ROSSINI PENA ABRANTES

METODOLOGIA PARA PROCESSAMENTO DISTRIBUÍDO COM APLICAÇÃO EM REPRESENTAÇÃO POLINOMIAL DE SUPERFÍCIES

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

VIÇOSA
MINAS GERAIS - BRASIL
2008

ROSSINII PENA ABRANTES

METODOLOGIA PARA PROCESAMENTO DISTRIBUÍDO COM APLICAÇÃO COM REPRESENTAÇÃO POLINOMIAL DE SUPERFICIES

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae*.

APROVADA: 28 de agosto de 2008

Carlos de Castro Goulart
(co-orientador)

Brauliro Gonçalves Leal
(co-orientador)

Ricardo dos Santos Ferreira

Carlos Frederico M. da C. Cavalcanti

Leacir Nogueira Bastos
(Orientador)

Dedico este trabalho a todos que contribuíram para a sua realização.

AGRADECIMENTOS

- A Deus, que em sua infinita bondade, dotou-me de inteligência e do desejo de aprimoramento como ser humano e como profissional.

- Aos meus pais, Sônia e Armando, pelo apoio e incentivo incondicional na realização de mais esta etapa na minha vida.

- À Renata, minha namorada e companheira de todas as horas, pela paciência e persistência demonstrada durante o período em que ficávamos distantes e pelas alegrias no período em que estávamos juntos.

- Aos meus irmãos, Armando e Caroline, pela amizade, pelos momentos de descontração e pela torcida sincera pelo meu sucesso pessoal e profissional.

- Ao Brauliro, pelos seus sábios conselhos e opiniões desde a época da graduação e que foram fundamentais no mestrado e pelas várias conversas esclarecedoras que tivemos.

- Ao Leacir pelos direcionamentos primordiais para que este trabalho pudesse se realizar.

- Aos companheiros de república (atuais moradores e aos que por lá passaram), é impossível esquecer a animação das horas vagas fazendo delas momentos de alegria e descontração, ajudando-me a preencher a lacuna deixada pela ausência da família.

- Ao professor Carlos Goulart, pelas orientações que durante este trabalho impulsionaram a dinâmica do pensamento produtivo.

- Ao professor José Luis Braga, pelas conversas que tivemos durante este período e pelos conselhos de poucas palavras e muito significado e importância.

- Ao Altino Alves pela paciência e dedicação que vão além do simples profissionalismo.

- Ao professor Jugurta Lisboa Filho pela iniciativa e apoio na execução deste projeto.

- A todos os professores do DPI que diretamente ou indiretamente contribuíram para esta vitória.

- Aos colegas de mestrado pelo apoio e companheirismo nas horas difíceis e pela parceria nos momentos de lazer.

- Aos amigos de Viçosa pelos bons momentos que passamos juntos.

- À Universidade Federal de Viçosa pela oportunidade de concretizar este desejo e pelo suporte durante este período.

- À CAPES por possibilitar que este trabalho fosse desenvolvido com dedicação e comprometimento.

BIOGRAFIA

Rossini Pena Abrantes, filho de Armando Vieira Abrantes e Sônia Maria Pena Abrantes, brasileiro, nascido em 6 de junho de 1983 na cidade de Teófilo Otoni, Minas Gerais.

Cursou na cidade de Pote, estado de Minas Gerais, o ensino fundamental, onde residiu até aos 14 anos. Em 1997 mudou-se para a cidade de Governador Valadares, onde finalizou o ensino médio e se graduou como Bacharel em Ciência da Computação pela Universidade Vale do Rio Doce - UNIVALE no ano de 2005.

Em 2006 se mudou para Viçosa, onde fez parte do programa de mestrado no Departamento de Informática – DPI da Universidade Federal de Viçosa – UFV. Tornou-se mestre em Sistemas Distribuídos, defendendo esta dissertação em agosto de 2008.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

ACM	Association for Computing Machinery
API	Application Program Interface
AES	Advanced Encryption Standard
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
IP	Internet Protocol
LAN	Local Area Network
MAN	Metropolitan Area Network
MDE	Modelo Digital de Elevação
MNT	Modelo Numérico de Terreno
RMI	Remote Method Invocation
RPC	Request For Comments
SAS	Service Architecture System
WAN	Wide Area Network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Model Language
XML	Extensible Markup Language

RESUMO

ABRANTES, Rossini Pena, M.Sc., Universidade Federal de Viçosa, Agosto de 2008. **Metodologia para Processamento Distribuído com Aplicação em Representação Polinomial de Superfícies.** Orientador: Leacir Nogueira Bastos. Co-Orientadores: Brauliro Gonçalves Leal e Carlos de Castro Goulart.

Pesquisas em diversas áreas do conhecimento estão envolvendo cada vez mais informações a serem processadas e demandam mais poder computacional para processar tais informações. Por esta razão, a área de Processamento Distribuído é uma área fundamental por possibilitar que grandes demandas de poder de processamento sejam supridas. Com base neste fato, o presente trabalho propõe uma metodologia para distribuição de processamento através da Internet utilizando computação voluntária. A distribuição de processamento através da Internet possibilita que o processamento seja distribuído para um grande número de máquinas. A computação voluntária possibilita que o processamento seja distribuído entre máquinas de voluntários que desejam colaborar com um determinado projeto. Estes dois fatos tornam possível o desenvolvimento de sistemas de distribuição de processamento viáveis em termos de custo e desempenho. O objetivo da metodologia é facilitar o desenvolvimento de sistemas de distribuição de processamento. A metodologia proposta foi utilizada em um estudo de caso desenvolvido para representação de relevo através de polinômio bidimensional, um problema real que demanda grande poder de processamento originado pelos cálculos necessários para estimar os coeficientes do polinômio. Neste estudo de caso, a metodologia se mostrou eficiente ao descrever todas as funcionalidades necessárias para implementar o sistema e robusta no momento da execução. A metodologia a ser proposta aborda uma classe de problema chamada *Bag-of-Tasks*. Problemas do tipo *Bag-of-Tasks* podem ser subdivididos em subproblemas e estes podem ser resolvidos de forma distribuída e independente. A comunicação entre os componentes do sistema é feita através da API do sistema operacional chamada *socket*.

ABSTRACT

ABRANTES, Rossini Pena, M.Sc., Universidade Federal de Viçosa, August of 2008. **Distributed processing methodology with polynomial surface representation application.** Adviser: Leacir Nogueira Bastos. Co-Advisers: Brauliro Gonçalves Leal and Carlos de Castro Goulart.

Researches in several areas are using more and more processing information which demand more computational power to process such information. For this reason Distributed Processing is a fundamental area to make possible and supply this great demand of processing. Based on this fact, the present work proposes a methodology for processing distribution, through the Internet, using volunteer computing. The processing distribution through the Internet guarantees that processing is distributed for a great number of machines. The volunteer computing guarantees that the processing is distributed among volunteers' machines that want collaborate with a certain project. These two facts turn possible and viable system development of processing distribution in terms of cost and efficiency. The objective of the methodology is to facilitate the system development of processing distribution. The proposed methodology was used in a case study developed in this work. The objective of the case study is to implement the methodology to solve a real problem of processing demand. This selected problem is related with the relief representation through a two-dimensional polynomial. The processing demand is originated by the necessary calculations to esteem the coefficients of such a polynomial. In this case study, the methodology was shown efficient when describing all of the necessary functionalities to implement the system and robust in the moment of its execution. The proposed methodology is based in a problem class called Bag-of-Tasks. Problems of Bag-of-Tasks type can be subdivided in smaller problems and these can be solved in a distributed and independent way. The communication among the components of the system is through API of the operating system called *socket*.

1. Introdução

Alguns fatos recentes têm proporcionado o desenvolvimento de aplicações que compartilhem recursos computacionais através da Internet de forma eficiente. Nos primórdios da Internet, ela era utilizada quase que exclusivamente por universidades e órgãos governamentais e possuía largura de banda e velocidade limitantes, além do número de computadores conectados a ela ser uma pequena fração do número atual. Nesta realidade era difícil desenvolver sistemas para compartilhamento de recursos que fossem eficientes, já que a taxa de transmissão era limitada e haviam poucos computadores conectados. Porém a Internet tem apresentado significativas evoluções, o que torna o compartilhamento de recursos uma solução viável e muitas vezes eficiente para determinados tipos de problemas. Com a Internet se tornando cada vez mais rápida [Fowler, 2005], o tempo gasto na transmissão de informações entre aplicações passa a não ser um gargalho para alguns tipos de sistemas, como sistemas de distribuição de processamento. Outro fato que contribui para tornar a Internet um meio de comunicações viável para compartilhamento de recursos é o número de computadores conectados atualmente, que tem apresentado um crescimento exponencial [Zakon, 2006][Argaez, 2007].

A evolução da tecnologia de comunicação é um dos fatores que contribuiu para um outro fenômeno chamado globalização que ampliou o espaço de trabalho institucional e, como novo paradigma deste milênio, desponta nas corporações a necessidade da sinergia em um ambiente multicultural e multiagentes. Neste cenário têm surgido projetos que envolvem mais de uma área do conhecimento, como projetos nas áreas da geoinformática, bioinformática, biofísica, bioquímica, matemática aplicada e em outras áreas. Estes projetos possuem grandes quantidades de informações a serem processadas e possuem informações de diferentes domínios que precisam ser correlacionadas [Braman, 2006]. Neste cenário, o elemento integrador é a tecnologia da informação, que torna possível a operacionalização destes processos interativos trabalhnado grandes quantidades de informações [Mattmann et al., 2006]. Sistemas Distribuídos é a área da Ciência da Computação que estuda e propõe modelos e técnicas para a interação entre aplicações. Sistemas distribuídos em redes locais já são utilizados há vários anos, porém com base em trabalhos publicados e projetos desenvolvidos recentemente como Folding@Home e Genome@Home [Larson et al., 2003], SETI@Home [Anderson et al, 2002] e Predictor@Home [Taufer et al., 2006], é

possível notar que há um crescimento no número de sistemas que utilizam a Internet como meio para interligar as aplicações.

O uso de sistemas distribuídos possui uma série de vantagens em relação aos sistemas centralizados [Coulouris et al., 2005], dentre as quais se destacam: 1) aproveitamento de máquinas potencialmente ociosas 2) é mais barato interconectar vários processadores que adquirir um supercomputador; 3) algumas aplicações são distribuídas por natureza e, portanto mais facilmente implementadas nesse tipo de ambiente; 4) em caso de falha de uma máquina, o sistema como um todo pode sobreviver, apresentando apenas uma degradação de desempenho; 5) é possível ter um crescimento incremental, pois o poder computacional pode ser ampliado através da inclusão de novos equipamentos; 6) sistemas distribuídos são mais flexíveis que máquinas isoladas, por isso muitas vezes são utilizados até mesmo quando não se está buscando desempenho. Porém, também existem algumas desvantagens: 1) pouco software de alto nível disponível para sistemas distribuídos; 2) dificuldades para evitar acesso indevido (segurança); 3) a rede de interconexão pode causar problemas ou não dar vazão à demanda; 4) falta de uma metodologia bem definida para desenvolvimento de aplicações distribuídas.

Uma outra característica dos projetos que distribuem processamento através da Internet é que, além de utilizarem técnicas de sistemas distribuídos para realizar as comunicações ou compartilhamento de recursos entre aplicações, eles pertencem a uma nova área da Computação chamada *Volunteer Computing*, também conhecida por outros termos como *Desktop Grid* [Choi et al., 2007]. Projetos nesta área geralmente possuem grandes quantidades de informação a ser processadas e necessitam de grande poder de processamento. A metodologia utilizada por eles para suprir a demanda de processamento é contar com a computação voluntária, que é o poder de processamento oferecido por voluntários conectados à Internet. Esta metodologia é viável com base nos fatos citados anteriormente: o número crescente de computadores conectados à Internet; a velocidade das conexões com a Internet e o fato de que os computadores *desktop* estão se tornando mais potentes [Fowler, 2005]. Estas características tornam viável a utilização de computação voluntária para resolver problemas que requerem grande poder computacional. As duas principais vantagens desta área é a possibilidade de obter poder de processamento inalcançável por um supercomputador [Geek, 2007] e o custo praticamente zero, já que não há investimentos em computadores potentes e caros. Porém a área ainda apresenta alguns desafios a serem superados pelos responsáveis por

projeto de sistemas *Volunteer Computing*. Um destes desafios é a volatilidade com que os voluntários entram e saem da sessão de processamento adicionando complexidade aos sistemas. Adicionalmente a falta de confiança nos resultado de processamentos enviados pelos voluntários deve ser levada em conta, já que *Volunteer Computing* geralmente opera com voluntários anônimos.

1.1. O problema e sua Importância

O principal problema abordado neste projeto é o crescente poder computacional envolvido nas pesquisas atualmente. Braman em seu trabalho publicado em agosto de 2006 identificou as características das pesquisas na atualidade [Braman, 2006]. Algumas destas características têm influência direta sobre o poder computacional necessário e a quantidade de informações a serem analisadas e processadas. Pesquisas em todas as disciplinas (áreas do conhecimento) estão se tornando computacionalmente intensivas. Pode-se citar as pesquisas na área da biologia, química e principalmente física envolvendo computação intensiva, além de outras áreas que estão se beneficiando com as possibilidades que a computação oferece como mecatrônica e música. Outra influência, que está relacionada à primeira, é o fato das pesquisas estarem se tornando multidisciplinares. Estão se tornando mais comuns projetos na área da bioquímica, bioinformática, geoinformática, biofísca, dentre outras. Por envolverem mais de uma disciplina, estes projetos necessitam de uma quantidade maior de informações a serem processadas e correlacionadas. Outro fator de influência é o trabalho de alguns projetos sobre dados originados de múltiplos estudos, de múltiplos locais, através de múltiplos períodos. Este fator, além de resultar na necessidade de maior poder de processamento, aumenta a necessidade de maior comunicação (compartilhamento de informações) entre grupos de pesquisas e instituições. Por fim, o crescimento exponencial da quantidade de informações produzidas, é uma influência que talvez esteja subentendida nas quatro anteriores. Lyman e Varian desenvolveram um trabalho que identificou a quantidade de informações geradas de 1999 a 2003 [Lyman e Varian, 2003]. Apesar da sua não continuidade, este trabalho é importante pois dá a idéia sobre a quantidade de informações que é gerada e trafegada na Internet, indicando que provavelmente a quantidade de informações envolvidas nas pesquisas tem crescido numa taxa semelhante.

Diversos grupos de pesquisa e projetos superaram a necessidade de poder computacional com investimento em supercomputadores com grande capacidade de

processamento e armazenamento. Ou seja, é um problema que possui solução possível e esta solução vem sendo utilizada em universidade e instituições governamentais. Porém grupos de pesquisas e projetos que não possuem suporte financeiro para adquirir supercomputadores deixam de realizarem pesquisas importantes por falta de poder de processamento ou espaço de armazenamento suficiente para conduzirem suas pesquisas. Pesquisas como a desenvolvida no estudo de caso, relacionada à representação de relevo através de polinômios, necessitariam de um supercomputador ou grande poder computacional originado de diversas máquinas *desktop*. Além disto, segundo Braman (2006), a computação de alto desempenho ou o grande poder computacional abre novos mundos para as pesquisas.

1.2. Objetivo

O objetivo principal do projeto é propor um modelo de distribuição de processamento que atenda aos principais requisitos da área de *Volunteer Computing* e que possibilite o desenvolvimento de sistemas de alto desempenho para possibilitar processamento voluntário. De acordo com uma análise empírica sobre os trabalhos disponíveis na literatura é possível notar que existem dois extremos com relação ao material disponível sobre distribuição de processamento através da Internet utilizando computação voluntária. Em um extremo (baixo nível) estão as metodologias e técnicas para desenvolvimento de sistemas distribuídos como modelos de comunicação, formas de sincronização de processos, técnicas para implementar exclusão mútua, técnicas de paralelismo, etc. No outro extremo (alto nível) estão apresentações de grandes projetos da área da computação voluntária como SETI@Home, FightAids@Home, etc. Nos trabalhos que apresentam estes grandes projetos não há detalhamento dos modelos de distribuição de processamento que deram origem a tais aplicações. Estes modelos, não descritos, representam a ligação entre o extremo de baixo nível e o de alto nível. Por esta razão, o objetivo desde trabalho é propor um modelo que faça a ligação entre as técnicas e metodologias disponíveis na literatura, para desenvolvimento de Sistemas Distribuídos e Distribuição de Processamento, e as aplicações de distribuição de processamento através da Internet utilizando computação voluntária. A metodologia a ser proposta aborda uma classe de problema chamada *Bag-of-Tasks* [Andrews, 1991]. Problemas do tipo *Bag-of-Tasks* podem ser subdivididos em subproblemas e estes podem ser resolvidos de forma distribuída e independente. Este tipo de problema está

presente em várias áreas como *Data Mining*, consultas massivas (consulta por chaves criptografadas), manipulação de imagens, etc. Como objetivos específicos temos:

- identificar quais são os principais requisitos envolvidos na distribuição de processamento do tipo *Bag-of-Tasks*;
- a partir dos requisitos levantados, propor uma metodologia de distribuição de processamento que seja facilmente adaptada a problemas do tipo *Bag-of-Tasks* que possibilite o desenvolvimento de sistemas de alto desempenho utilizando computação voluntária. O modelo deverá possuir três classes de componentes: os componentes que não mudam de um problema para outro (núcleo do sistema); os componentes adaptáveis para resolver problemas específicos e os componentes opcionais, que poderão ou não ser utilizados na solução de um determinado problema;
- o terceiro objetivo é desenvolver um sistema de distribuição de processamento baseado na metodologia proposta que resolva um problema real. Deste modo será possível avaliar todos os aspectos da metodologia e analisar se ela possui as funcionalidades necessárias para atender demandas do mundo real;
- como último objeto específico, se pretende executar o sistema desenvolvido em uma rede de computadores e avaliar o desempenho do sistema, pois o desenvolvimento de sistemas de alto desempenho faz parte do objetivo principal do projeto.

1.3. Organização do Texto

Este trabalho está organizado da seguinte forma: O Capítulo 1 apresenta uma introdução sobre a evolução dos sistemas computacionais e tecnologias de comunicação juntamente com a necessidade de poder de processamento das pesquisas atualmente. Mostra como a área de Sistemas Distribuídos e *Volunteer Computing*, que utilizam os sistemas computacionais e tecnologias de comunicação para resolver problemas, podem oferecer uma solução à demanda por poder de processamento das pesquisas e projetos. Adicionalmente, neste capítulo é apresentado a importância que representa o poder de processamento para alguns projetos e pesquisas e, por fim, são apresentados os objetivos deste trabalho, que é propor uma metodologia para distribuição de processamento através da Internet baseada em computação voluntária.

O Capítulo 2 descreve as principais características da área de Sistemas Distribuídos, Distribuição de Processamento, Comunicação Inter-Processos, *Volunteer Compunting* e dois tópicos relacionados ao Estudo de Caso desenvolvido: Modelagem Numérica de Terreno e Regressão não-Linear.

No Capítulo 3 são apresentados os métodos utilizados neste projeto para se alcançar os objetivos apresentados no Capítulo 1. É também descrita, neste capítulo, a metodologia proposta.

O Capítulo 4 mostra o estudo de caso desenvolvido em que é implementado um sistema que resolva um problema do mundo real. Este problema está relacionado com a representação de relevo através de um polinômio bidimensional de grau elevado.

O Capítulo 5 mostra os resultados obtidos com o sistema desenvolvido para validar a metodologia.

O Capítulo 6 apresenta as conclusões e sugestões para trabalhos futuros que poderão ser desenvolvidos utilizando este projeto como base.

2. Referencial Teórico

2.1. Sistemas Distribuídos

2.1.1. Características

Existem duas formas de caracterizar Sistemas Distribuídos. A primeira caracterização é mais genérica e leva em consideração todas as áreas da Computação que possuem algum sistema que não seja centralizado, cujos componentes são organizados para atingir um objetivo em comum. Existe um sistema de classificação criado pela ACM (Association for Computing Machinery) que definiu uma estrutura em árvore onde são listadas e classificadas todas as áreas da Computação [ACM, 1998]. De acordo com esta classificação, Sistemas Distribuídos está presente em áreas como Arquitetura de Processadores, Redes de Computadores, Sistemas Operacionais, etc. Na Figura 2.1 está ilustrada quais as áreas da Computação que possui algum tipo de Sistemas Distribuídos segundo a ACM.

A segunda forma de caracterização de Sistemas Distribuídos leva em consideração os sistemas chamados fracamente acoplados ou geograficamente distribuídos. Normalmente são sistemas que são executados em computadores e dispositivos móveis que trocam informações através da troca de mensagens. Pode ser entendido como uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente [Tanenbaum e Steen, 2002]. Coulouris et al. (2005) definem a área como um sistema no qual componentes localizados em computadores ligados por uma rede, se comunicam e coordenam suas ações somente através da troca de mensagens.

