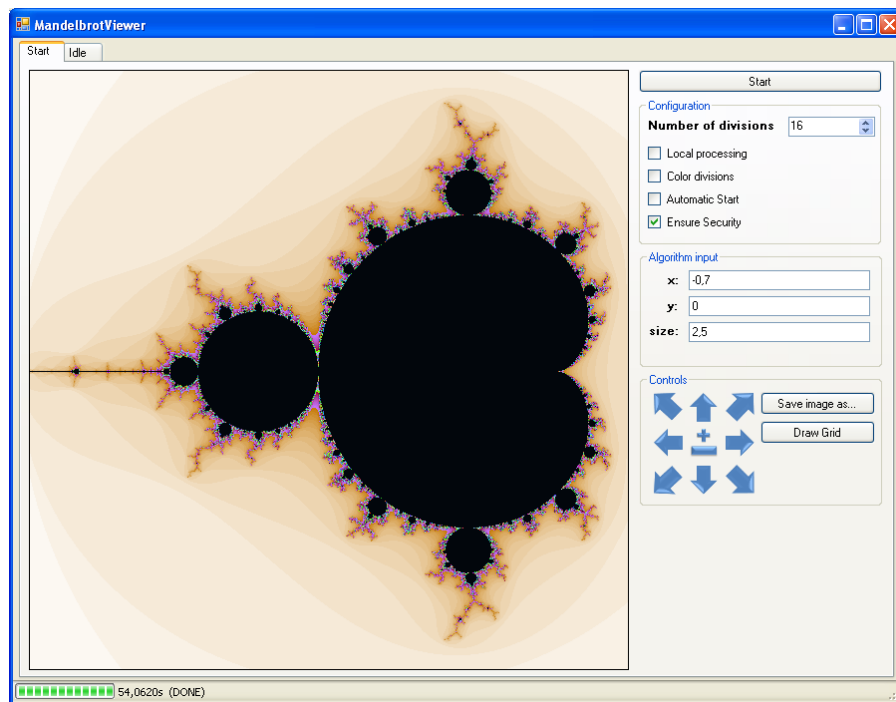


Figura B.7: Aspecto da aplicação MandelbrotViewer após gerar a imagem (tab 1)

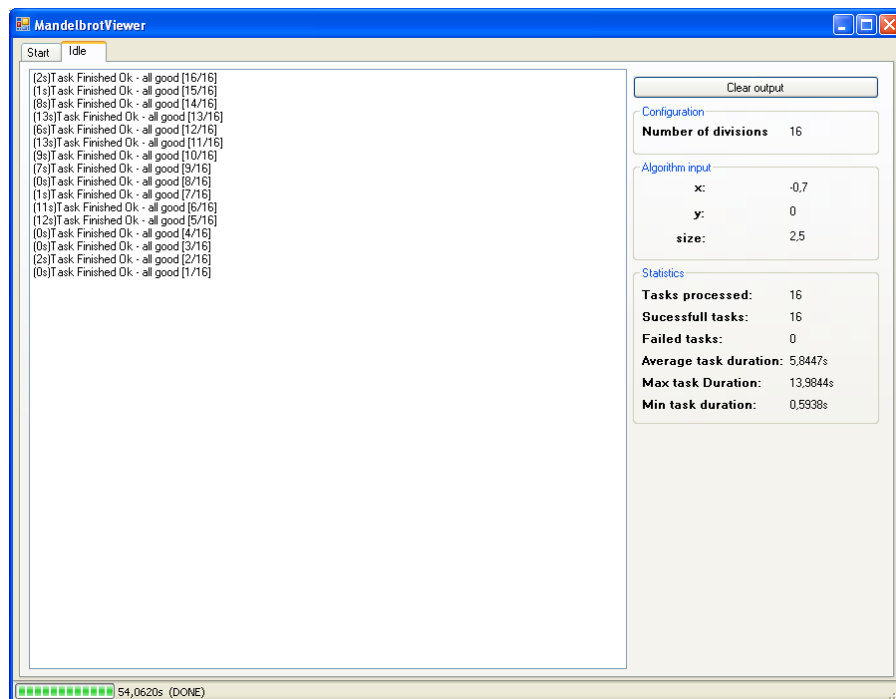


Figura B.8: Aspecto da aplicação MandelbrotViewer após gerar a imagem (tab 2)