A principal motivação para se desenvolver e utilizar sistemas distribuídos é o compartilhamento de recursos. Os recursos podem ser tanto de hardware (por exempo, disco rígido e processadores) como de software (dados, arquivos e objetos). Cada sistema de compartilhamento tem suas características próprias, porém estes sistemas também possuem características em comum que devem ser levadas em consideração na modelagem de qualquer sistema distribuído. Estas características são descritas nos modelos de arquitetura que incluem as camadas de software e os modelos de comunicação.

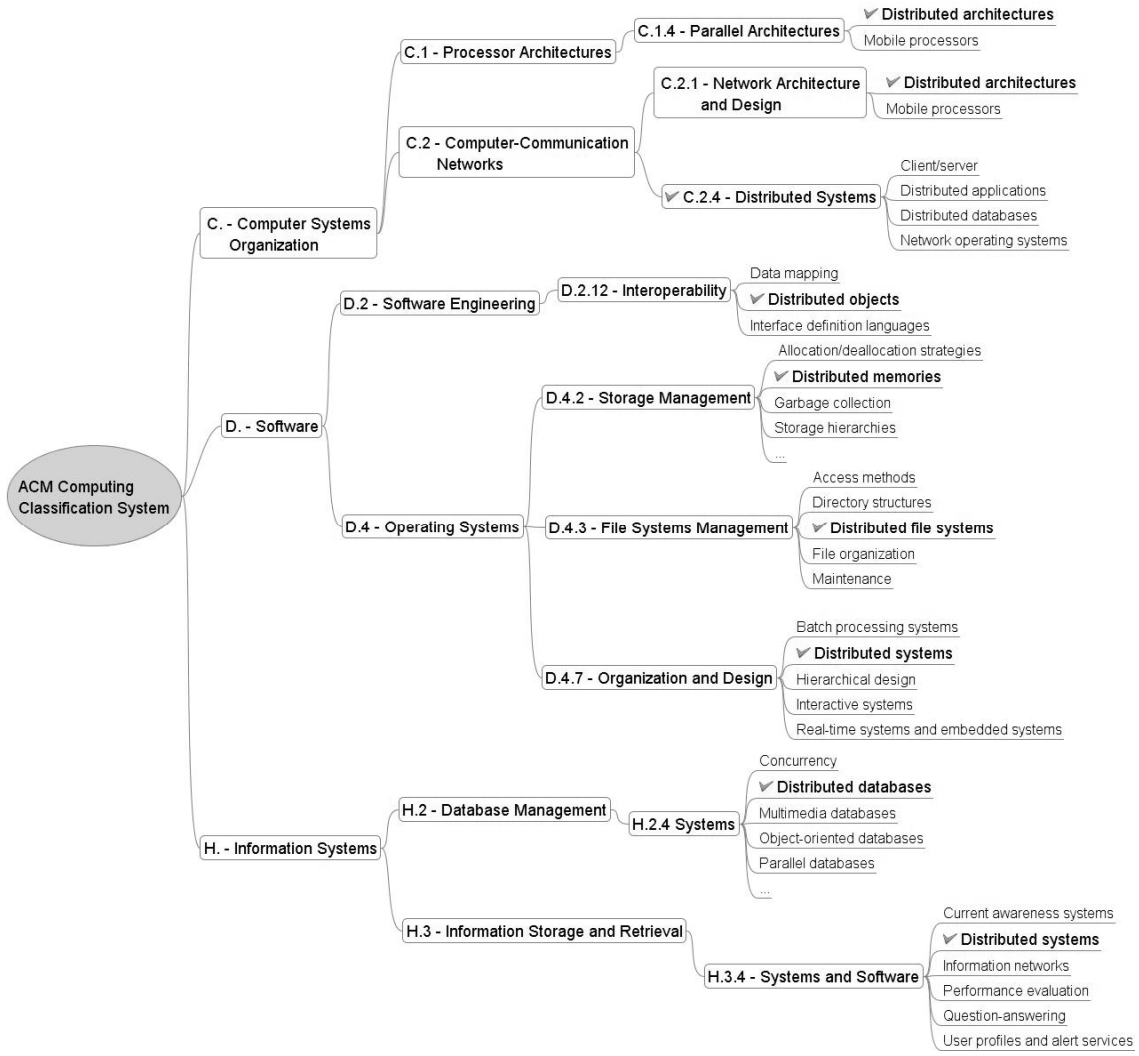


Figura 2.1. Áreas da Computação que possuem Sistemas Distribuídos.

2.1.2. Modelos de Arquitetura

Os modelos de arquitetura propostos por Coulouris et al. (2005) descrevem os componentes envolvidos nos sistemas distribuídos e como eles interagem, sendo (a) a localização do componente através de uma rede de computadores, que busca definir padrões para distribuição de dados e processamento e (b) o inter-relacionamento entre componentes, que abrange suas funcionalidades e padrões de comunicação entre eles. O principal objetivo dos Modelos de Arquitetura é atender a requisitos inerentes dos sistemas distribuídos para torná-los estáveis, confiáveis e gerenciáveis, independente do recurso compartilhado e da disposição ou das funcionalidades dos seus componentes.

Arquitetura de Software

A arquitetura de software define as camadas de software que constituem um sistema distribuído. Estas camadas podem ser compreendidas como prestação de serviço assim

como as camadas de um protocolo de rede [Tanenbaum, 2003a], onde a camada inferior presta algum tipo de serviço para a camada superior. As camadas da arquitetura de software estão ilustradas na Figura 2.2.

Figura 2.2. Arquitetura de Software de Sistemas Distribuídos

As duas camadas inferiores da arquitetura, constituídas pelo hardware (computadores) e software (sistema operacional) são denominadas plataforma por oferecer suporte ao desenvolvimento de sistemas distribuídos. As camadas que compõem a plataforma disponibilizam interfaces de programação de aplicações conhecidas como API (Application Program Interface) [Wikipedia, 2008a]. Estas API's possibilitam o desenvolvimento de sistemas distribuídos em mais alto nível, já que os responsáveis pelo desenvolvimento não precisam se preocupar com aspectos de baixo nível tais como escalonamento de processos, gerencia de memórias, montagem de pacotes para transmissão de dados, etc.

A camada *middleware* possibilita o desenvolvimento de sistemas distribuídos em um nível ainda mais alto. O desenvolvimento de aplicações passa a ser realizado em um nível de abstração mais alto já que o propósito dos *middlewares* é oferecer um modelo de desenvolvimento conveniente para os programadores. O principal objetivo desta camada é atender a requisitos de comunicação de sistemas e compartilhamento de recursos presentes em diversos tipos de sistemas distribuídos como: (a) comunicação entre grupos de processos; (b) localização e recuperação de objetos compartilhados entre computadores; (c) replicação de informações e (d) transmissão de informações multimídia em tempo real. Alguns *middlewares* orientados a objeto mais conhecidos e utilizados são: CORBA [Bolton, 2001], Java RMI [Downing, 1998] e DCOM [Redmond, 1997]. Porém a camada *middleware* é considerada opcional já que algumas aplicações possuem requisitos de desempenho e modelagem que não são atendidos

pelos *middlewares* existentes. Sua utilização freqüentemente significa perda de eficiência já que são desenvolvidos de forma a atender uma gama de problemas genéricos ao invés de problemas específicos de cada aplicação [Saltzer et al., 1984].

Arquitetura de Sistema

A Arquitetura de Software apresentada anteriormente, especifica a organização das camadas de softwares que estão presentes em cada componente do sistema distribuído. A Arquitetura de Sistema especifica o modelo de interação e comunicação entre os componentes que constituem um sistema distribuído. Um dos modelos de comunicação é o *Peer-to-Peer* [Steinmetz e Wehrle, 2005], [Schollmeier, 2002] onde não existe diferenciação entre os componentes do sistema. Todos os componentes possuem os mesmo papéis, não há hierarquia e cada componente pode se comunicar com todos os demais componentes do sistema. Atualmente, o modelo *Peer-to-Peer* vem sendo largamente estudado e diversos sistemas que utilizam este modelo de comunicação e *middlewares* estão sendo desenvolvidos. Porém o modelo mais citado na literatura e mais utilizado em sistemas distribuídos ainda é o modelo Cliente/Servidor, descrito em [Sinha, 1992]. Neste modelo de comunicação existem basicamente dois tipos de componentes, um cliente que requisita ou utiliza algum serviço e o servidor que presta serviço ao cliente. Neste modelo a comunicação se dá apenas entre cliente e servidor, clientes não têm conhecimento sobre a existência de outros clientes, apenas o servidor se comunica com todos os clientes. Diferente do modelo *Peer-to-Peer* que não possui tipos de mensagens pré-definidas, o modelo Cliente/Servidor possui dois tipos básicos de mensagens conhecidas como *Request* e *Reply*, que representam o pedido de um cliente e sua respectiva resposta como ilustrado na Figura 2.3.

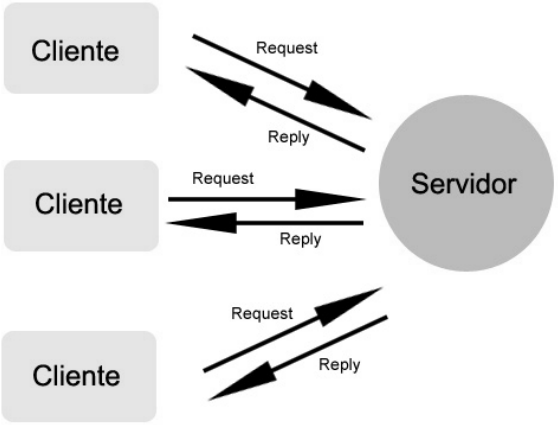


Figura 2.3. Troca de mensagens no modelo Cliente/Servidor

2.1.3. Desafios

Apesar de parecer simples, os modelos de arquiteturas e os sistemas distribuídos podem apresentar desafios complexos que devem ser previstos por projetistas. Seja um sistema de banco de dados distribuído, sistema de replicação de dados, de distribuição de processamento, de compartilhamento de dados, todos eles irão apresentar os mesmo desafios inerentes à área de Sistemas Distribuídos. Alguns desafios já foram superados pelas camadas inferiores da Arquitetura de Software, mas é importante que estes projetistas estejam cientes das suas características e a importância da sua superação.

Heterogeneidade

A Internet possibilita que computadores em redes heterogeneas se comuniquem. Isto é possível porque a Internet utiliza regras de comunicação em que todos os computadores devem conhecer e utilizar os mesmos protocolos para que possam se comunicar. Portanto, assim como a Internet, que é um exemplo de sistema distribuído [Coulouris et al., 2005], os projetistas devem levar em consideração a heterogeneidades relacionadas a: (a) redes onde computadores estarão localizados; (b) configurações dos hardwares; (c) sistemas operacionais; (d) a linguagem de programação utilizada para desenvolver os componentes e (e) os componentes que podem ser desenvolvidos por diferentes programadores com diferentes estilos de programação.

Extensibilidade

Este desafio está relacionado basicamente à capacidade de um sistema ser estendido, reutilizado ou adaptado, o que requer padronização de seus componentes e da interação entre eles. A melhor forma de superar este desafio é tornar público os padrões de comunicação definidos, as interfaces dos componentes e os serviços prestados por cada componente, para que eles possam ser estendidos ou adaptados por outras pessoas ou empresas para atender a diferentes requisitos. Um exemplo de uma solução para este desafio são os documentos criados pelos projetistas de protocolos da Internet chamado "*Request For Comments*" ou RFC's (http://www.ietf.org/) onde as especificações dos protocolos são disponibilizadas publicamente.

Segurança

Segurança é um dos principais desafios da área de Sistemas Distribuídos, pois estes sistemas trocam informações importantes utilizando a Internet que não oferece segurança da transmissão de dados, como meio de comunicação. As ameaças são várias: (a) confiabilidade: a revelação do conteúdo de uma mensagem por pessoas não autorizadas; (b) integridade: violação ou alteração do conteúdo das mensagens; (c) disponibilidade: interferência com intuito de acessar algum recurso do sistema; e outros dois problemas ainda sem soluções definitivas (d) *Denial of Service*: ataque com intenção tornar indisponível um serviço prestado através de um número elevado de requisição feitas ao servidor ao mesmo tempo e (e) código móvel: alteração de um trecho de código que foi enviado a um componente do sistema para ser executado.

Escalabilidade

Está relacionado com a capacidade de um servidor de atender a um crescente número de clientes, ou no caso de um sistema que utiliza o modelo de comunicação *Peer-to-Peer*, de suportar a comunicação entre um número crescente de componentes ao mesmo tempo. Sistemas distribuídos freqüentemente envolvem um número elevado de componentes e é tarefa dos responsáveis pelo projeto de sistemas evitar que clientes deixem de ser atendidos por falta de capacidade do servidor ou que sejam atendidos com atraso.

Tolerância a Falhas

Em sistemas distribuídos, cada componente funciona de forma independente dos demais componentes do sistema. Desta forma, um componente pode apresentar uma falha enquanto os demais continuam funcionando. A falha de um componente pode gerar resultados incorretos ou influenciar o correto funcionamento de outros componentes. Por esta razão a resistência à falhas em sistemas distribuídos é um fator crítico. Algumas técnicas podem ser adotadas para se criar resistência à falhas: (a) detecção de falhas; (b) mascaramento de falhas; (c) tolerância a falhas; (d) recuperação de falhas e (e) redundância.

Concorrência

O controle de concorrência é necessário quando há mais de um componente do sistema tentando acessar uma mesma informação ao mesmo tempo. Este acesso concorrente, se

não for controlado, poderá gerar resultados inconsistentes. Esta concorrência é mais evidente em servidores *multi-thread*, onde várias *threads* acessam à mesma informação ao mesmo tempo. Nesta situação, o servidor deve possuir algum controle de concorrência para evitar inconsistência dos dados compartilhados. Um controle muito conhecido e utilizado em sistemas operacionais é o semáforo (implementa exclusão mútua), que permite que apenas um componente ou objeto acesse o dado compartilhado por vez.

Transparência

É o desafio de fazer com que o sistema distribuído pareça, aos olhos do usuário, como um sistema único e consistente, ao invés de um conjunto de componentes que estão interagindo. Existem diversos tipos de transparência: (a) transparência de acesso: permite o acesso a recursos locais e remotos utilizando as mesmas operações; (b) transparência de local: o usuário não deve ter conhecimento da localização geográfica ou localização na rede de um recurso que ele está acessando; (c) transparência de concorrência: os usuários não devem tomar conhecimento que estão utilizando um recurso concorrentemente; (d) transparência de replicação: uso de uma replica da informação original pelo usuário sem o conhecimento do mesmo; (e) transparência a falhas: o sistema deve identificar e tratar as falhas sem que o usuário perceba a sua ocorrência; (f) transparência de mobilidade: em que o usuário não deve perceber que um recurso utilizado por ele mudou de lugar (lugar geográfico ou na rede); (g) transparência de desempenho: o usuário não notar que o sistema foi reconfigurado para obter melhor eficiência e (i) transparência da escala: o sistema deverá ser escalável sem modificar sua estrutura básica ou seu funcionamento.

2.2. Distribuição de Processamento

Conforme Hariri e Parashar (2004), três fatores contribuíram e continuam contribuindo para a evolução dos sistemas de distribuição de processamento. A primeira é relacionada à revolução que ocorreu nas duas últimas décadas na Computação. É a mudança do processamento centralizado dos *Mainframes* para o processamento em computadores pessoais de pequeno porte que são significativamente mais baratos do que foram os *Mainframes*. Com esta revolução o número de computadores cresceu exponencialmente, assim como seu poder de processamento, capacidade de armazenamento, e outras características. Este crescimento da capacidade dos

computadores é o segundo fator que contribuiu para a evolução da distribuição de processamento. Segundo a Lei de Moore [Stokes, 2003], o número de transistores de um circuito integrado dobra a cada dois anos. Esta lei também se aplica as outras configurações dos computadores como velocidade de processamento, capacidade de memória, etc. Um número cada vez maior de computadores cada vez mais potentes é o ambiente adequado para a distribuição de processamento. Para que este ambiente se torne ainda melhor é necessário tecnologias de comunicação entre computadores. Este é o terceiro fator que contribuiu para a área de distribuição de processamento. A tecnologia de rede de computadores tem evoluído significativamente nos últimos anos. Desafios do passado, como taxa de erro das redes, latência, largura de banda, etc., estão sendo superados pelas novas tecnologias [Tanenbaum, 2003a]. Uma tecnologia que tem contribuído para esta evolução é a Internet, que possibilitou que um número cada vez maior de computadores possam estar conectados [ISC, 2008]. Com computadores cada vez mais eficientes e se comunicando também de forma mais eficiente, a área de distribuição de processamento passa a apresentar vantagens sobre o processamento centralizado.

2.2.1. Arquitetura de Software

O projeto de um sistema de distribuição de processamento é constituído basicamente de três partes: (a) o modelo da arquitetura dos componentes que farão parte do sistema; (b) o métodos e a tecnologia de comunicação que irão possibilitar a comunicação entres os componentes do sistema, e (c) os serviços e as funcionalidades que cada componente irá possuir, sendo que nesta fase é definida o papel de cada componente. A arquitetura de software trata da primeira parte do projeto, que é definir e descrever a arquitetura interna de cada componente do sistema distribuído. A arquitetura de software possui três elementos ou camadas básicas, comunicação, arquitetura de sistema e serviços e paradigma de distribuição de processamento, enumeradas a seguir.

Comunicação

Descreve os principais sistemas de comunicação que serão utilizados para transmitir mensagens de controle e de dados entre os componentes. Esta camada é subdividia em três partes: (a) tipo da rede: define em qual tipo de rede os dados serão trafegados, podendo ser rede LAN, MAN ou WAN, com ou sem fio; (b) quais serão os protocolos de rede utilizados para tornar a comunicação entre componentes possível e (c) qual a

interface de rede será utilizada para realizar a comunicação. Esta é a camada inferior na arquitetura de software e assim como os modelos de camadas de outras arquiteturas e modelos de camadas de protocolos, neste modelo cada camada presta algum tipo de serviço para a camada superior.

Arquitetura de Sistema e Serviços

Esta camada representa a visão que o projetista tem da plataforma onde o sistema distribuído será implementado. Também conhecida como camada de Arquitetura de Sistema e Serviços – SAS em inglês. Os serviços desta camada estão relacionados a alguns serviços que o sistema operacional disponibiliza para o desenvolvimento de aplicações. São estes serviços que irão dar suporte a um sistema distribuído transparente, que aos olhos do usuário pareça um sistema único e consistente. Alguns serviços são: (a) sistema de arquivos distribuídos; (b) controle de concorrência; (c) gerenciamento de redundância; e (d) balanceamento de carga.

Paradigma de Distribuição de Processamento

Esta camada está relacionada com aspectos de mais alto nível como Modelos de Computação e Modelos de Comunicação. A definição desta camada está mais relacionadas à distribuição de processamento especificamente, ao contrário da camada de Arquitetura de Sistema e Serviços que está relacionada a aspectos mais genéricos de sistemas distribuídos. O Modelo de Computação é dividido em duas partes: (a) Computação Paralela: todos os componentes executam os mesmo programas sobre os mesmo dados, porém cada componente executa uma operação sobre uma parte específica do dado; e (b) Computação Distribuída: cada componente do sistema executa uma função diferente dos demais componentes. O Modelo de Comunicação também se divide em duas partes: (a) Transmissão de Mensagens: em que a única forma de compartilhar informações entre os componentes do sistema é através de transmissão de mensagens entre os componentes; e (b) Memória Compartilhada: onde um espaço de memória é compartilhado e todos os componentes do sistema têm acesso a este mesmo espaço. Na Figura 2.4 estão ilustrados os principais elementos da arquitetura de software.

Figura 2.4. Arquitetura de Software de Sistemas de Distribuição de Processamento

2.2.2. Vantagens dos Sistemas de Distribuição de Processamento

Desempenho

Um conjunto de máquinas trabalhando em paralelo para solucionar um problema em comum pode atingir um poder de processamento que seria inalcançável por um supercomputador. Em 2007 uma parceria entre Sony e o projeto Folding@Home [Folding@Home] da Universidade de Stanford alcançou o poder computacional superior a um petaflop (mil teraflops ou mil trilhões de flops) e bateu o recorde de rede de distribuição de processamento mais poderosa do mundo [Geek, 2007]. Atualmente existem diversos dispositivos com poder de processamento que podem ser utilizados para processar informações em um sistema de processamento distribuído como videogames, celulares, dispositivos móveis, etc. Na parceria citada acima, foram utilizados o poder de processamento de 670 mil usuários do console *Play Station* 3 para alcançar o recorde mundial.

Compartilhamento de Recursos

O compartilhamento de recursos em sistemas de distribuição de processamento é a mais notória vantagem, além de ser o principal propósito da tecnologia de distribuição de processamento. Não só este tipo de sistema, mas todos os tipos de sistemas distribuídos possuem a vantagem de compartilhar recursos, muitas vezes caros, como poder de processamento, espaço de armazenamento, etc.

Extensibilidade

É possível modelar sistemas de distribuição de processamento de forma modular e adaptativo, para que este sistema possa se auto configurar para suportar um número crescente de componentes. Desta forma, para se obter maior poder de processamento para um sistema, seria necessário apenas acrescentar máquinas à rede de computadores. Tecnologias de comunicação como *Peer-to-Peer* [Schollmeier, 2002] e comunicação em grupo [Coulouris et al., 2005], tem possibilitado que um número cada vez maior de componentes em um sistema distribuído se comunique com eficiência.

Custo

Existem duas vantagens relacionadas ao custo de sistemas distribuídos sobre sistemas centralizados. A primeira é a possibilidade de comprar um conjunto de computadores cujo poder de processamento seja igual a um supercomputador a um preço inferior ao deste. A segunda vantagem é ainda mais significativa por possibilitar o processamento de informações com gastos com equipamento praticamente zero. É a nova área de pesquisa, cuja classificação e caracterização foram criadas em 2007, chamada *Desktop Grid* [Choi et al., 2007], também citada na literatura como *Public Computing* [Anderson, 2004]. Projetos nesta área utilizam o tempo em que as máquinas de voluntários ficam ociosas para processar informações relacionadas a um determinado projeto. Os computadores dos voluntários se interligam ao servidor do projeto por meio da Internet.

Confiabilidade, Disponibilidade e Tolerância a Falhas

Um sistema de distribuição de processamento poderá implementar técnicas de tolerância a falhas que o tornem mais disponíveis em caso de falha de um dos componentes do sistema. Nesta situação, se um componente apresentar algum tipo de falha sempre haverá um outro componente para assumir sua tarefa, tornando o sistema mais disponível, como acontece com replicação de informações, por exemplo.

2.2.3. Desvantagens dos Sistemas de Distribuição de Processamento

Teoria

Em algumas áreas dos sistemas distribuídos ainda não existe uma teoria madura e consistente que identifique e apresente soluções para todos os seus desafios. Ainda há falta de linguagens específicas, ferramentas para desenvolvimento e testes que consigam torna a modelagem de um sistema distribuído parecido com a modelagem de sistemas centralizados. Alguns tipos de sistemas distribuídos só conseguiram uma definição única e consistente recentemente. Como o caso da área de Computação em Grid que teve uma caracterização única e consistente publicada em 2007, depois das diversas mudanças pelas quais passou a área desde sua criação [Stockinger, 2007].

Comportamento assíncrono e independente

Em um sistema de processamento distribuído cada componente é executado de forma assíncrona e independente, o que dificulta alguns controles sobre estes componentes como: (a) sincronização; (b) detecção de falhas; (c) qualidade de serviço; e (d) controle de segurança. Além do fato de uma falha em um componente poder afetar a execução de outros, assim como uma falha de segurança em um componente também poderá afetar a segurança de todo o sistema.

Rede de Comunicação

Apesar de as tecnologias de comunicações estarem avançadas e consistentes, os projetistas de sistemas distribuídos ainda possuem dificuldades para projetar sistemas seguros, robustos e confiáveis. Fazer com que um sistema funcione sem problemas de eficiência ou robustez com milhares de componentes é uma tarefa árdua para os projetistas, pois ainda é necessário desenvolver técnicas para tornar a comunicação entre componentes mais segura e robusta. A comunicação entre componentes de um sistema distribuído utilizando a Internet como meio de comunicação poderá acrescentar algumas ameaças ao sistema com relação à segurança na comunicação. Sistemas que trocam mensagens através da Internet poderão sofrer diversos ataques como citado da sessão 2.1.3.

2.3. Comunicação Inter-Processos

Comunicação inter-processos são recursos e técnicas utilizadas para que dois ou mais processos possam trocar informações. Os recursos que são utilizados por desenvolvedores são protocolos de rede, API's do sistema operacional, e recursos de linguagens de programação. Além destes recursos são utilizadas técnicas de

comunicação, geralmente específicas de cada solução, para atender aos requisitos de comunicação de uma solução. Algumas técnicas genéricas de comunicação são implementadas por *middlewares* de comunicação como CORBA e Java RMI, outras técnicas mais específicas são implementadas por desenvolvedores utilizando os recursos de mais baixo nível.

A comunicação inter-processos pode ser dividida em camadas, que possuem papeis específicos na comunicação. As camadas são separadas por níveis de abstrações como ilustrado na Figura 2.5.

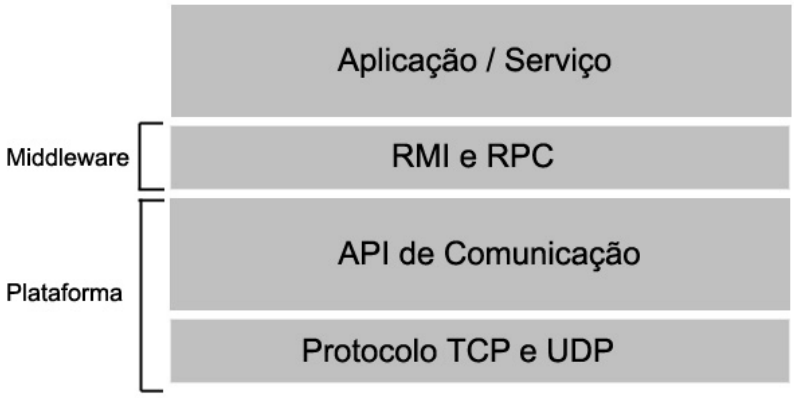


Figura 2.5. Camadas da Comunicação Inter-Processos

2.3.1. Protocolo TCP e UDP

Segundo Tanenbaum (2003),os protocolos TCP e UDP são protocolos de comunicação utilizados para enviar e receber informações de computadores remotos. O protocolo UDP é uma das formas mais simples de se transmitir e receber informações entre computadores. Este protocolo oferece rotinas para enviar e receber informações do modo não orientado a conexão, onde não há estabelecimento de conexão entre os computadores e não há garantias de entrega ou recebimento da informação transmitida. Neste protocolo a informação a ser transmitida é dividida em blocos de informações chamados datagramas, de tamanho variável, mas que possuem um tamanho máximo predefinido. O protocolo TCP é mais confiável, pois opera orientado a conexão, o que significa que uma conexão é estabelecida antes do início da transferência de dados. As trocas de informações orientadas a conexão são mais confiáveis por possuir confirmação de recebimento de mensagens. Com as confirmações, um processo que envia uma mensagem para outro processo sempre saberá se sua mensagem chegou ou não a seu destino. Ambos utilizam o protocolo IP para rotear os pacotes na Internet e

fazer com que eles cheguem na máquina destino e a porta de comunicação. A porta é um número inteiro que serve para identificar qual dos processos, dos vários que poderão estar ativos na máquina, é o processo de origem ou destino de uma mensagem. Desta forma, para estabelecer conexão e enviar mensagens para outros processos, o processo local deve conhecer o endereço IP e porta do processo remoto.

2.3.2. API de Comunicação

Comunicação Síncrona e Assíncrona

A comunicação entre processos se resume a enviar mensagens para um processo remoto através da função *send* para que a mensagem seja acrescentada ao *buffer* do processo remoto até que este processo execute a função *receive* e retirar a mensagem do *buffer*. Na comunicação síncrona, a execução das duas funções deve ser sincronizada. Quando um processo executa a função *send*, este processo irá permanecer bloqueado até que o processo destinatário execute a função *receive*. Assim como a execução da função *receive* bloqueia o processo local até que o processo remoto envie alguma informação executando a função *send.* No modo de comunicação síncrona, as funções para enviar e receber mensagens são funções conhecidas como bloquantes por bloquear o processo até haja sincronia na execução destas funções. Em processos *multi-thread* apenas a *thread* que executou a função de enviar ou receber mensagens é bloqueada, não influenciando na execução das demais *threads*. Na comunicação assíncrona, a execução das funções *send* e *receive* não precisam ser sincronizada. Um processo que executa a função *send* não ficará bloqueado até que o processo remoto execute a função *receive*. A mensagem que é enviada fica armazenada no *buffer* de entrada do processo remoto até que ele execute a função para receber a mensagem. Desta forma o processo que enviou a mensagem fica livre para continuar sua execução. Esta forma de comunicação tem a vantagem de não bloquear os processos envolvidos na comunicação, porém acrescenta complexidade no gerenciamento da execução destes processos, principalmente em processos *multi-thread*.

Juntamente com a comunicação (síncrona ou assíncrona) pode ser utilizada uma representação externa de dados. Uma representação muito utilizada é XML em que as informações a serem transmitidas são formatadas de acordo com um padrão previamente estabelecido para que aplicações possam se comunicar. Outra representação conhecida é a utilizada pela tecnologia CORBA para representar tipos

primitivos de dados e parâmetros passados para métodos. Porém as representações externas geralmente adicionam um *overhead* de processamento e informação às mensagens transmitidas. Por esta razão projetos como BOINC [Anderson, 2004] evitam utilizar recursos de representação externa de mensagem pelo seu alto *overhead* [Anderson et al., 2005].

Socket

É uma abstração que provê pontos finais de comunicação entre dois processos. Um ponto de comunicação pode utilizar o protocolo TCP ou o UDP para trocarem mensagens. Esta abstração *socket* foi criada inicialmente no BSD UNIX (http://www.bsd.org), porém atualmente está disponível para as principais plataformas como Linux, Microsoft Windows e Macintosh OS. O processo de comunicação utilizando *socket* se resume em enviar uma mensagem de um *socket* local para um *socket* remoto. Cada *socket* possui as informações de endereço IP e porta que identificam unicamente cada processo na rede. Mensagens podem ser enviadas e recebidas utilizando um mesmo *socket*. Existem 2^{16} (algumas portas são reservas para processos do sistema operacional e outras aplicações) possíveis portas que um processo pode utilizar para se comunicar com outro processo. Um processo poderá utilizar várias portas para se comunicar com outros processos, porém apenas um processo poderá utilizar uma determinada porta, as portas não podem ser compartilhadas entre processos. Diversas linguagens de programação como Java, C e C++ possuem bibliotecas para serem utilizadas por desenvolvedores para realizar a comunicação entre processos. A troca de mensagens entre um par de *socket's* é ilustrada na Figura 2.6. De acordo com Seixas Filho (2004) "A comunicação através de *sockets* tem grandes vantagens em relação à comunicação utilizando tecnologias baseada em camadas como CORBA e DCOM, mais recentes e de maior apelo comercial: (a) é mais eficiente; (b) permite execução em ambientes heterogêneos envolvendo diferentes plataformas como Windows, UNIX, etc. e (c) desfrutam de grande aceitação no mercado. Grande parte dos protocolos presentes na Internet (http, ftp e smtp) utilizam a biblioteca de sockets."

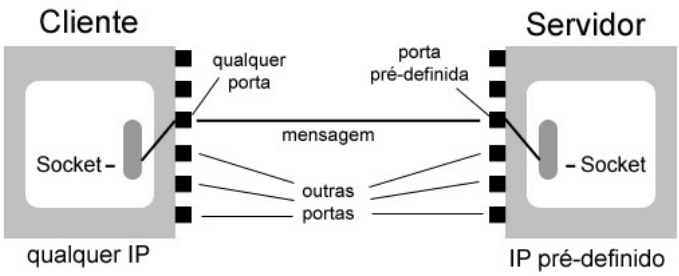


Figura 2.6. Transmissão de mensagens entre *sockets*

2.3.3. RMI e RPC

Tanto a técnica RMI (*Remote Method Invocation)* como RPC (*Remote Procedure Call)* são técnicas utilizadas para desenvolver aplicações distribuídas baseadas no modelo de comunicação cliente/servidor. A utilização destas técnicas torna o desenvolvimento de aplicações em mais alto nível e são opcionais já que os desenvolvedores poderão utilizar as funcionalidades oferecidas pelos *socket's* diretamente. A técnica RPC é similar à execução de uma função. Quando uma RPC é executada, os parâmetros desta função são passados para a função remota, esta função é executada e é retornado o resultado da execução para o processo que invocou o RPC [Nelson, 1981]. O RMI [RMI] é uma tecnologia Java semelhante ao RPC, porém RMI opera orientado a objetos. A invocação de um método em um objeto remoto é semelhante a invocação de um método local. Assim como na chamada de métodos locais, no RMI existe verificação de tipos.

2.4. Volunteer Computing

Volunteer Computing possui algumas semelhanças com a área de Computação em Grade. Em ambas o objetivo principal é alcançar uma computação de alto desempenho através da união de diversos computadores utilizando-se uma rede de computadores. Assim como em Computação em Grade, a área de *Volunteer Computing* objetiva reunir dispositivos heterogêneos em larga escala e de domínio diferentes para prover acesso a recursos computacionais de forma segura, transparente e coordenada [Choi et al., 2007]. A diferença entre estas duas áreas é o fato de *Volunteer Computing* utilizar máquinas de voluntários conectadas à Internet, no período em que estejam ociosas, ao invés de utilizar recursos computacionais, muitas vezes caros como supercomputadores, *cluster's*, instrumentos científicos, dispositivos de armazenamento de dados, etc.

Esta ainda é uma área nova na Computação e por esta razão a sua denominação não está completamente consolidada. A ACM não possui uma classificação formal para

Volunteer Computing ou qualquer outro termo utilizado para descrever a área. Geralmente artigos publicados sobre o tema são classificados como *Distributed Application* (subitem da sessão C.2.4 da Figura 2.1) e *Distributed Systems* (subitem da sessão H.3.4 da Figura 2.1). Diversos trabalhos nesta área dão denominações diferentes para a área. *Volunteer Computing* é uma das denominações mais difundidas e foi escolhida esta denominação por representar com maior fidelidade as características da área (que serão apresentadas posteriormente). Dentre as diversas denominações encontradas na literatura destacam-se:

- **Volunteer Computing** [Anderson et al., 2005][Taufer et al., 2007] e [Christensen et al., 2005];
- **Desktop Grid** [Queiroz et al. 2006][Choi et al., 2007] e [Kondo et al., 2004];
- **Public Computing** [Anderson, 2004a][Maheswaran et al., 2004] e [Kotsovinos, 2005];
- **Public-Resource Computing** [Anderson, 2004b][Anderson et al., 2002] e [Taufer et al., 2005].

Os aspectos que impulsionaram e motivaram a criação e desenvolvimento desta área são: (a) a grande quantidade de processamento requerido em alguns projetos, principalmente da área de Física, Biologia e Matemática, assim como projetos multidisciplinares; (b) evolução das tecnologias de comunicação como a Internet tem apresentado uma largura de banda, taxa de erro cada vez melhores; (c) baixo custo dos computadores *desktops*; e (d) o número crescente de computadores conectados à Internet. Computação voluntária pode prover maior poder computacional que qualquer supercomputador, *cluster* ou *grid* e esta disparidade irá crescer com o tempo [Anderson, 2004a].

2.4.1. Caracterização

Só recentemente, em 2007, foi publicado um trabalho cujo objetivo é definir a caracterização e classificação de *Volunteer Computing* [Choi et al., 2007]. Neste trabalho, os autores se referirem à área como *Desktop Grid*, porém há pouca ou nenhuma diferença entre as caracterizações. Apesar de alguns projetos não seguirem com fidelidade o modelo proposto por Choi et al. (2007), este trabalho será utilizado como base para a descrição das características da área. *Volunteer Computing* pode ser

classificada de acordo com quatro aspectos, organização, plataforma, escala e provedor de serviços, conforme ilustrado na Figura 2.7**.**

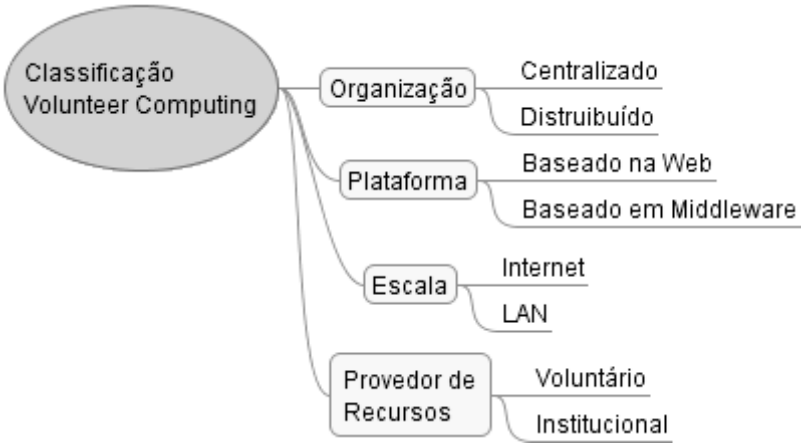


Figura 2.7. Classificação de *Volunteer Computing*

Organização

Sistema de computação voluntária centralizado é constituído por um servidor centralizado, um cliente e vários provedores de serviços. O cliente é responsável por submeter uma determinada tarefa ao servidor, requisitando que estas tarefas sejam processadas. Geralmente estas tarefas podem ser processadas em paralelo e fica a cargo do servidor gerenciar a distribuição. O papel do servidor é receber as tarefas do cliente, subdividir estas tarefas em sub-tarefas, realizar um agrupamento de provedores de recursos (etapa opcional) para que estes provedores sejam agrupados de acordo com alguma norma predefinida (agrupar provedores em classes de acordo com os tipos de processadores, por exemplo). Depois de agrupado, o servidor passa para a fase de alocação de sub-tarefas entre os diversos provedores que estão conectados a ele, para que estes provedores de recursos processem as sub-tarefas. O servidor então recebe todos os resultados parciais dos provedores, os agrupa e retorna o resultado final para o cliente. Os provedores de recursos são voluntários que doam recursos computacionais para um determinado projeto, são os responsáveis pelo processamento de informações, geralmente são anônimos (podem utilizar algum forma de identificação se desejável), estão ligados ao servidor através de uma rede local ou, como é mais comum, através da Internet. A tarefa principal dos provedores de recursos é se conectar ao servidor requisitando tarefas, processar estas tarefas e depois envia os resultados para o servidor e informando estar pronto ou não para receber mais tarefas. Alguns projetos que utilizam um servidor centralizado como SETI@Home e BOINC não utilizam este

modelo proposto por Choi et al. (2007) na integra, com a existência de um cliente que submete tarefas a um servidor. Estes projetos utilizam um modelo em que o servidor conhece previamente quais são as tarefas a serem processadas e se encarrega de distribuí-las. Na Figura 2.8 estão ilustradas as etapas da distribuição de processamento e o papel de cada uma das partes envolvidas.

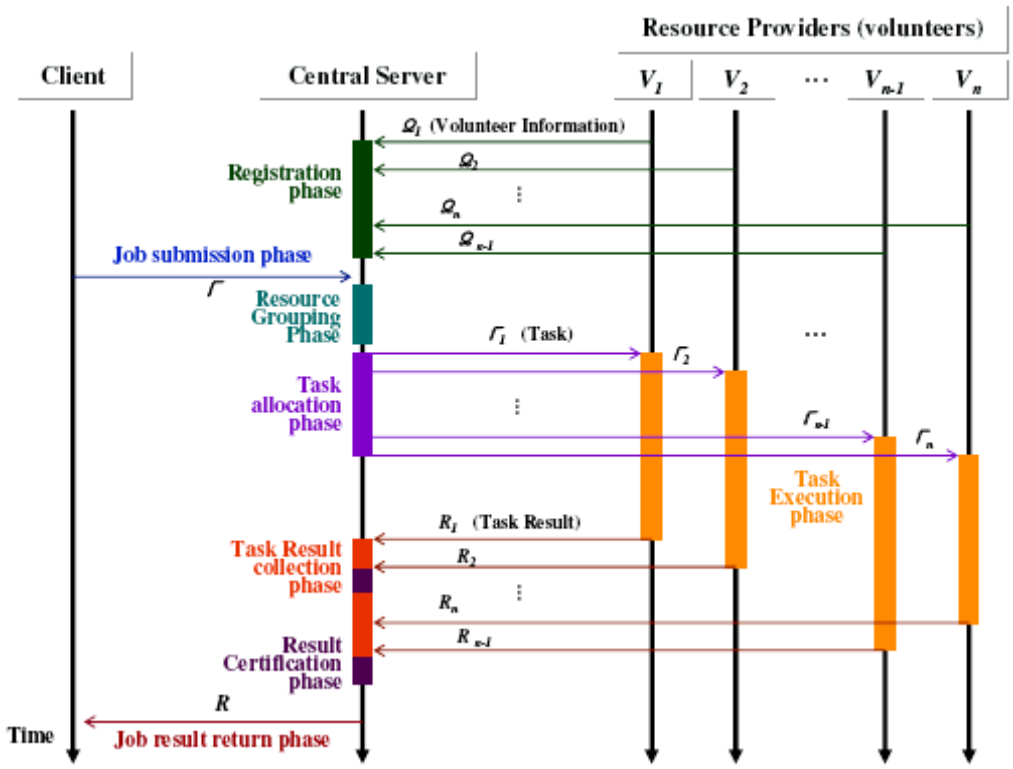


Figura 2.8. Modelo de Execução Centralizado (Fonte: [Choi et al., 2007]).

No sistema de computação voluntária cuja organização é distribuída, não existe um componente com papel de servidor. Na fase de registro (autenticação) cada provedor de recursos troca suas informações com os demais provedores e de forma distribuída criam grupos de provedores. Um cliente envia uma tarefa a ser processada a um ou mais provedores próximos. A partir daí estes provedores passam a operar de forma parecida a um servidor. Eles serão responsáveis por subdividirem a tarefa principal em sub-tarefas e alocar estas sub-tarefas para os demais provedores. Ao final do processamento das sub-tarefas, cada provedor que recebeu uma tarefa do cliente é responsável por reunir os resultados parciais e retornar o resultado geral para o cliente. Na Figura 2.9 estão ilustrados os papéis e as etapas envolvidas na computação voluntária distribuída.

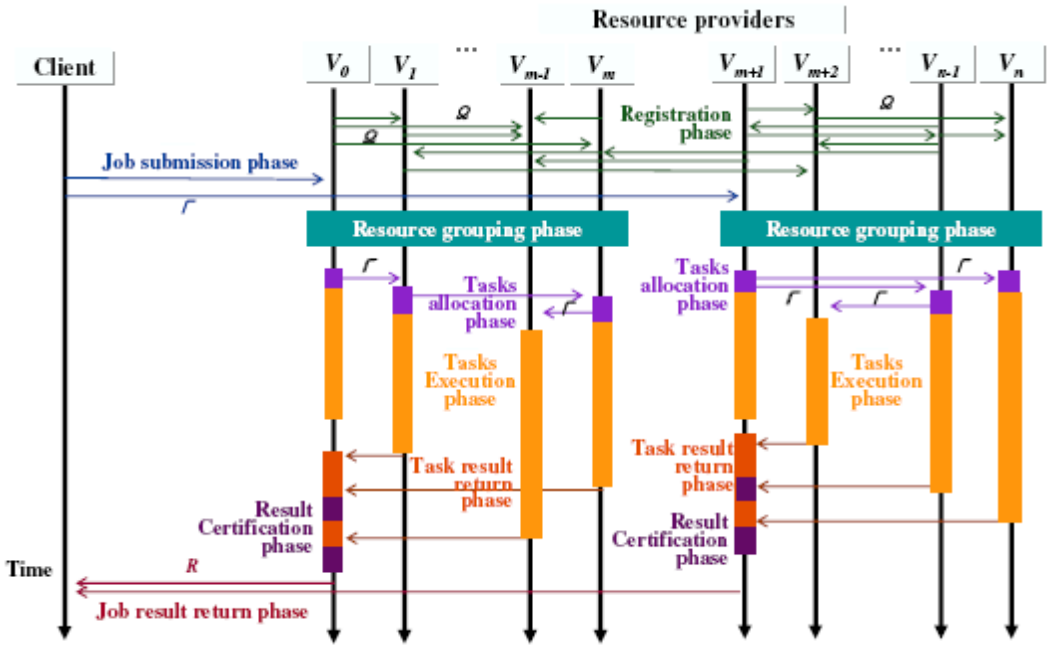


Figura 2.9. Modelo de Execução Distribuído (Fonte: [Choi et al., 2007]).

Plataforma

Volunteer Computing utiliza duas plataformas que os provedores de recursos (voluntários) podem utilizar para colaborarem com um determinado projeto. Em uma plataforma, chamada *Web-based*, um *Java Applet* é executado a partir de uma página. Neste caso o voluntário só irá colaborar enquanto estiver acessando o site do projeto, não será possível colaborar de outra forma. Quando o voluntário acessar a página, será carregado um *applet* responsável pelo processamento de informações recebidas do servidor de páginas web. A *applet* irá processar as informações e retornar o resultado para o servidor. A outra plataforma é conhecida como *Middleware-based* em que uma aplicação é instalada em cada máquina voluntária. Com esta plataforma o voluntário poderá contribuir a qualquer momento sem a necessidade de acessar uma determinada página A aplicação que será executada na máquina voluntária poderá ser programada para executar em tempo parcial para não prejudicar o uso da máquina. Uma política muito utilizada é programar a aplicação para executar apenas quando a máquina do voluntário estiver ociosa.

Escala

Está relacionada ao número de computadores voluntários e com o domínio destas máquinas. A escala poderá ser no nível de Internet, onde os voluntários podem ser anônimos e cada máquina de cada voluntário provavelmente possuirá configuração de

hardware diferente e diferente latência de rede. Outra escala é no nível de rede local, onde o número de computadores é menor, cujas máquinas são de mesmo domínio (instituição, universidade, empresa, etc.).

Provedores de Recursos

Uma diferença que existe entre a área de *Volunteer Computing* e *Desktop Grid*, na área foco do trabalho de Choi et al. (2007) é a possibilidade de se ter provedores de recursos que não possuem o papel de voluntários. Estes provedores são involuntários e geralmente são máquinas que pertencem a domínios fechados como instituições, empresas e universidade. Desta forma, sistemas que possuem provedores de recursos não voluntários passam a se comportar de forma semelhante a um *Computação em Grade*. Nesta categoria as máquinas são mais facilmente gerenciadas, pois geralmente possuem um administrador responsável por conjuntos de máquinas. Já em sistemas onde os provedores são voluntários, onde normalmente estes provedores se conectam ao servidor através da Internet, possuem um gerenciamento mais complicado por trabalhar com máquinas de diferentes configurações, com diferentes tempos de resposta e voluntários, muitas vezes, anônimos.

2.4.2. Desafios

Apesar de alguns desafios de *Volunteer Computing* serem semelhantes aos desafios de Sistemas Distribuídos, já que *Volunteer Computing* também é um tipo de sistema distribuído, os desafios mais específicos são volatilidade, ambiente dinâmico, falta de confiança, falhas, escalabilidade e participação voluntária, discutidas a seguir.

Volatilidade

Voluntários poderão, a qualquer momento, interromper a execução de uma tarefa que recebeu do servidor. Geralmente aplicações voluntárias não são executadas durante longos períodos ininterruptamente. A volatilidade está relacionada não só com a interrupção por completo da execução de uma tarefa, mas também com a interrupção parcial, no período em que a máquina do voluntário estiver ocupada executando outros aplicativos que também requerem significativo poder computacional. Estes aspectos devem ser considerados para prover bom desempenho e robustez aos sistemas de *Volunteer Computing.*

Ambiente Dinâmico

Cada voluntário pode entrar na sessão de processamento ou abandoná-la no momento que achar conveniente. Uma sessão de processamento poderá mudar (com relação ao número de voluntários) constantemente. O sistema deverá suportar este ambiente dinâmico, assim como oferecer meios para que voluntários possam entrar e sair de sessões de processamento de forma eficiente.

Falta de Confiança

Como os voluntários não precisam fornecer informações pessoais, não há um alto grau de confiabilidade entre o servidor e voluntários. Alguns poderão enviar resultados incorretos ou corrompidos voluntariamente ou não. É tarefa do sistema diferenciar resultados válidos dos inválidos.

Falhas

Este desafio está presente em todos os tipos de sistemas distribuídos, principalmente os sistemas que utilizam a Internet. Os diversos voluntários podem possuir tempos de resposta e latências de rede diferentes e um tempo de resposta longo poderia ser considerada erroneamente uma falha. Cabe ao sistema a diferenciação entre falha e característica da rede, além de possuir mecanismos de tolerância à falhas.

Escalabilidade

Escalabilidade em *Volunteer Computing* é principalmente influenciada pelo modelo de distribuição adotado, centralizado (Figura 2.8) ou distribuído (Figura 2.9). No sistema centralizado o servidor poderá se tornar o gargalho do sistema se o número de clientes crescer continuamente e se o servidor não for eficiente o bastante na distribuição de tarefas, além do servidor ser um ponto único de falhas, o que compromete a robustez do sistema. Modelos distribuídos, por não possuir um ponto central de controle, são mais escaláveis, pois o escalonamento de tarefas é feito de forma distribuída. Porém sistemas distribuídos possuem desempenho inferior a sistemas centralizados já que o escalonamento é realizado apenas com informações locais, sem uma visão geral do sistema.

Participação Voluntária

Já que os voluntários podem entrar e sair de uma sessão de processamento quando acharem conveniente, o sistema deverá prover formas de incentivar a doação de recursos computacionais baseados em alguma política de recompensa caso os voluntários não estejam interessados em doar recursos computacionais para um projeto. Geralmente os voluntários doam recursos computacionais por vontade de contribuir com um determinado projeto, mas podem existir diferentes políticas de incentivo como a do projeto OurGrid (http://www.ourgrid.org/) que utiliza a metodologia de comunicação *Peer-toPeer*. Uma instituição voluntária que doa recursos computacionais para outras instituições ou projetos obtém créditos. A instituição que possuir mais crédito é favorecida na hora de encontrar outros voluntários que pertencem à rede *Peer-to-Peer* para processar suas informações.

2.5. Modelagem Numérica de Terreno (Estudo de caso)

Um Modelo Numérico de Terreno (MNT) é uma representação matemática computacional da distribuição de um fenômeno espacial que ocorre dentro de uma região da superfície terrestre [Namikawa et al., 2003]. Um MNT pode representar diversas informações geográficas de um terreno como: informações geológicas, umidade do ar, informações de altitude do terreno, dados geofísicos, etc. Através de um MNT é possível obter informações mais ricas de um dado geográfico como a geração de imagens sombreadas ou em tom de cinza, informações de área, volume, etc. A Modelagem Numérica de Terreno é dividida em três partes: (a) a amostragem dos dados; (b) a modelagem propriamente dita dos dados; e (c) o uso deste modelo em alguma aplicação [Felgueiras, 2001].

2.5.1. Amostras

A amostragem compreende a aquisição de um conjunto de informação que representa a variação de um fenômeno espacial de interesse [Câmara e Felgueiras]. As amostras geralmente são organizadas com base em três coordenadas (x,y,z), onde (x,y) representam a posição do fenômeno de interesse e (z) representa o valor deste fenômeno. No caso de representação de altitude, (x, y) representaria a posição (latitude, longitude) e (z) representaria a altitude daquele ponto. Existem três tipos de amostragem: (a) regular, em que tanto x quanto y são regularmente espaçados; (b) semi-regular, pelo menos uma coordenada, x ou y, é regularmente espaçada; e (c) irregular,

em que não há nenhuma relação entre os espaçamentos usados nas coordenadas x e y. A Figura 2.10 é um exemplo de uma amostra de dados de altitude com espaçamento regular, onde a coluna em que um valor se encontra representa a posição x, a linha em que um valor se encontra representa a posição y e o valor propriamente dito representa a altitude naquela posição (z).

1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	3	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
5	5	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
5	5	5	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	3	3	3	3	3	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	3	3	3	3	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4

Figura 2.10. Amostra de dados de altimetria regularmente espaçados (Fonte: [Câmara e Felgueiras])

2.5.2. Modelagem

A modelagem é constituída pela criação de uma estrutura de dados que irá representar a amostra de dados de forma digital e da definição de uma superfície de ajuste que irá ajustar valores nas posições onde não foi realizada a coleta de dados. Este ajuste poderá ser feito de forma global ou local. No ajuste global é aplicada uma interpolação sobre toda a amostra de dados de uma só vez. O ajuste local leva em consideração apenas os pontos mais próximos. Os dois modelos mais conhecidos são o Modelo de Grade Regular e Modelo de Grade Irregular Triangular.

Modelo de Grade Regular Retangular

Existem duas formas de gerar uma grade regular. A primeira é utilizando um ajuste global onde é ajustada uma função polinomial através da técnica de regressão polinomial. Esta técnica não é muito utilizada pelo poder computacional exigido para realizar a regressão em um conjunto de dados muito grande [Câmara e Felgueiras]. Outra forma de geração da grade regular é com base no ajuste local. Este ajuste geralmente utiliza técnicas de interpolação local conhecida como média móvel. Os principais tipos de média móvel são: interpolação por vizinho mais próximo,

interpolação por média simples e interpolação por média ponderada. Na Figura 2.11 é ilustrada um exemplo de grade regular gerado a partir de uma amostra regularmente espaçada que representa informações de altitude de uma área.

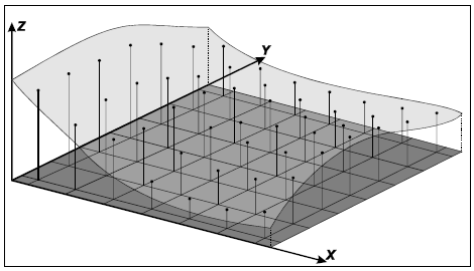


Figura 2.11. Grade Regular (Fonte: [Namikawa, 1995]).

Modelo de Grade Irregular Triangular

No Modelo de Grade Irregular Triangular a superfície que é ajustada é constituída por triângulos. Geralmente os vértices dos triângulos correspondem aos pontos contidos na amostra. Neste modelo não é necessário nenhuma técnica de ajuste global ou local já que a grade é gerada apenas com os pontos presentes na amostra, sem a necessidade de estimar os demais pontos ausentes [Namikawa et al., 2003]. A geração da grade irregular e manipulação da estrutura de armazenamento são mais complexas que em grades regulares. Além disto, esta técnica pode gerar problema com relação ao espaço de armazenamento e tempo de processamento para grandes áreas [Câmara e Felgueiras]. O método de geração de grade irregular triangular mais conhecido é a Triangulação de Delaunay [Wikipedia, 2008b].

2.6. Regressão Não-Linear (Estudo de caso)

Regressão Não-Linear é uma modelagem matemática que procura descrever a relação existente entre fenômenos observados. De acordo com Rawlings et al. (1998) "Modelagem refere ao desenvolvimento de uma modelagem matemática que descreve, de alguma forma, o comportamento randômico de uma variável de interesse". A modelagem não-linear é utilizada para descrever o comportamento de variáveis independentes cujo relacionamento com a variável dependente é mais bem representada de forma não-linear. A Estatística têm a importante missão de encontrar a relação, se houver alguma, em um conjunto de variáveis quando pelo menos uma destas variáveis for randômica, estando sujeita a flutuações aleatórias e possivelmente erros de medição [Seber e Wild, 1989].

A relação entre as variáveis é descrita através de funções. Em funções polinomiais unidimensionais, a flutuação da variável dependente (y) está relacionada à flutuação da variável independente. A variável dependente pode ser o preço de uma ação na bolsa de valores, a taxa de crescimento de um tipo particular de tumor, etc. Enquanto a variável independente corresponde a uma unidade particular na qual a observação foi feita, como o período no qual o preço da ação foi registrado ou uma unidade experimental na qual o crescimento do tumor foi registrado. Em funções polinomiais bidimensionais, a variável dependente está relacionada com duas variáveis independentes. Desta forma, a representação visual da função não mais é constituída por uma linha como nas funções unidimensionais, mas por uma superfície. A Figura 2.11 é um exemplo de uma função polinomial bidimensional onde a variável dependente z varia de acordo com as variáveis independentes x e y.

Especificamente no estudo de caso desenvolvido neste projeto, a regressão não-linear foi utilizada para descrever a relação entre duas variáveis independentes (latitude e longitude) e uma variável dependente (altitude). A modelagem matemática desenvolvida possibilita que os coeficientes de funções polinomiais bidimensionais, de diferente grau em x e em y, que represente a variação de altitude do relevo de uma área qualquer sejam estimados.

3. Materiais e Métodos

3.1. Modelagem do Sistema

O modelo de distribuição de processamento proposto atende aos desafios da área de Sistemas Distribuídos apresentados na sessão 2.1.3 (heterogeneidade, extensibilidade, segurança, escalabilidade, tolerância a falhas, concorrência e transparência), que são genéricos e que fazem parte que todos os tipos de sistemas distribuídos. A heterogeneidade das diversas máquinas e a escalabilidade do sistema, que é a capacidade do sistema de suportar um número crescente de componentes. Atende também os desafios da área de *Volunteer Computing* apresentados na sessão 2.4.2 (volatilidade, ambiente dinâmico, falta de confiança, falhas, escalabilidade e participação voluntária), específicos dos sistemas que utilizam computação voluntária,desafios como ambientes dinâmico e falta de confiança.

De uma forma geral, a metodologia proposta segue a arquitetura de sistema da área de Sistemas Distribuídos apresentada na sessão 2.1.2, em que um servidor é responsável por atender os pedidos dos diversos clientes. Esta definição está diretamente ligada à forma de comunicação utilizada. Um dos tipos de comunicação utilizado nesta arquitetura é conhecido como comunicação cliente/servidor em que o servidor se comunica com todos os clientes e estes clientes só se comunicam com o servidor, nenhum cliente tem conhecimento da existência de outros clientes. A metodologia de distribuição de processamento criada não segue o Modelo de Execução Centralizado proposto por Choi et al. (2007), ilustrado na Figura 2.8, em que um cliente envia uma tarefa a ser processada pelo servidor central que é encarregado de dividir esta tarefa em sub-tarefas e distribuí-las entre os voluntários. O modelo deste trabalho utiliza um modelo de execução em que:

- Não existe o papel de cliente proposto no trabalho de Choi et al. (2007);
- O servidor é desenvolvido para realizar uma tarefa específica, o que dispensa a necessidade de um cliente que informa o servidor o que ele deve fazer;
- O servidor, ao invés de servir ao cliente, passa “servir” aos voluntários com tarefas a serem processadas;
- Os voluntários requisitam tarefas do servidor, como os clientes requisitam algum serviço ao servidor no modelo tradicional.

Neste trabalho o componente que possui a função de servidor é chamado de Coordenador por estar em concordância com sua função, já que ele irá coordenar a distribuição de processamento e os demais componentes, que têm a função de clientes, são chamados de Agentes. De forma geral, o modelo possui: (a) um Coordenador; (b) um Repositório de Dados onde são armazenadas informações utilizadas pelo Coordenador como tarefas a ser processadas, resultados, configurações, etc.; (c) o meio de comunicação, que neste caso é a Internet e (d) os Agentes. Estes Agentes podem estar ativos (conectados ao Coordenador) ou inativos. A estrutura geral do modelo é ilustrada na Figura 3.1.

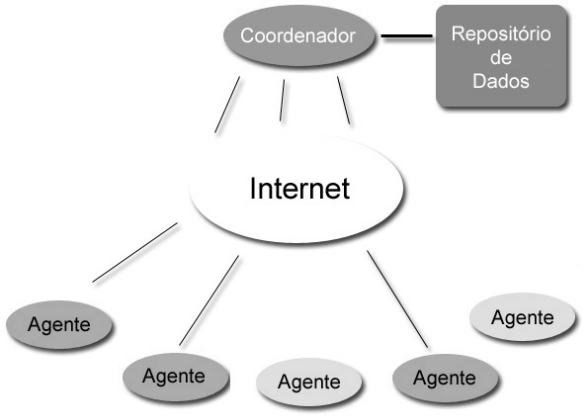


Figura 3.1. Estrutura Geral do Modelo de Distribuição de Processamento

As características estáticas e dinâmicas do modelo, assim como a interação entre seus componentes são especificadas utilizando diagramas UML (http://www.uml.org/), usado na especificação de sistemas orientados a objeto. Foram utilizados os seguintes diagramas:

- Diagrama de Caso de Uso – especifica a interação entre usuário e o sistema.
- Diagrama de Classe – representa a estrutura e relações entre as classes de um sistema.
- Diagrama de Estado – ilustra os possíveis estados que um determinado objeto assume durante a execução do sistema.
- Diagrama de Atividade – um gráfico de fluxo, semelhante ao de Estado, mas que representa um fluxo de execução e interação de dois ou mais objetos.
- Diagrama de Seqüência – descreve uma situação específica na execução do sistema e como um grupo de objetos se relacionam nesta situação.

- Diagrama de Componente – representa componentes que podem ser códigos-fonte, subsistemas de implementação, aplicações, arquivos, etc.
- Diagrama de Execução – mostra a arquitetura física de hardware e componentes de software do sistema.

De acordo com a Estrutura Geral do Modelo ilustrada na Figura 3.1, é possível notar que ele possui dois tipos de aplicações a serem modeladas, o Coordenador e Agente. Os modelos de ambas as aplicações possuem três tipos distintos de componentes. O primeiro tipo são os componentes que pertencem ao núcleo do sistema. Isto quer dizer que, independente do problema a que o modelo será adaptado, estes componentes irão permanecer com pouca ou nenhuma modificação. São componentes cuja funcionalidade independem do problema a ser resolvido. Um exemplo de componente do núcleo é o componente responsável pela troca de mensagens entre Coordenador e Agente. Este componente obtém uma operação do repositório de dados, envia esta operação a um Agente, aguarda a resposta, grava o resultado no repositório de dados, buscar outra operação e recomeça o ciclo. Esta função é independente do problema a ser resolvido sendo a mesma para qualquer aplicação desenvolvida com base na metodologia proposta. O segundo tipo de componente é aquele adaptado a cada tipo de problema a ser resolvido. Cada aplicação desenvolvida com base na metodologia irá implementar os componentes adaptáveis para um problema específico. Um exemplo de componente adaptável é o responsável por acesso aos dados (ou operações). Já que os dados podem ser acessados de diferentes formas e os próprios dados podem possuir diferentes formatos (arquivo XML, arquivo modo texto, arquivo binário, banco de dados, etc.) necessitando assim adaptar o módulo de acesso a dados. O terceiro tipo são os componentes opcionais que poderão ou não ser utilizados, dependendo do tipo de processamento a ser distribuído. Desta forma, os componentes adaptáveis e opcionais irão possuir algumas especificações básicas, porém as funcionalidades mais detalhadas só poderão ser descritas no desenvolvimento de uma aplicação específica.

A descrição das características da metodologia será separada por aplicação. Primeiro serão descritas as especificações do Coordenador, seguida das especificações do Agente e por último será apresentado o modelo de interação entre Coordenador e Agentes.

3.1.1. Coordenador

Uma característica, tanto do Coordenador quanto do Agente, é que ambos não possuem muita interação com o usuário. Tendo em vista que o Coordenador foi desenvolvido para distribuir um processamento específico, ele precisa de pouca ou nenhuma intervenção do usuário (administrador) para que a distribuição seja feita. A interação com o usuário, no caso do Coordenador se resume a ativar e desativar o Coordenador, além de consultas sobre o andamento da distribuição, que nesta metodologia é representado pelo número de operações processadas e outras informações que serão detalhadas posteriormente. O diagrama de Caso de Uso da Figura 3.2 ilustra as interações entre o administrador e o sistema (Coordenador).

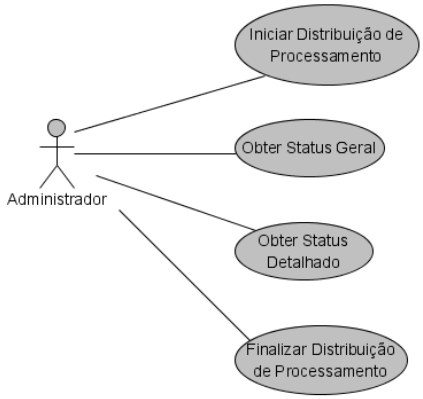


Figura 3.2. Caso de Uso do Coordenador

A atividade principal que o Coordenador desempenha é iniciar a distribuição de processamento (Figura 3.2). As principais tarefas que o Coordenador deverá desempenhar para distribuir uma determinada demanda de processamento:

- criar uma conexão primária através da qual os Agentes irão se conectar ao Coordenador;
- criar uma conexão secundária através da qual cada Agente poderá ser autenticado, deixando a conexão primária livre para receber novos pedidos de conexão de outros Agentes;
- autenticar os Agentes (voluntários);
- obter a próxima operação a ser processada;
- enviar esta operação ao Agente;
- aguardar o resultado do Agente;

- gravar o resultado enviado pelo Agente e reiniciar o ciclo obtendo a próxima operação a ser processada.

Este ciclo irá se repetir até que uma das partes (Coordenador ou Agente) mostre intenção de encerrar a sessão de troca de mensagens (enviar operação e receber resultado). Com base nestas tarefas, foram identificados os principais módulos, assim como suas funções. Estes módulos estão representados através do Diagrama de Classe que está ilustrado na Figura 3.3. No Anexo A.1 é apresentado o Diagrama de Classe do Coordenador completo, com atributos e métodos. Os módulos (ou classes) em azul pertencem ao núcleo do sistema, os módulos verdes pertencem ao segundo tipo que são os módulos adaptáveis e os módulos amarelos são opcionais.

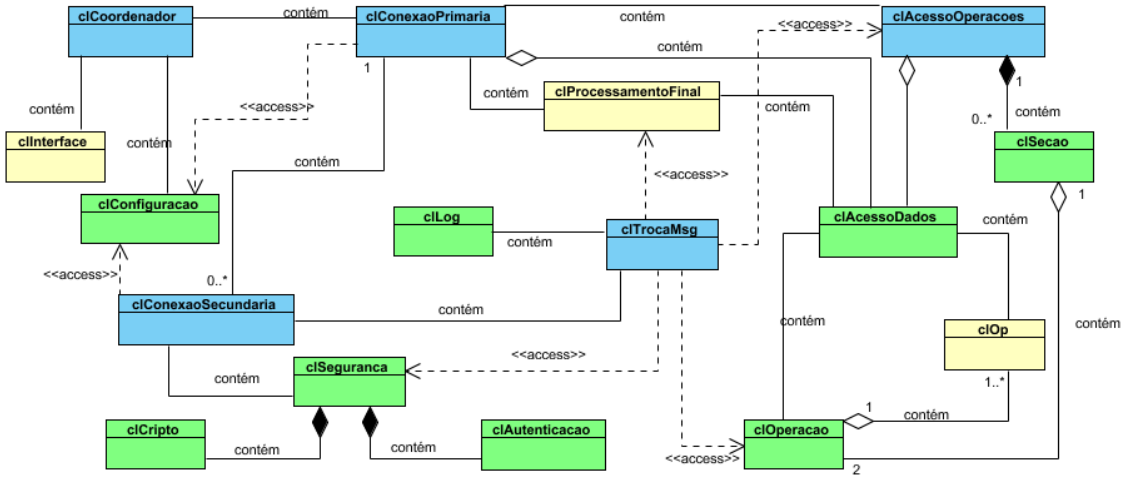


Figura 3.3. Diagrama de Classe do Coordenador.

clCoordenador

Módulo responsável por fazer interface entre o usuário, que no caso do Coordenador opera como administrador do sistema. É este módulo que oferece ao usuário todas as funcionalidades do sistema e sabe como obter cada funcionalidade. Ao ser ativado, configura o Coordenador de acordo com as especificações constantes do módulo *clConfiguracao* e inicia a conexão primária através do módulo *clConexaoPrimaria*, que é executado em paralelo e independente da execução das demais funções do *clCoordenador*. Isto para que o módulo *clCoordenador* fique livre para atender às requisições do administrados através dos comandos executados por ele. Outra tarefa deste módulo é obter o *status* de todo o sistema. Na metodologia são especificados dois tipos distintos de *status*, um descreve o estado geral do sistema, que inclui quantos Agentes estão conectados ao Coordenador e o número de operações processadas até o

momento. O outro tipo detalha, além do estado geral, a porta que cada Agente está usando para se conectar ao servidor e quantas operações cada Agente conectado já processou. É importante notar com relação ao *status* que cada Agente conectado é diferente de voluntário conectado. Tendo em vista que cada voluntário possui informações cadastradas no Coordenador para poder se autenticar, cada voluntário poderá ativar mais de uma instância da aplicação Agente. Neste caso, no *status* detalhado obtido do sistema, a porta usada e o número de operações processadas é informado por aplicação Agente conectado e não por voluntário. Estes dois tipos de *status* são básicos, assim como todas as operações discutidas na descrição da aplicação Coordenador como da aplicação Agente e poderão ser modificados ou estendidos quando o modelo for utilizado como base para o desenvolvimento de um sistema específico. No Diagrama de Estado da Figura 3.4 são ilustrados todos os estados assumidos por este módulo. Os estados azuis representam o fluxo principal, os verdes são fluxos alternativos e podem não ser executados e os estados representados pela cor vermelha são relacionados à finalização do sistema. Este padrão de cores é adotado para os demais Diagramas de Estados que serão apresentados neste trabalho.

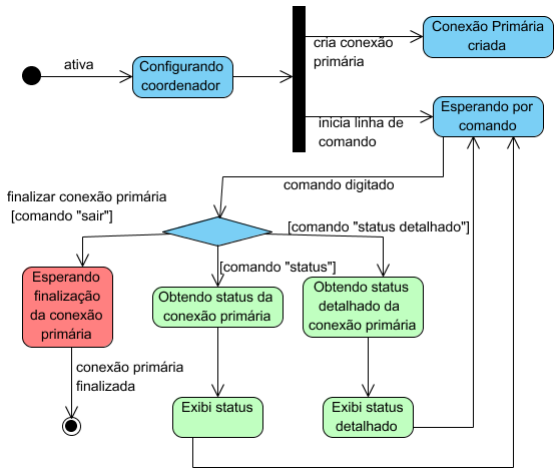


Figura 3.4. Diagrama de Estado do módulo clCoordenador

clConexaoPrimaria

Uma característica da metodologia proposta é que a troca de mensagens entre o Coordenador e os diversos Agentes conectados é feita em paralelo. Esta característica é desejável em sistemas distribuídos onde o servidor é centralizado. Desta forma, a função principal deste módulo é receber um pedido de conexão de um Agente, aceitar este pedido e criar uma *thread* para que a autenticação e troca de mensagens seja feita em paralelo. Sem o paralelismo na troca de mensagens, os Agentes seriam atendidos

seqüencialmente, o que tornaria o servidor o gargalo do sistema à medida que o número de Agentes conectados a ele aumentasse, comprometendo a escalabilidade do sistema. Dado que o Coordenador terá diversos Agentes atendidos em paralelo, o módulo de conexão primária terá duas funções primordiais. Uma função é servir como ponto de controle para os diversos módulos *clConexaoSecundaria* que serão executados em paralelo. A outra função é ser o meio através do qual os Agentes irão estabelecer a primeira das duas conexões com o Coordenador.

Este módulo possui três pontos de controle (variáveis globais) duas variáveis mutex, para implementar exclusão mútua [Tanenbaum, 2003b], e um contador global:

- **mutex para acesso ao arquivo de operações**: como será comentado na sessão referente ao módulo *clAcessoDados*, há duas possibilidades para implementar o arquivo de operações. Poderá haver dois arquivos, um para armazenar as operações ainda não processadas e outro para as operação já processadas; outra possibilidade é utilizar um único arquivo de operações, que será atualizado à medida que as operações vão sendo processadas. Nesta segunda possibilidade poderá haver condição de corrida [Tanenbaum, 2003b] por acesso ao arquivo de operações e assim haverá a necessidade de um único mutex que será compartilhado por todas as conexões secundárias.
- **mutex para acesso ao arquivo de log:** Tendo em vista que, apesar dos módulos *clConexaoSecundaria* acessarem o arquivo de log para registrar informações, não haverá condição de corrida ou perigo de resultados inconsistentes, pois os processos irão apenas escrever no arquivo, não havendo leituras. Este fato não descarta a necessidade de exclusão mútua já que poderia haver uma mistura das escritas de diversos processos em função da troca de contexto do sistema operacional.
- **contador global para o número de operações já processadas:** é um contador que será compartilhado pelos diversos módulos *clConexaoSecundaria*. Cada módulo irá incrementar este contador à medida que as operações vão sendo processadas. Desta forma o módulo *clConexaoPrimaria* terá o controle sobre a quantidade de operações processadas pelo sistema como um todo.

Como comentado, este módulo possui outra função primordial, que é servir como meio através do qual os Agentes irão realizar a primeira conexão com o Coordenador. Como um dos objetivos desta metodologia é distribuir processamento com eficiência e

alto desempenho, optou-se por utilizar uma forma de comunicação inter-processos (entre aplicações) em nível mais baixo. A comunicação é feita através da API do sistema operacional chamada *socket*. Diversas linguagens de programação possuem suporte para esta API. Com ela a comunicação entre aplicações se dá através de troca de mensagens modo texto, que é mais eficiente que a troca de informações utilizando *middleware* como CORBA e Java RMI [Saltzer et al., 1984]. Além disto, utilizando troca de informações no formato de mensagens modo texto, não há a possibilidade de incompatibilidade entre linguagens de programação, compiladores, sistemas operacionais e versões de sistema operacional já que os principais sistemas operacionais possuem suporte para a comunicação inter-processos baseada nesta API.

Dado que a forma de comunicação entre aplicações será através da API *Socket* e que ela utiliza a metodologia de comunicação em que a comunicação se dá entre dois *socket*'s, um em cada aplicação, o módulo *clConexaoPrimaria* possui dois *socket*'s. Um será o *socket* do Coordenador e o outro irá representar o do Agente que estabelecer conexão com o Coordenador. O *socket* do Coordenador possuirá endereço de rede e porta pré-definidos e conhecidos pelos Agentes. Quando um Agente se conectar ao *socket* do Coordenador, as informações necessárias para enviar e receber mensagens para este Agente será armazenada no *socket* Agente, que possui uma porta diferente da utilizada para estabelecer conexão com o *socket* do Coordenador. Esta é uma funcionalidade linguagem de programação que redireciona a conexão para outra porta e assim libera a porta original para receber outros pedidos de conexão.

Este módulo poderá informar dois tipos distintos de *status*, como comentado anteriormente, o *status* global e o *status* detalhado. No *status* global, a variável responsável por armazenar tal *status* apenas conterá a quantidade de Agentes conectados (que é igual à quantidade de conexões secundárias ativas) e o número global de operações processadas. No *status* detalhado, esta variável, além de conter as informações anteriores, será composta pelas informações detalhadas fornecidas por cada módulo *clConexaoSecundaria*. Ou seja, depois de preencher a variável com as informações globais, as informações detalhadas de cada módulo *clConexaoSecundaria* são concatenadas nesta variável que é retornada para o módulo *clCoordenador* que irá exibi-las.

Este módulo possui um método, que será executado em paralelo com o módulo *clCoordenador*, que é o responsável por receber pedidos de conexão dos Agentes e criar dinamicamente o módulo *clConexaoSecundaria*. Porém, antes de criar os módulos para

conexão secundária dinamicamente é verificado se o número máximo de conexões secundárias foi atingido. Caso tenha sido atingido, o pedido recém recebido deve ser negado e o módulo *clConexaoSecundaria* não será criado. Se o número máximo não foi atingido, o pedido de estabelecimento de conexão é aceito e a conexão secundária é estabelecida. Este procedimento de receber pedido e criar a conexão secundária é executado em um laço infinito. Para que o módulo possa ter controle sobre este método, que possui execução independente do restante o módulo, é necessário uma variável compartilhada entre o módulo e este método. Esta variável é responsável por informar ao método quando é o momento de encerrar o laço. Quando o administrador do Coordenador deseja finalizar a aplicação, o módulo *clCoordenador* informa ao módulo *clConexãoPrimária* a finalização da aplicação e através da variável de finalização compartilhada o módulo *clConexãoPrimária* poderá enviar o aviso de finalização para a *thread* para que ela finalize o laço. O método que finaliza o módulo *clConexaoPrimaria* inicializa a variável *finalizar* e entra em modo de espera até que a *thread* altere a variável novamente, o que significa que o laço foi interrompido e que a *thread* foi finalizada. O Diagrama de Estado da Figura 3.5 ilustra os estados assumidos durante a execução da *thread* responsável por receber pedidos de conexão e instanciar dinamicamente os módulos *clConexaoSecundaria*. Uma característica do laço que é executado na *thread* é ser não bloqueante, isto é,,ela permanece bloqueada até que um pedido de conexão seja feito.

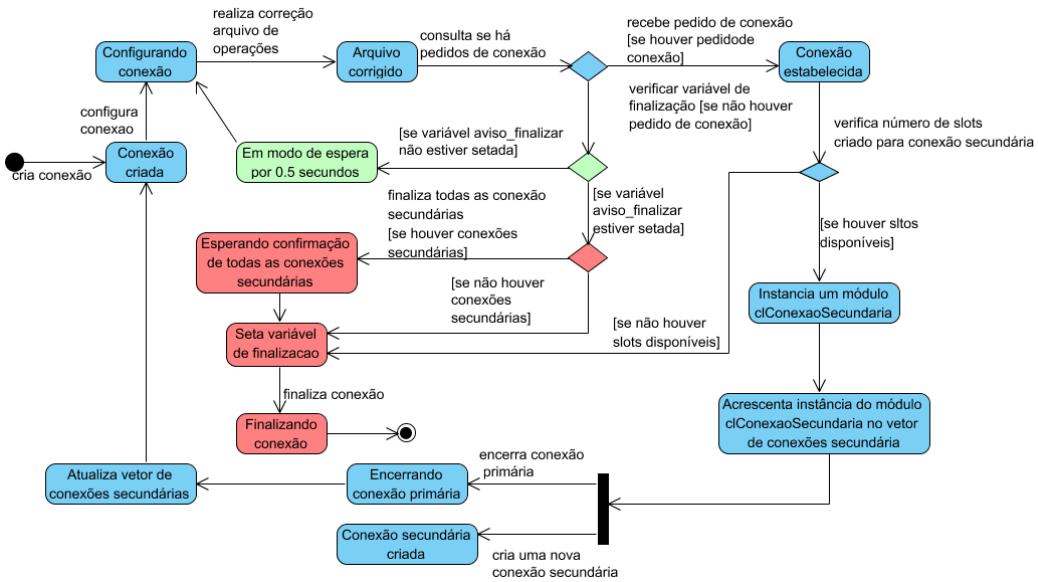


Figura 3.5. Diagrama de Estado do módulo clConexaoPrimaria

clConexaoSecundaria

Este módulo é executado em paralelo no terceiro nível de execução. O primeiro nível corresponde à execução do módulo *clCoordenador* que ativa a conexão primária e executa o método para receber os comandos do usuário. O módulo *clConexaoPrimaria* executa no segundo nível. No terceiro nível é executado o módulo *clConexaoSecundaria*, que é criado para cada Agente que estabelece a conexão primária com o Coordenador. A principal função deste módulo é possibilitar que todos os Agentes conectados ao Coordenador sejam autenticados e a troca de mensagem seja iniciada. Todos os módulos serão executados de forma independente dos demais módulos *clConexaoSecundaria* e dos módulos *clCoordenador* e *clConexaoPrimaria*. Para que possa ficar clara a execução em paralelo destes módulos, a Figura 3.6 ilustra os três níveis de execução e a Figura 3.7 ilustra, através do Diagrama de Atividades, o paralelismo na execução dos módulos *clCoordenador*, *clConexaoPrimaria* e *clConexaoSecundaria*.

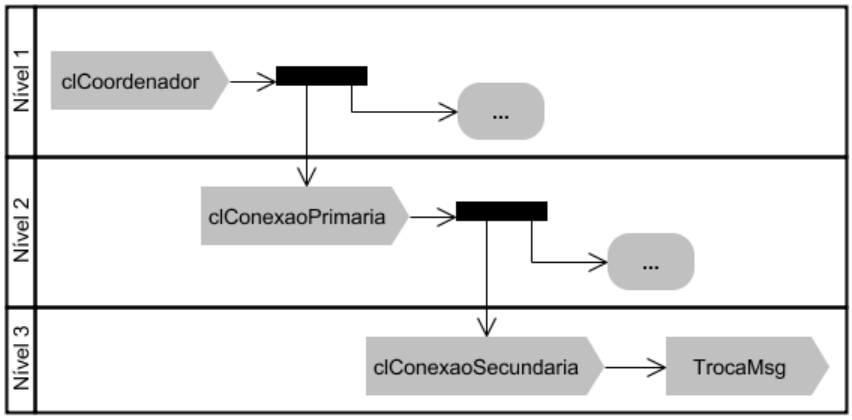


Figura 3.6. Níveis de paralelismo de execução da aplicação Coordenador

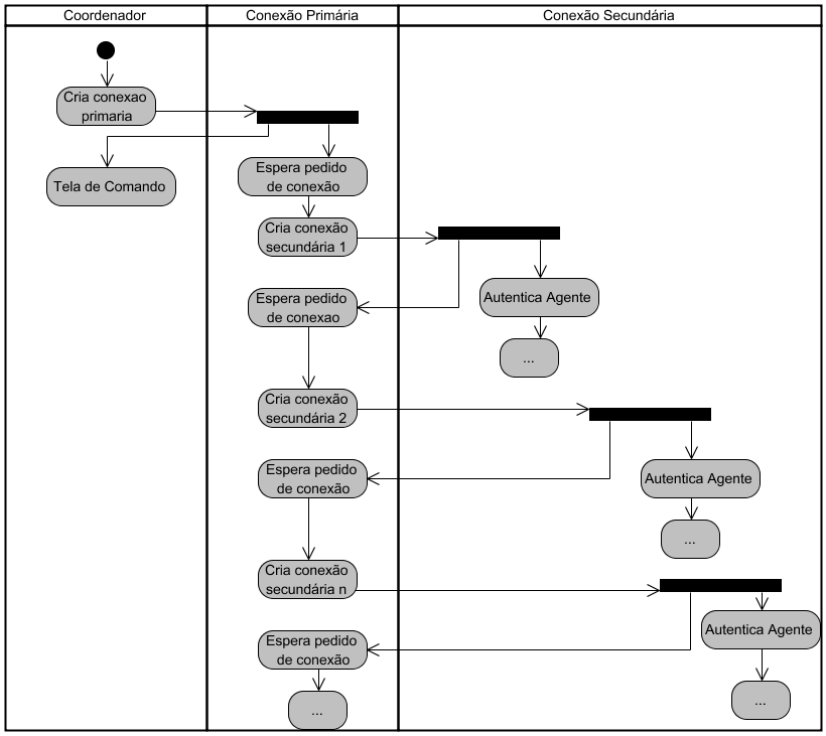


Figura 3.7. Processo de criação dos módulos clConexaoSecundaria dinamicamente

Na criação deste módulo são passadas algumas informações que serão utilizadas por ele e pelo módulo *clTrocaMsg*, que será criado para realizar a troca de mensagens entre Coordenador e Agente. Algumas destas informações são:

- informações de configuração (ver sessão que descreve o módulo clConfiguracao);
- mutex para acesso ao arquivo de operações;
- mutex para acesso ao arquivo de log;
- variável a ser atualizada com o número global de operações processadas;
- referência ao módulo *clProcessamentoFinal* (caso este módulo seja utilizado);
- o *socket* através do qual este módulo irá se comunicar com o Agente.

Apesar da caracterização e classificação de Desktop Grid (Choi et al. 2007) especificar que os voluntários são anônimos, diversos projetos como SETI@Home, Folding@Home utilizam alguma forma de autenticação dos voluntários. Neste modelo também é utilizado uma forma de autenticação para que os voluntários possam ser identificados, o que não elimina a falta de confiança entre Agente e Coordenador. A falta de confiança é um dos desafios da computação voluntária. A falta de confiança não é eliminada porque a autenticação e cadastramento dos voluntários são feitos utilizando informações básicas destes voluntários, como e-mail, nome e senha. Desta forma, as

duas tarefas principais do módulo *clConexaoSecundaria* são realizar a autenticação do Agente e iniciar a troca de mensagens. Na autenticação, o Coordenador envia o pedido de nome e senha para o Agente. O nome e senha que será enviado para o Coordenador deve ser criptografado para evitar, no caso da mensagem ser interceptada, a revelação da senha do voluntário. Seguindo uma metodologia que geralmente é adotada em autenticações realizadas em aplicações, o voluntário terá um número pré-estabelecido de chances de realizar a autenticação. Caso o voluntário informe nome ou senha incorretamente uma quantidade de vezes além do permitido, a conexão deve ser finalizada.

Assim como no módulo *clConexaoPrimaria*, este módulo possui um método principal, que será executado em paralelo. Neste caso também existe a necessidade de uma variável compartilhada entre o módulo e o método. A metodologia é a mesma adotada no módulo *clConexaoPrimaria* em que uma variável é alterada pelo módulo para sinalizar algum tipo de aviso para o método. A diferença é que não será o método que irá verificar esta variável e sim o método do módulo *clTrocaMsg*. Para isto, uma referência da variável deve ser passada para o módulo *clTrocaMsg*. Este módulo poderá finalizar sua execução a qualquer momento, independente da execução da aplicação Coordenador. Isto porque cada módulo que é instanciado é responsável por atender um Agente especificamente e assim que a conexão entre este módulo e a aplicação Agente for encerrada (os motivos para encerramentos da conexão serão apresentados na sessão referente ao módulo clTrocaMsg) este módulo é encerrado. Antes de se auto-encerrar, o método principal altera a variável de finalização para que o módulo possa ter conhecimento sobre o estado da conexão com cada Agente. Este módulo também possui um campo *status* que guarda informações sobre seu estado atual. O *status* contém a porta utilizada para comunicação com o Agente, o nome do voluntário, em que estado o Agente se encontra (autenticando ou trocando mensagens) e quantas operações o Agente já processou. Os *status* dos módulos *clConexaoSecundaria* irão compor o *status* detalhado, caso seja solicitado pelo usuário da aplicação. A Figura 3.8 ilustra os estados assumidos pelo módulo *clConexaoSecundaria*.

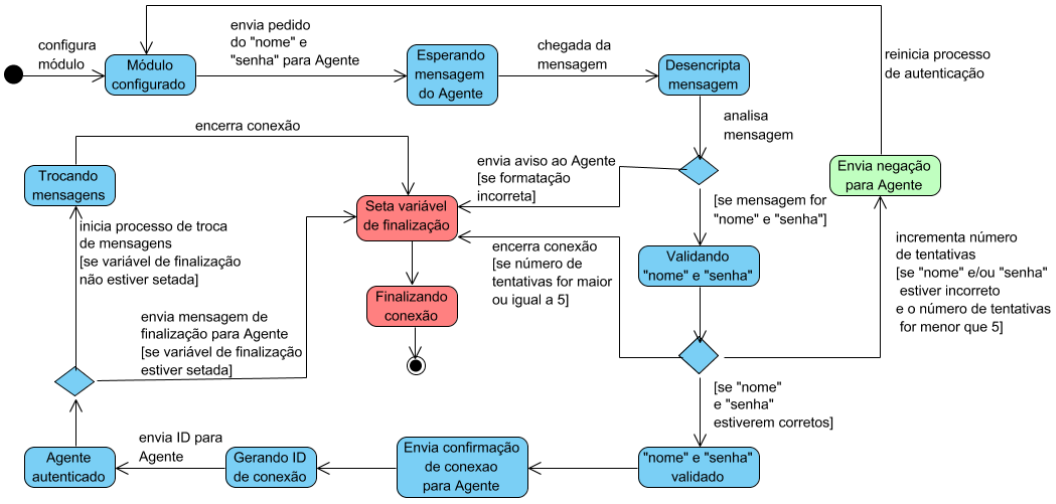


Figura 3.8. Diagrama de Estado no módulo clConexaoSecundaria

clTrocaMsg

As duas tarefas principais deste módulo são enviar operações a serem processadas para um determinado Agente e entrar em modo de espera até que este Agente retorne o resultado do processamento da operação. Para que esta tarefa possa ser realizada, este módulo precisa ser configurado para conhecer o endereço para onde enviar as operações, qual ID de conexão a ser utilizado para troca de mensagens, dentre outras. A lista de todas as informações que este módulo irá receber do módulo *clConexaoSecundaria* e que servirá para configurar o módulo clTrocaMsg, está descrita a seguir:

- socket do Agente – contém o identificador do ponto de comunicação através do qual este módulo se comunica com o Agente.
- ID de conexão – usado para validar as mensagens recebidas do Agente;
- mutex para acesso ao arquivo de operações – mutex passado para o módulo clAcessoDados para acesso ao arquivo de operações;
- mutex para acesso ao arquivo de log – mutex passado para o módulo clLog para que as informações de log possam ser gravadas no arquivo de log;
- variável finalizar – um apontador para a variável finalizar do módulo *clConexaoSecundaria* para que este possa informar ao módulo a hora de finalizar a troca de mensagens;
- nome usuário – passado para os módulos *clAcessoDados*, para que ele registre quais usuários processaram quais operações e para o *clLog*, para registrar informações sobre a conexão estabelecida por cada voluntário;

- módulo *clSeguranca* – referência para o módulo *clSeguranca*, que foi criado no módulo *clConexaoSecundaria*, para realizar operações como criptografia de mensagens, se necessário;
- número global de operações processadas – referência para a variável com o número global de operações já processadas durante a execução do Coordenador, todos os módulos *clTrocaMsg* terão acesso a esta variável;
- módulo *clProcessamentoFinal* – as características e funcionalidades deste módulo opcional serão apresentadas na sessão referente ao módulo *clProcessamentoFinal*;

A primeira tarefa deste módulo é configurar o módulo *clAcessoDados* passando o mutex para acesso a arquivo de operações. Desta forma, toda vez que o módulo responsável pelo acesso a dados acessar o arquivo de operações, ele irá tentar travar o mutex para ter acesso exclusivo ao arquivo. Depois da configuração, o módulo requisita a próxima operação. Se não houver mais operações ou se houver um erro qualquer na obtenção da próxima operação, o Agente será avisado e a conexão será encerrada. Caso contrário, a nova operação que é representada por um módulo *clOperacao* é formatada. Como a metodologia sugere o uso da API Socket para a comunicação entre processos, neste caso, formatar a operação significa montar uma mensagem modo texto com as informações contidas no módulo *clOperacao* retornada pelo módulo *clAcessoDados*. Caso outras tecnologias de comunicação inter-processos sejam utilizadas como RMI e CORBA, formatar a operação significa montar uma mensagens a ser enviada ao Agente de acordo com o que é especificado pelo *middleware* de comunicação utilizado. O próximo passo é acrescentar o ID de conexão ao início da mensagem para possa ser validada pelo Agente. Então a mensagem é enviada e o módulo permanece bloqueado até que a mensagem contendo o resultado do processamento seja recebida.

Para registrar as informações relativas ao tempo gasto na transmissão das mensagens (envio operação e recebimento do resultado) e no processamento da operação, são medidos o tempo decorrido entre o envio da operação e o recebimento do resultado. Considerando que, concatenado com a mensagem que contém o resultado da operação enviada pelo Agente está o tempo gasto durante o processamento, é possível calcular o tempo gasto na transmissão das mensagens e de processamento da operação separadamente. Os tempos são calculados de acordo com a Equação (1) e Equação (2) apresentadas a seguir.

$$T = t2 - t1 \quad (1)$$

$$tt = T - tp \quad (2)$$

- T = tempo total do processo
- t1 = tempo inicial, registrado no momento do envio da operação
- t2 = tempo final, registrado após o recebimento do resultado
- tp = tempo de processamento, registrado pelo Agente e enviado concatenado com a mensagem de resultado do processamento
- tt = tempo de transmissão, representa o tempo gasto na transmissão da operação e do resultado

Dada a volatilidade da participação do voluntário na sessão de processamento, deve ser implementado um mecanismo que possibilite a desconexão do Agente sem grande custo computacional. A forma encontrada para possibilitar essa rápida saída é permitir ao Agente enviar, junto a qualquer resultado, um aviso sobre a intenção de finalizar a conexão. Ao perceber o aviso de finalização junto à mensagem, o Coordenador não irá enviar outra operação ao Agente e a conexão será encerrada. O Coordenador também possui a funcionalidade de, ao invés de enviar uma operação para um Agente, enviar um aviso de finalizar e em seguida finalizar a execução do módulo. Antes da finalização, são registrados informações sobre a interação entre Coordenador e Agente. Estas informações serão discutidas em detalhes na sessão relativa à descrição do módulo *clLog*. Outra motivação para a finalização do módulo é a interrupção da comunicação entre o módulo e o Agente. Os motivos da interrupção podem ser vários, tal como interrupção da execução do Agente, problemas no meio de comunicação Neste caso o módulo *clTrocaMsg* irá informar ao módulo *clAcessoDados* que a operação não foi processada para que sejam tomadas as devidas providências com relação ao arquivo de operações. Estas providências serão detalhadas na sessão de descrição do módulo *clAcessoDados*.

Caso nenhum imprevisto ocorra, este módulo irá receber a mensagem contendo o resultado do processamento e armazenar o resultado no módulo *clOperacao* que será passado ao módulo *clAcessoDados* para que o arquivo de operações seja atualizado. Em seguida, o número local (número de operações processada pelo Agente conectado ao módulo) e o número global de operações são atualizados. Antes que o processo de obter

e enviar uma nova operação ao Agente seja reiniciado, este módulo verifica se o módulo *clConexaoSecundaria* enviou um aviso de finalização através da variável compartilhada "finalizar". Se esta variável estiver alterada, ao invés de reiniciar o processo de troca de mensagens, este módulo enviará um aviso de finalização para o Agente e se auto encerrará. Os estados assumidos por este módulo está representado por um Diagrama de Estado na Figura 3.9.

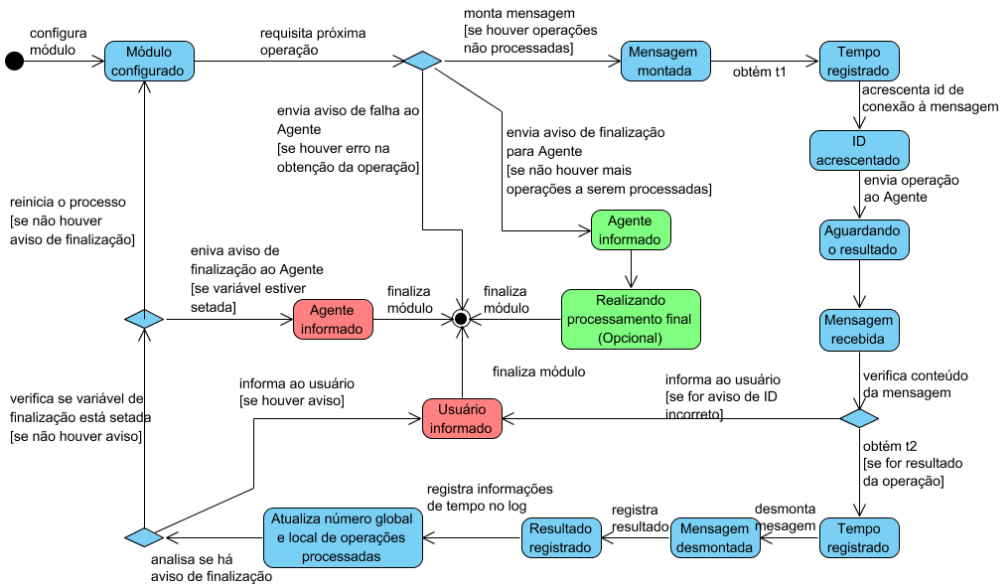


Figura 3.9. Diagrama de Estado do módulo clTrocaMsg

clConfiguracao

Neste módulo, assim como em todos os demais módulos que são adaptáveis, são sugeridas funcionalidades genéricas, necessárias para distribuir uma demanda de processamento do tipo *Bag-of-Tasks*. Porém, por serem adaptáveis, novas funcionalidades poderão ser acrescentadas, dependendo da solução específica à qual a metodologia será adaptada.

Este módulo possui configurações que serão importantes nas diversas etapas de execução do Coordenador. Estas configurações são informações estáveis e por esta razão devem ser armazenadas em um arquivo externo. Este arquivo pode ser do tipo texto, arquivo binário, arquivo XML, uma tabela de um banco de dados, etc. A forma como as configurações serão armazenadas ficam a cargo de cada solução específica. Este módulo terá duas tarefas básicas, uma é acessar o arquivo contendo as configurações e carregá-las, a outra tarefa é proporcionar meios para que alguns módulos possam acessar tais configurações. As configurações genéricas sugeridas pelo modelo estão listadas a seguir.

- Número máximo de conexões secundárias: esta é uma configuração para limitar o número de Agentes que podem ser atendidos pelo Coordenador. Este limite não permite que o Coordenador se sobrecarregue. Quando o Coordenador atingir o número máximo de conexões secundárias, ele passa a informar ao Agente que tentam estabelecer a conexão primária que não há *slots* disponíveis para atendê-lo. O número máximo de conexões secundárias irá variar de acordo com algumas características, que são dependentes do tipo da aplicação que será desenvolvida.
- *Timeout* para recebimento de resultados: esta configuração é necessária para evitar que a demora apresentada por um voluntário em enviar o resultado não seja entendido como uma interrupção na comunicação. A definição deste *timeout* é dependente da solução que será desenvolvida, já que cada tipo de operação terá características diferentes e tempos de processamento diferentes.
- *Buffer* para troca de mensagens: para evitar que seja necessário executar a função da API *socket* responsável por enviar mensagens mais de uma vez para enviar apenas uma mensagem, o *buffer* para envio de mensagens poderá ser configurado de acordo com cada solução para que seja necessária apenas uma execução da função de envio de mensagem. Em casos onde a API *Socket* não é utilizada, esta configuração terá o mesmo significado, mas será implementada de outra forma. No caso da utilização de *middlewares* de comunicação como CORBA e Java RMI, esta configuração será implementada de acordo com as características de cada *middleware*.
- Porta utilizada na conexão primária: como a porta para estabelecimento da primeira conexão com o Coordenador deve ser fixa e de conhecimento de todas as aplicações Agentes, esta porta é definida no arquivo de configurações.
- Fila para estabelecimento da conexão primária: como a conexão primária é estabelecida através de uma única porta e cada Agente é atendido seqüencialmente, poderá se formar filas de pedidos de conexões de Agentes. Esta configuração específica qual o tamanho máximo para estas filas (Estas filas são gerenciada pela própria API *Socket*). Os pedidos que excederem o tamanho da fila, não serão aceitos.
- Tamanho bloco de operações: no modelo foi especificado que uma operação poderia representar, ao invés de uma única operação, um bloco de operações.

Esta configuração especifica o tamanho desde bloco, ou seja, o número de sub-operações que formará o bloco.

clSeguranca

Antes de descrever as características deste módulo, será descrito em quais situações é necessário algum nível de segurança. A primeira necessidade de segurança é na criptografia de alguns dados contidos no arquivo com informações dos voluntários. Este arquivo será detalhado na sessão referente ao módulo *clAutenticacao*, no entanto, ele contém basicamente o nome e senha de cada voluntário. Esta senha precisa ser criptografada para evitar acesso indevido. Outra necessidade é no momento da autenticação dos voluntários, quando a senha dos voluntários irá trafegar na rede. Nesta situação há a necessidade de criptografia das mensagens de autenticação. A terceira necessidade de segurança é na geração e verificação do ID de conexão contido nas mensagens. Esta última funcionalidade é a única desempenhada pelo próprio módulo *clSeguranca*. As demais funcionalidades, como autenticação dos voluntários e criptografia de dados, serão desempenhadas por outros módulos. Ou seja, este módulo será composto de outros módulos (*clAutenticacao* e *clCripto*) que irão oferecer as funcionalidades necessárias para implementar o nível desejado de segurança ao sistema.

Este módulo servirá de intermediário entre o restante do sistema e os módulos *clAutenticacao* e *clCripto*, além de gerar e realizar a verificação do ID de conexão das mensagens recebidas. O ID de conexão será gerado logo após a autenticação de um Agente. A partir deste momento, todas as mensagens trocadas entre o Agente e o Coordenador deverão conter o ID de conexão e ambas as aplicações só aceitarão mensagens com ID válido. Este mecanismo evita que terceiros enviem mensagens para o Coordenador se fazendo passar por um Agente. Deve-se salientar o não uso de criptografia nas mensagens trocadas entre Agentes e Coordenador. Este mecanismo de segurança não é eficiente neste tipo de distribuição de processamento e os motivos para tal afirmação serão apresentados na sessão referente ao módulo *clAcessoOperacoes*, que descreve um outro mecanismo, que não a criptografia, para proporcionar confiabilidade aos resultados recebidos dos Agentes. Basicamente, a criptografia não é utilizada, pois não há confiança entre as partes. Desta forma, proporcionar confiabilidade, integridade ou autenticidade das mensagens não evita que um voluntário envie resultados incorretos, conscientemente ou não.

clCripto

No Diagrama de Classe apresentada no anexo A.1, foram especificados seis métodos para o módulo *clCripto*: (a) métodos para codificar e decodificar dados armazenados no arquivo com informações dos voluntários; (b) método para codificar e decodificar mensagens trocadas na autenticação do Agente e (c) métodos através das quais serão especificadas as chaves utilizadas em ambas as criptografia. Porém, estas propostas de métodos, podem ser substituída por qualquer classe ou algoritmo de criptografia, que consequentemente irão possuir métodos próprios. Os métodos propostos e apresentados no diagrama são sugeridos, já que este módulo é adaptável e certamente será modelado a cada solução que utilizar esta metodologia.

clAutenticacao

De acordo com as características descritas nos dois módulos anteriores é possível notar que a autenticação dos voluntários é baseada em um nome e senha, que serão cadastrados previamente. O que não descarta outras formas de autenticação, como a metodologia utilizada no projeto Globus (http://www.globus.org/), baseada em certificados. Com a autenticação baseada em nome e senha, é necessária uma forma de armazenar as informações dos voluntários. Elas podem ser armazenadas em arquivos modo texto, banco de dados, arquivo XML, etc. A melhor forma a ser utilizada irá depender das características de cada solução. Então se torna necessário que este módulo tenha acesso a este arquivo, que é representado através do atributo “arquivo”, contido no Diagrama de Classe do Anexo A.1. A tarefa primordial deste módulo é receber um nome e senha e realizar a busca por estas informações no arquivo para autenticar um determinado voluntário. Como resultado desta tarefa, o módulo iria apenas indicar se as informações estão de acordo com algum voluntário ou não.

A título de esclarecimento, é conveniente destacar a diferença entre a tarefa de autenticação desempenhada pelo módulo *clConexaoSecundaria* e a tarefa desempenhada por este módulo. A tarefa executada pelo primeiro módulo está relacionada com a comunicação necessária entre Coordenador e Agente para que a autenticação seja realizada. Esta comunicação envolve o envio do pedido de nome e senha, o recebimento desta informação e o aviso confirmando ou negando a autenticação. No caso do número de tentativas exceder o número máximo, o módulo *clConexaoSecundaria* irá enviar uma mensagem de finalização de conexão. Ou seja, a tarefa de autenticação do módulo *clConexaoSecundaria* só trabalha troca de mensagens

com o Agente. O módulo *clAutenticacao* realiza a autenticação de um voluntário do ponto de vista do acesso às informações dos voluntários e validação de um nome e senha recebidos o módulo *clConexaoSecundaria*.

clAcessoOperacoes

O objetivo deste módulo é oferecer uma solução para um dos principais desafios da área de computação pública. Este desafio, descrito na sessão 2.4.2, está relacionado com a falta de confiança entre o servidor (Coordenador) e os clientes (Agentes). Aplicações clientes poderão enviar resultados incorretos, conscientemente ou não. Além disto, um determinado resultado incorreto, recebido pelo servidor, poderia ter sido enviado por um terceiro participante. Como na computação voluntária não existe um conhecimento preciso sobre a identidade dos clientes, pois estes clientes possuem informações básicas cadastradas tais como nome, senha, e-mail, etc., não há como existir confiabilidade entre servidor e clientes.

Uma solução implementada no projeto SETI@Home e descrita em [Anderson, et al., 2002] é a redundância de processamento. Ou seja, a mesma operação é enviada para mais de um cliente para que seja possível identificar os clientes que enviam resultados incorretos ou resultados incorretos enviados por terceiros. Para eliminar a segunda ameaça, as mensagens trocadas entre servidor e clientes poderiam ser criptografadas, para garantir autenticidade e integridade. Porém a criptografia as mensagens não elimina a primeira ameaça, onde o cliente envia, intencionalmente ou não, resultados incorretos. Em resumo, a criptografia só é eficiente onde existe confiabilidade entre as partes envolvidas na comunicação, que não é o caso da computação voluntária. Por esta razão a redundância de processamento se mostra eficiente na validação dos resultados recebidos.

A política adotada pelo projeto SETI@Home é de enviar a mesma operação para três clientes aleatórios. Ao receber os resultados dos três clientes, estes são comparados. Se um dos clientes tiver enviado um resultado incorreto, este será diferente do demais. Desta forma o servidor irá assumir que os outros dois resultados, que são iguais, correspondem a um resultado correto. Nesta política, caso um cliente envie um resultado incorreto, além deste resultado ser detectado, não há necessidade de reenviar a mesma operação a outros clientes para obter um resultado correto. Porém existe a desvantagem de reduzir em 1/3 a utilização do poder computacional disponível.

A solução proposta neste trabalho, é que a mesma operação seja enviada para dois Agentes distintos, ao invés de três. Ao receber os resultados dos dois Agentes para os quais duas cópias de uma mesma operação foram enviadas, estes resultados serão comparados. Se forem iguais, estes resultados são assumidos como resultados corretos. Se forem diferentes, ambos os resultado são classificados como incorretos e outras duas cópias são enviadas para mais dois Agentes aleatórios. O processo de comparação dos resultados se repete e caso estes resultados também sejam diferentes, o processo de envio da mesma operação para mais dois Agentes se repete até que se obtenha resultados iguais.

Esta política poderá parecer mais ineficiente que a política adotada pelo projeto SETI@Home, pois em caso de resultados incorretos, as operações devem ser reenviadas para mais dois Agentes. Enquanto no caso de resultados incorretos quando se está utilizando a política de três réplicas, em geral é possível obter um resultado correto mesmo na presença de um resultado incorreto dentre os três resultados recebidos sem a necessidade de reenvio de operações. Porém, o recebimento de um resultado incorreto é exceção no processo de distribuição utilizando computação voluntária. Apesar de não haver um trabalho publicado sobre a porcentagem de resultados incorretos recebidos por servidores de distribuição de processamento, acredita-se que esta situação é uma exceção, ao invés de regra. Caso contrário, a distribuição de processamento utilizando computação voluntária se mostraria inviável do ponto de vista da confiabilidade entre servidor e voluntários. Se tomarmos esta situação como exceção (suponhamos em 10% dos casos), a política de duas réplicas terá um desempenho melhor em 90% dos casos, já que o poder computacional disponível seria 45% do universo disponível e não 33%..

Este módulo servirá de intermediário entre o módulo *clTrocaMsg* e o módulo *clAcessoDados*. Ao receber uma requisição de operação, do módulo *clTrocaMsg*, ele irá requisitar uma operação do módulo *clAcessoDados* para que esta operação possa ser enviada ao *clTrocaMsg*. Porém a função principal deste módulo é garantir que réplicas de operações sejam processadas por dois Agentes distintos de forma transparente, sem que o módulo *clTrocaMsg* tenha conhecimento deste fato. Para cada operação que é obtida do módulo *clAcessoDados* é criada uma sessão de processamento de réplicas. Cada sessão irá conter duas réplicas de uma mesma operação, que serão enviadas para dois módulos *clTrocaMsg* distintos, as características da sessão de processamento de réplicas são apresentadas na descrição do módulo *clSessao*. Ou seja, enquanto as duas réplicas de uma sessão não forem enviadas para módulos *clTrocaMsg*, uma nova

operação não será requisitada ao módulo *clAcessoDados*. Este módulo poderá possuir diversas seções de processamento de réplicas, que serão criadas dinamicamente à medida que os módulos *clTrocaMsg* irão requisitar operações. As três funções principais deste módulo são:

- Obter uma operação, de uma sessão já criada ou do módulo *clAcessoDados* caso não haja nenhuma sessão já criada ou todas as seções criadas já tiverem suas réplicas de operações já enviadas para módulos *clTrocaMsg*. Em seguida enviar esta operação para o módulo *clTrocaMsg* que a requisitou.
- Registrar um resultado recebido de um módulo *clTrocaMsg* na sua respectiva sessão. Este módulo possui uma tabela onde irá mapear, baseado no ID, qual operação pertence a qual sessão, para viabilizar o registro dos resultados recebidos nas respectivas seções. Caso o resultado recebido seja o segundo resultado de uma determinada sessão, significa que esta sessão já poderá executar o processo de validação dos resultados. Caso os resultados sejam validados, a operação é rotulada como processada pelo módulo *clAcessoOperacoes* e este resultado é armazenado no arquivo de operações. Caso os resultados não sejam validados, a sessão é indificada como "em aberto" e suas duas réplicas serão enviadas a módulos *clTrocaMsg* para serem processadas novamente.
- Registrar falha do processamento. Quando houver algum tipo de falha na comunicação entre Coordenador e Agente, o módulo *clTrocaMsg* irá reportar erro ao módulo *clAcessoOperacoes*. Em caso de erro, este módulo irá informar à respectiva sessão que a réplica da operação não foi processada. Desta forma, está replica será enviada na próxima requisição de operação.

Além das funções principais existem duas funções cujas funcionalidades serão implementadas no módulo *clAcessoDados*, são a função para atualizar o índice denominado "Etapa Completa" e a função para correção do arquivo de operações. Como os módulos *clTrocaMsg* e *clConexaoPrimaria*, que irão executar tais funções, não têm acesso direto ao módulo *clAcessoDados*, serão executadas as funções deste módulo que repassará a chamada para o módulo *clAcessoDados*. As três funções deste módulo estão representadas por Diagramas de Estado, ilustrados na Figura 3.10 e Figura 3.11.

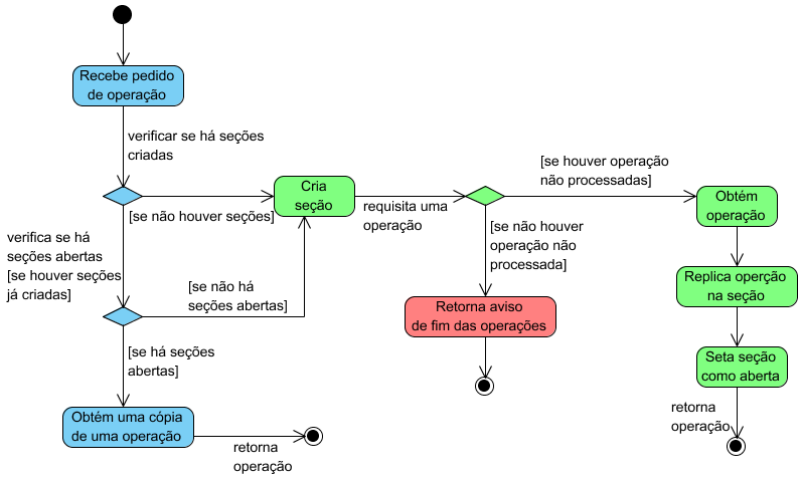
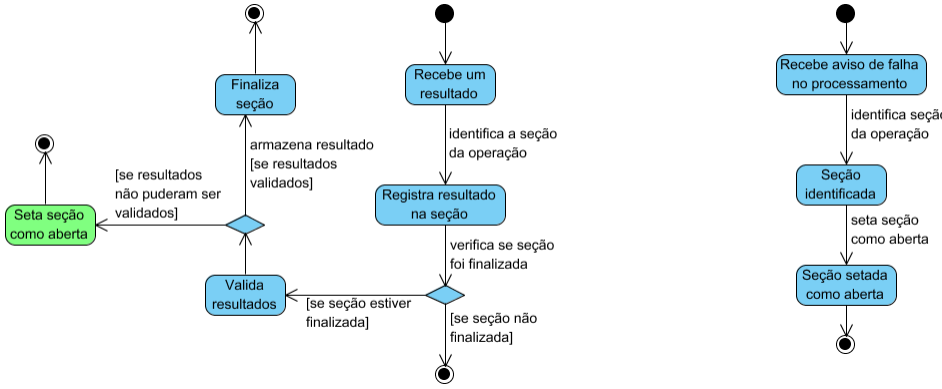


Figura 3.10. Diagrama de Estado que representa o tratamento de um pedido de operação pelo módulo clAcessoOperacoes



(a) (b)

Figura 3.11. (a) Diagrama de Estado que representa o tratamento do recebimento de um resultado pelo módulo clAcessoOperacoes (b) Diagrama de Estado que representa o tratamento do aviso de falha no processamento de uma operação pelo módulo clAcessoOperacoes

clSessao

Este módulo é um complemento para o módulo *clAcessoOperacoes*. Como foi definido que seria utilizado o processamento redundante para garantir confiabilidade dos resultados, o processamento das operações foi dividido em seções e este módulo representa as seções de processamento de réplicas. Cada módulo deste, agregado ao módulo *clAcessoOperacoes*, representará uma sessão de processamento em que uma operação será processada duas vezes.

Desta forma, o módulo *clSessao* é composto por dois módulos *clOperacao*, que na verdade serão réplicas de uma mesma operação. Cada módulo poderá estar em dois estados distintos: (a) em aberto: quando há pelo menos uma operação no módulo que

ainda não foi processada e (b) fechado: quando todas as operações do módulo já foram processadas. Este módulo possui três funções básicas:

- Cria sessão: criar duas réplicas a partir o módulo *clOperacao* recebido no módulo *clAcessoOperacoes* e identificar a sessão como "em aberto" já que ambas as operações ainda não foram processadas.
- Enviar operação: quando uma operação é requisitada pelo módulo *clAcessoOperacoes*, este módulo realiza uma verificação para saber se existe pelo menos uma operação ainda não processada (sessão em aberto), em caso positivo, uma das réplicas é retornada ao módulo que requisitou a operação, caso negativo um aviso de "sessão finalizada" é enviado ao mesmo módulo.
- Valida sessão: quando ambas as operações tiverem sido processadas, é necessário validar os resultados. Os resultados são validados comparando-os, se os dois resultados forem iguais, então estes são válidos e o resultado é gravado no arquivo de operações através do módulo *clAcessoDados*. Caso os resultados não sejam iguais, a sessão é idenficada como "em aberto" para que as mesmas réplicas sejam processadas por outros Agentes para que a validação seja realizada novamente. Este processo irá se repetir até ambos os resultados sejam validados. O Diagrama de Estado, ilustrado na Figura 3.12, descreve os estados assumidos pelo módulo *clSessao*.

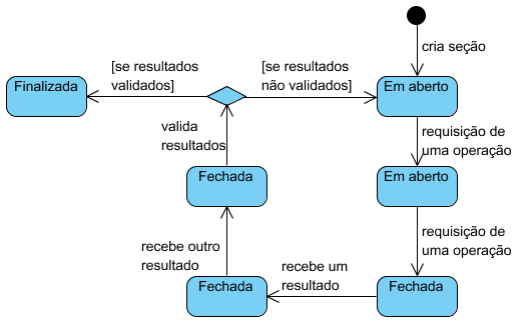


Figura 3.12. Diagrama de Estado que representa os estados assumidos pelo módulo clSessao.

clAcessoDados

A principal função desde módulo é permitir o acesso às operações, que estarão armazenadas em um determinado formato. Uma operação é uma representação de alguma tarefa, ou alguma informação, que deverá ser processada. Estas operações poderão estar armazenadas em diversos formatos como XML, arquivos modo texto,

arquivos binários, banco de dados, etc. É função desde módulo isolar a representação dos dados para que a implementação dos demais módulos seja independente da forma que as operações estão armazenadas. O único módulo que conhece a formatação e a forma como acessar as operações é o módulo *clAcessoDados*. Este módulo deverá possui algumas informações básicas, comentadas a seguir:

- Arquivo: representa uma meio através do qual o arquivo de operações poderá ser acessado, independente de sua representação (arquivo modo texto, XML, banco de dados, etc.)
- Etapa Completa: Como as operações a serem processadas não guardam nenhuma relação entre elas e não possuem inter-dependência, de acordo com a definição de problemas do tipo *Bag-of-Tasks*, uma estrutura de dados adequada para representá-las seria uma lista. Independente do formado de armazenamento, estas operações poderiam ser organizadas na forma de lista. Porém um ponto negativo na representação na forma de lista seria uma provável busca linear por operações não processadas. Com um número elevado de operação, a busca linear se mostraria ineficiente do ponto de vista do tempo de processamento para se obter a próxima operação. Este fato poderia fazer que o Coordenador se torne o gargalho do sistema rapidamente. Caso a estrutura de lista seja adotada na organização das operações, um índice que indicasse até que ponto na lista todas as operações já foram processadas, seria vital para evitar uma busca linear. Por esta razão, este atributo é importante por indicar a posição em um determinado arquivo onde o módulo *clAcessoDados* poderia começar sua busca por operações ainda não processadas. Com a implementação de um mecanismo (implementado no módulo *clTrocaMsg*) para atualizar este índice periodicamente, se evitaria a realização de buscas lineares que comprometesse o desempenho do Coordenador. Neste mecanismo, este índice é atualizado sempre que a troca de mensagens entre o Coordenador e um Agente é encerrada. Dada a volatilidade que os voluntários estabelecem e finalização conexão com o Coordenador, este índice, que é armazenado junto com o arquivo de operações, estará sempre atualizado.
- *Buffer*: onde ficará a informação que representa uma operação, obtida do arquivo de dados. Este módulo possui o conhecimento sobre como acessar o arquivo de operações e obter as informações que representam uma operação. Ele não conhece a formatação interna de cada operação, o significado e organização das

informações que compõe uma operação. Apenas os módulos *clOp* e *clOperacao* conhecem a formatação interna que uma operação é armazenada. Por esta razão, a informação contida neste atributo é passada para o módulo *clOperacao* para que este possa ser configurado de acordo com as informações contidas no *Buffer*.

- Mutex Arquivo Operações: usado para implementar exclusão mútua no arquivo de operações já que diversos módulos *clTrocaMsg* poderão invocar o pedido de uma nova operação em paralelo.

Este módulo poderá receber do módulo *clAcessoOperacoes*, que é o módulo que o acessa algumas informações de configuração como nome do arquivo que contém as operações ou nome da tabela de um banco de dados, por exemplo. De forma geral as informações a serem passadas para este módulo ficam a cargo de cada solução que utilizará este modelo, já que este é um módulo adaptável. Outra função é a obtenção do índice referente à "Etapa Completa" para evitar longas buscas lineares por operações. Com base neste índice é possível realizar a busca pela próxima operação de forma eficiente.

Uma operação armazenada no arquivo de operações poderá ter três estados distintos: (a) não processada: quando não enviada para nenhum Agente até o momento; (b) em processamento: quando enviada para um Agente cujo resultado ainda não foi recebido pelo Coordenador e (c) processada: quando enviada para um Agente e o resultado recebido pelo Coordenador já foi registrado no arquivo de operações. Com base neste três estados é possível notar que uma operação identificada como "em processamento" poderá voltar ao estado "não processado" caso ocorra alguma falha no processo de comunicação entre Coordenador e Agente. Por esta razão, é função deste módulo, além de registrar o resultado de um processamento bem sucedido, recuperar o estado anterior de uma operação quando ocorrer algum erro no processo de comunicação.

Outra função deste módulo é a atualização do índice "Etapa Completa". Para isto, deve ser verificado quantas operações já foram processadas na lista de operações a partir do índice "Etapa Completa" atual. É realizada uma contagem do número de operações processadas ininterruptamente, para que o índice seja atualizado. Uma importante função deste módulo é manter a consistência do arquivo de operações. Uma consistência básica sugerida neste modelo é evitar que uma operação que foi

identificada como “em processamento”, que é um estado passageiro (ao contrário dos outros dois estados), permaneça neste estado mesmo depois do Coordenador ter sido finalizado. Esta situação poderá ocorrer em algumas ocasiões, uma delas é um encerramento não planejado da aplicação Coordenador, por falta de energia ou qualquer outro motivo. Nesta situação todas as operações que estiverem identificadas como “em processamento” não poderão ser identificadas para outros dos dois estados. Neste caso, sempre que o Coordenador for iniciar sua execução, ele irá requisitar do módulo *clAcessoOperacoes* que a consistência do arquivo de operações seja verificada. Desta forma, a função básica deste método será verifica todas as operações identificadas como “em processamento” e identificá-las como “não processadas” para que o processamento destas seja realizado novamente.

Caso uma determinada solução possua como característica a necessidade da realização de um processamento final, após todas as operações terem sido processadas, será responsabilidade do módulo *clAcessoDados* registrar as informações referentes ao processamento final em um arquivo, que pode ser separado do arquivo de operações ou não. Informações mais detalhadas sobre o processamento final serão descrito na sessão referente ao módulo *clProcessamentoFinal*.

clOperacao

Este módulo pode ter duas configurações distintas. Em uma configuração, ele irá representar uma única operação. Nesta configuração não há informações genéricas o bastante que possam ser descritas nesta metodologia. O conceito de operação é tão dependente de uma solução específica que torna inviável criar atributos padrão para todos os possíveis tipos de operações. Por esta razão, no caso em que este módulo represente uma única operação não há como definir nem atributos nem funcionalidades básicas para este módulo.

Na segunda configuração possível, este módulo irá representar, ao invés de uma única operação, um conjunto de operações. Esta característica é importante em casos onde o tempo de processamento de uma única operação não é tão grande em relação ao tempo gasto para a transmissão desta operação para o Agente. Não existe tempo definido na literatura, relacionada à distribuição de processamento, para uma informação a ser processada. Porém é evidente que Agente que recebe informações cujo tempo de processamento é curto, irá realizar várias requisições de novas operações ao Coordenador em um curto intervalo de tempo, o que poderia sobrecarregar o

Coordenador. Possivelmente um único Agente realizando várias requisições de operações ao Coordenador não irá sobrecarregá-lo, mas quando o número de Agentes realizando várias requisições aumentar, o Coordenador poderá se tornar o gargalo do sistema à medida que terá que despachar uma alta taxa de operações num curto intervalo de tempo. Por esta razão, se uma operação levar pouco tempo para ser processada pelos Agentes, é interessante, ao invés de enviar uma única operação, enviar um conjunto de operações para cada Agente.

Desta forma, o módulo *clOperacao* passará a representar um conjunto de operações e este conjunto contém informações e funcionalidade que serão comuns a diversos tipos de soluções e consequentemente de operações. Neste caso, este módulo conteria um conjunto de módulos básicos denominados *clOp* que representariam uma única operação. A princípio, o módulo *clOperacao* conteria um número fixo de módulos básicos que estaria definido no arquivo de configurações e que seria passado para este módulo.

Uma das funções deste módulo seria receber uma operação por vez do módulo *clAcessoDados*, à medida que este módulo obtivesse as operações do arquivo de dados. As operações seriam acrescentadas até que um bloco de operações fosse completado. O módulo *clAcessoDados* também teria conhecimento do número de operações que formaria um bloco e enviaria a quantidade certa para o módulo *clOperacao*. Já que este módulo é um dos poucos a conhecer a formatação interna de uma operação (o módulo *clOp* também conhece a formatação e na verdade define esta formatação), ele seria o responsável por montar mensagens, possivelmente mensagens modo texto, que seriam enviadas aos Agentes. Assim que o Coordenador recebesse uma mensagem contendo o resultado do processamento, este módulo seria requisitado a desmontar esta mensagem para obter dela os resultados enviados por um Agente. Deve-se ressaltar que o módulo *clTrocaMsg* não conhece o conteúdo nem formatação das mensagens enviadas e recebidas dos Agentes. Como última função, este módulo seria responsável por enviar ao módulo *clAcessoDados* operações (com seus respectivos resultados) individuais quando este módulo requisitasse as operações para que os resultados sejam gravados no arquivo de operações.

clLog

Como este módulo é da classe dos módulos adaptáveis, é proposto o registro de informações genéricas e que são relevantes, independente do tipo da solução específica.

As informações registradas são relacionadas ao tempo médio de processamento, número de operações processadas por usuário, etc. Seguindo as mesmas especificações de outros módulos com acesso a arquivos, este módulo poderá ser desenvolvido para acesso a arquivos com diferentes formatações como arquivos modo texto, arquivos binários, XML, banco de dados, etc. Todas as informações registradas estão listadas a seguir:

- Data e hora do registro do log – Cada registro realizado no log corresponde às informações relativas à participação de um voluntário no processamento de operações. Durante o período em que o voluntário permanecer conectado ao Coordenador vão sendo colhidas algumas informações que serão registradas ao final de sua participação. A data e hora registradas correspondem à data e hora do final da participação de um determinado Agente na sessão de processamento.
- ID do voluntário.
- Número de Mensagens enviadas – representa o número de mensagens que o Coordenador enviou ao determinado Agente.
- Número de Operações Processadas – No caso em que o módulo *clOperacao* represente um bloco de operações, ao invés de uma única operação, o número de operações processadas será relativo ao número de sub-operações de um bloco de operação, e não diretamente relativo ao número de mensagens enviadas. Neste caso o número total de operações processadas será o número de operações em um bloco multiplicado pelo número de mensagens transmitidas.
- Tempo Médio de Processamento por Mensagem Enviada – A cada mensagem que é enviada para um Agente, o módulo *clTrocaMsg* registra o tempo entre o envio da mensagem (operação) e o recebimento do resultado. Esta informação é enviada para o módulo *clLog* juntamente com a informação de tempo de processamento (enviada pelo Agente). O módulo *clLog* vai acumulando estes valores para que ao final da participação do voluntário, os tempos médios de transmissão de mensagens e de processamento possam ser registrados em log.
- Tempo Médio de Transmissão de Mensagens – Esta é outra informação que derivada do acúmulo dos tempos de transmissão de todas as mensagens enviadas e dos resultados recebidos. Ao final da conexão, o módulo *clLog* calcula o tempo médio.
- Tempo Total – é o tempo total em que o voluntário permaneceu conectado ao Coordenador.

Com base nestas informações básicas é possível extrair informações mais ricas, como o voluntário que mais contribuiu para o determinado projeto, ou o voluntário cuja aplicação Agente apresentou melhor desempenho do processamento de operações, etc. Diversos projetos das áreas de computação pública criam listas de participantes, com base na quantidade de operações processadas por voluntário.

clProcessamentoFinal

Este e os dois módulo seguintes, são módulos opcionais. Em alguns tipos de problemas estes módulos serão necessários e em outros não. Este módulo representa o processamento realizado sobre as operações, ou seus resultados, depois que todas elas foram processadas. O processamento final poder ser uma análise dos resultados (em busca de um resultado específico, por exemplo), organização dos resultados (como ordenação), ou algum processamento propriamente dito que seria realizado sobre os resultados das operações. No caso específico do estudo de caso realizado neste projeto, o problema cujo processamento foi distribuído, tem como característica a necessidade da realização de um processamento final depois que todas as operações foram processadas. Em outros projetos, para quebrar a criptografia, por exemplo não há necessidade do processamento final já que o interessante é saber se a tentativa, por força bruta, de algum Agente foi bem sucedida em quebrar a criptografia de um bloco de texto. O projeto SETI@Home também não necessita de um processamento final já que as operações enviadas a seus clientes são espectros de freqüências a serem analisadas na busca de algum tipo de sinal gerado artificialmente. Desta forma, não existe um processamento que seja necessário depois que todos os espectros de freqüência foram analisados, os resultados são analisados à medida que são recebidos.

Cada módulo apresentado neste modelo possui funções bem definidas e cada módulo deste não deve ter o conhecimento ou desempenhar as funções de outros módulos. Sobre este ponto de vista, cabe aqui uma discussão sobre qual seria o módulo responsável por invocar o processamento final especificado no módulo *clProcessamentoFinal*. Como o módulo *clTrocaMsg* e *clAcessoOperacoes* terão conhecimento sobre o final das operações, estes módulos seriam os mais indicados para invocar o processamento final. Porém este processamento deve ser invocado apenas uma única vez. Tendo em vista que no momento em que um módulo *clTrocaMsg* "percebesse" o fim as operações, outros módulos *clTrocaMsg* sendo executados em paralelo iriam ter a mesma percepção, o que levaria a mais de uma instanciação do

processamento final. Uma solução seria fazer que o módulo *clConexaoPrimaria*, que é um módulo único e central na arquitetura do Coordenador, invocasse o processamento final. Porém, este módulo não possui recursos de "perceber" o fim do processamento das operações, já que seu papel é apenas estabelecer a conexão primária com os Agentes e criar módulos *clConexaoSecundaria* para atender a cada Agente individualmente.

Desta forma, a solução é fazer que o *clConexaoPrimaria* contenha o módulo *clProcessamentoFinal*. A referência a este módulo será passada a todos os módulos *clConexaoSecundaria* criado, que por sua vez irá passar a referências aos respectivos módulos *clTrocaMsg*. Quando um módulo *clTrocaMsg* for avisado de que as operações a serem processadas acabaram-se, então ele irá invocar o processamento final através da referência ao módulo passado para ele. No módulo de processamento final haverá um controle para que ele execute uma única vez. Depois de executado a primeira vez, este módulo irá "ignorar" a chamada realizadas dos outros módulos *clTrocaMsg* que estarão executando em paralelo. Desta forma garante-se que o processamento final será executado no momento correto e uma única vez.

Como poderá haver diversas chamadas a este módulo simultaneamente, é necessária a implementação de uma região crítica. Por esta razão, este módulo possui um mutex que é travado para verificar se o processamento já foi realizado anteriormente através da análise da variável lógica "processou". Depois de verificar a variável, o módulo saberá se o processamento final deve ser realizado, ou se já foi realizado anteriormente. Caso já tenha sido realizado, o módulo apenas encerra sua execução.

clOp

Neste módulo não há especificações de atributos e funcionalidades, nem mesmo básicas, pois estes características são totalmente dependentes das características de cada problema. O objetivo deste módulo é representar a informação que deve ser processada. Em situações onde o bloco de operações não é utilizado, apenas uma operação básica será enviada aos Agentes, as funcionalidades deste módulo serão desempenhadas pelo módulo *clOperacao*. Por esta razão, este módulo é classificado como opcional, já que ele só será necessário se uma determinada solução utilizar a representação de operações baseada em blocos. Neste caso, um módulo *clOperacao* irá representar um conjunto de operações básicas representadas por módulos *clOp*.

clInterface

A razão para este módulo ser opcional é o fato de sistemas de distribuição de processamento necessitarem pouca interação com o usuário. O cenário mais comum são aplicações com interfaces básicas, possivelmente linha de comando, em que o administrador da aplicação servidor (Coordenador) execute operações básicas como as ilustradas na Figura 3.2. Nesta situação não haveria necessidade de um módulo específico para interface com usuário. Já em soluções que projetam interfaces mais elaboradas com o usuário, seria interessante isolar as funcionalidades relacionadas à interface em um módulo específico. Como as funcionalidades de uma determinada interface são diretamente relacionadas com as características de uma determinada solução, não foram definidas características e funcionalidades para este módulo.

Uma visão geral de todo o processo de distribuição de processamento que o Coordenador segue, com as interações entre módulos é apresentada, através de um Diagrama de Seqüência, no Anexo A.2. Diagramas de Seqüência são utilizados para ilustrar cenários específicos da execução de uma aplicação, assim como a interação entre seus componentes. No diagrama do Anexo A.2 é representado um cenário em que o fluxo principal dos módulos da aplicação Coordenador é executado. Este fluxo principal é representado pelos estados na cor azul nos diversos Diagramas de Estado apresentados nesta sessão.

3.1.2. Agente

Na sessão 3.1.1, tanto a aplicação Coordenador como a aplicação Agente possuem pouca interação com o usuário. Esta característica é devida às características da distribuição de processamento, que geralmente utiliza interação somente entre máquinas, necessitando de pouca ou nenhuma intervenção humana. De acordo com o Diagrama de Caso de Uso da Figura 3.13, é possível notar que há pouca interação entre a aplicação Agente e o voluntário (usuário).

Figura 3.13. Diagrama de Caso de Uso da aplicação Agente.

É possível notar que a principal função da aplicação Agente é ativar-se, o que significa o início de sua execução no processamento de operações. Para que a execução seja iniciada e a conexão com o Coordenador estabelecida, é necessário a realização da autenticação do voluntário, que requer uma segunda interação do voluntário com a aplicação. A partir deste ponto, a aplicação dispensa a intervenção humana para que as operações sejam processadas. Como funções secundárias, é possível obter o número de operações processadas até o momento e finalizar a aplicação. Como a ativação da aplicação é a função primordial do sistema, as etapas envolvidas nesta ativação foram identificadas e listadas a seguir.

- o primeiro passo é configurar a aplicação para que ela possa estabelecer conexão com o Coordenador e em seguida estabelecer, de fato, a conexão;
- ao estabelecer a conexão, a aplicação irá receber um pedido de nome e senha. A aplicação deverá requisitar estas informações do voluntário, que terá um número máximo de tentativas antes que a conexão seja encerrada;
- após a autenticação bem sucedida, o Agente irá receber um ID de conexão usado para validar as mensagens recebidas do Coordenador e concatena-lo junto às mensagens enviadas ao Coordenador para que estas possam ser validadas;
- as próximas duas etapas correspondem a um ciclo em que uma operação é recebida, processada e o resultado é enviado ao Coordenador;

Este ciclo irá se repetir até que uma das partes mostre intenção de encerrar a conexão.De acordo com as tarefas a serem desempenhadas pela aplicação Agente é possível notar que ela é menos complexa que a aplicação Coordenador. Esta maior simplicidade resulta em uma aplicação com menos módulos. Na Figura 3.14 os módulos que compõe a aplicação Agente estão representados através do Diagrama de Classe. O padrão de cores adotado no diagrama da Figura 3.3 é o mesmo padrão adotado aqui, em que os módulos azuis compõem o núcleo do sistema, os módulos verdes são adaptáveis e os módulos amarelos são opcionais. No Anexo A.3 é apresentado o diagrama de classe do Coordenador completo, com os atributos e métodos.

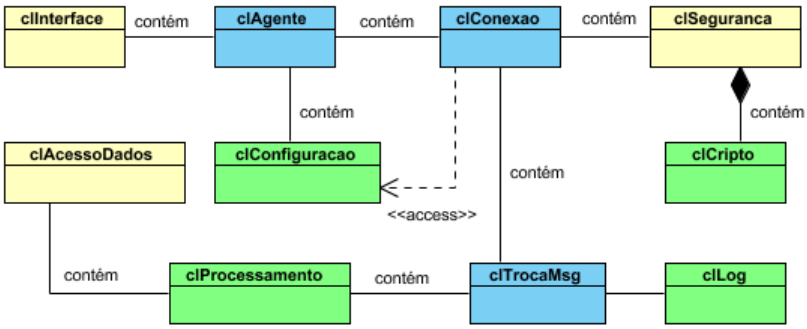


Figura 3.14. Diagrama de Classe da aplicação Agente

clAgente

Assim como o módulo clCoordenador, descrito na sessão 3.1.1, este módulo tem a função básica de fazer intermediação entre as funcionalidades que a aplicação oferece e as necessidades do usuário, ou seja, este módulo desempenha a função de interface funcional entre aplicação e usuário. Ele tem como responsabilidade duas funções básicas. Uma é obter as configurações da aplicação Agente através do módulo *clConfiguracao*. A outra função é iniciar a conexão com o Coordenador através do módulo *clConexao*. Este módulo opera em um nível de abstração alto, sem ter o conhecimento de aspectos mais detalhados da aplicação. Ele só tem conhecimento que deve carregar as configurações necessárias em etapas posteriores da execução da aplicação e iniciar a conexão com o Coordenador passando uma instância do módulo *clConfiguracao* com as configurações para o módulo *clConexao.*

clConexao

Diferente da aplicação Coordenador, as funcionalidades relativas à primeira e segunda conexão não são separadas em módulos distintos. A primeira razão é que na aplicação Coordenador a primeira e a segunda conexão possuem diferentes funcionalidades e arquiteturas. A conexão primária é um ponto central e de controle para os demais módulos da aplicação, enquanto a conexão secundária é criada dinamicamente e executa em paralelo com o módulo *clConexaoPrimaria*. Além disto, na aplicação Agente, as funções a serem desempenhadas são mais simples que no Coordenador, o que dispensa a separação das funcionalidades relativas à primeira e segunda conexão em dois módulos.

Como a aplicação Agente será executada remotamente, ela deve ser resistente a falhas, robusta o bastante para não necessitar de intervenção do voluntário, que certamente não será um usuário com conhecimentos técnicos. Adicionalmente deve-se

evitar que a aplicação Agente tenha execução interrompida em caso de rápidos períodos em que a aplicação Coordenador poderá ficar *offline*, por motivos técnicos ou quando todos os *slots* para conexões secundárias estejam ocupados fazendo que o Coordenador não possa receber mais nenhum pedido de conexão. Por esta razão, este módulo responsável pela conexão primária e secundária com o Coordenador, deve ser persistente na tentativa de estabelecer conexão com o Coordenador. Caso o Coordenador esteja inoperante, o Agente deve continuar tentando até que obtenha sucesso nas tentativas.

A aplicação Coordenadora possui um recurso que impossibilita que ela fique sobrecarregada com diversas conexões secundárias executando em paralelo. Esta funcionalidade foi descrita na sessão 3.1.1, no tópico referente ao módulo *clConexaoPrimaria*. O Coordenador limita o número de conexões secundárias criadas (consequentemente, limita o número de Agentes conectados a ele), fazendo com que alguns Agentes possam não ser atendidos durante um intervalo de tempo. Tendo em vista este comportamento do Coordenador, o módulo *clConexao* possui uma forma de estabelecimento da conexão primária persistente. Ou seja, o módulo executa um ciclo de realização de pedido de conexão até que seu pedido seja atendido. Para não sobrecarregar o Coordenador com pedidos consecutivos, o módulo entra em modo de espera por um tempo até efetuar o próximo pedido de conexão.

Depois de estabelecido a conexão primária com o Coordenador, a primeira mensagens recebida pelo Agente será o pedido de nome e senha. Depois de receber esta mensagem, o Agente saberá que deve iniciar os procedimentos relativos à conexão secundária. A função relacionada com a conexão secundária é a autenticação do voluntário. Para isto, a aplicação irá requisitar do voluntário nome e senha e em seguida enviar estas informações para o Coordenador. Este processo de requisição e envio de nome e senha irá se repetir até que o Agente seja informado que a autenticação foi bem sucedido ou que o número máximo de tentativas de autenticação foi extrapolado. No segundo caso a aplicação Agente será encerrada.

Em casos de autenticação bem sucedida, o Agente receberá o ID de conexão para validar as mensagens recebidas do Coordenador e para acrescenta-lo às mensagens enviadas ao Coordenador para que ele valide tais mensagens. Após o recebimento do ID de conexão, a próxima etapa será o recebimento de operações a serem processadas, o que é tarefa do módulo *clTrocaMsg*. Caso o módulo *clInterface*, que é opcional, não seja utilizado, tendo em vista que a interação com o voluntário seja simplória o bastante

para não necessitar de um módulo específico, após o início da troca de mensagens um método que executa uma linha de comando básico pode ser invocado no próprio módulo *clConexao*.

Para que o módulo *clTrocaMsg* possa receber operações do Coordenador, é passado como parâmetro para o módulo o *socket* que deve ser utilizado no recebimento de operações e envio de resultados, além do ID de conexão para que o módulo valide as mensagens recebidas. Como a execução deste módulo deve ser independente da execução da linha de comando (ou da interface gráfica, caso o módulo *clInterface* seja utilizado) a função "troca_msg()" deste módulo será executada em uma *thread*. Os estados assumidos pelo módulo *clConexao* estão representados pelo Diagrama de Estado ilustrado na Figura 3.15.

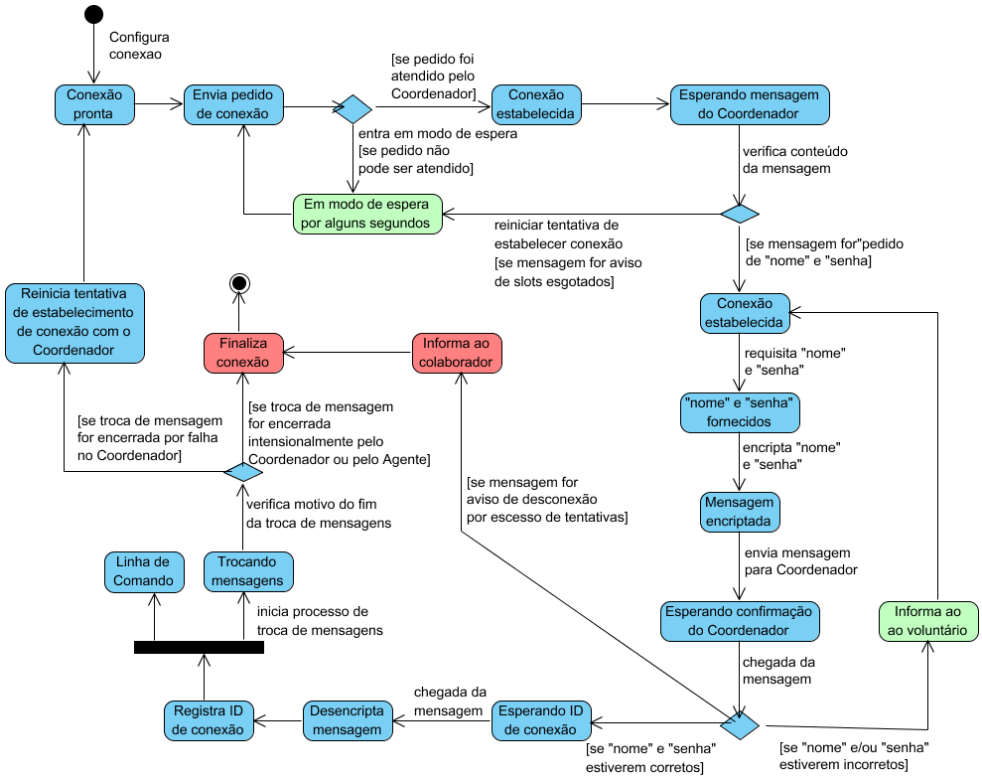


Figura 3.15. Diagrama de Estado do módulo clConexao

clTrocaMsg

Semelhante ao módulo *clTrocaMsg* da aplicação Coordenador, este módulo possui duas funções básicas. Uma é receber uma mensagem contento uma operação a ser processada. A outra é, depois de a operação ter sido processada, enviar o resultado do processamento para o Coordenador. O ciclo constituído por estas duas operações será

executado até que uma das partes mostre intenção de finalizar a conexão ou até que ocorra alguma falha na comunicação entre Agente e Coordenador.

A primeira tarefa deste módulo é receber uma operação a ser processada. Todas as mensagens recebidas passam pela validação do ID de conexão antes que qualquer ação seja realizada. Na validação o ID de conexão contido na mensagem é comparado com o ID de conexão passado a este módulo pelo *clConexao*. Caso a validação não seja bem sucedida, o módulo envia um aviso para o Coordenador e finaliza a conexão. Caso contrário, a mensagem é tida como autentica dando continuidade à execução do módulo. É interessante ressaltar que este módulo poderá receber mensagens contendo avisos além de mensagens com operações. Os tipos de avisos estão listados a seguir. Deve-se ressaltar que em todos os casos de recebimento de algum tipo de aviso a conexão, e consequentemente a aplicação Agente, serão finalizados.

- Fim da Conexão: quando o usuário da aplicação Coordenador mostra intenção de finalizar a aplicação.
- Falha ao obter próxima operação: quando houver algum erro que impossibilite que o Coordenador obtenha a próxima operação que seria enviada ao Agente.
- Fim das operações: quando todas as operações contidas no repositório de operações do Coordenador já foram processadas.
- ID incorreto: quando a mensagem enviada anteriormente pelo Agente contenha um ID de conexão incorreto.

É importante notar que, por definição, o módulo *clTrocaMsg* não deve ter conhecimento sobre a formatação ou o conteúdo das operações recebidas. Isto porque sua função é receber operações e enviar resultado, que na verdade são tratadas como mensagens modo texto. Por esta razão é necessário um módulo que saiba como interpretar as mensagens modo texto contendo operações. Esta tarefa é desempenhada pelo módulo *clProcessamento*. Em caso de mensagem que contenha operação, ao invés de algum aviso, esta mensagem é passada para o módulo *clProcessamento* para que seja configurado com as informações contidas na mensagem.

O próximo passo é registrar o tempo inicial de processamento e invocar o processamento da operação através da chamada ao método do módulo *clProcessamento* responsável pelo processamento de operações. O módulo *clTrocaMsg* permanecerá em modo de espera até que o processamento finalize. Então o tempo final é registrado para que o tempo decorrido do processamento seja obtido. O próprio módulo

clProcessamento é responsável por criar uma mensagem contendo o resultado do processamento para que o módulo *clTrocaMsg* possa enviá-lo ao Coordenador.

Antes que o resultado seja enviado são executadas três tarefas. A primeira é verificar se o módulo *clConexao* enviou aviso de "finalizar aplicação". Caso o aviso tenha sido enviado, o ciclo de recebimento de operações e envio de resultados deve ser interrompido. Neste caso um aviso de finalização de conexão deve ser enviado concatenado à mensagem contendo o resultado do último processamento realizado. Independente da concatenação do aviso de finalização de conexão, outra duas informações devem ser concatenadas à mensagem, uma é o ID de conexão, a outra é o tempo decorrido no último processamento realizado. Em seguida a mensagem é enviada ao Coordenador e o ciclo de troca de mensagens é reiniciado. Este ciclo de troca de mensagens executado por este módulo está representado, através de um Diagrama de Estado, na Figura 3.16.

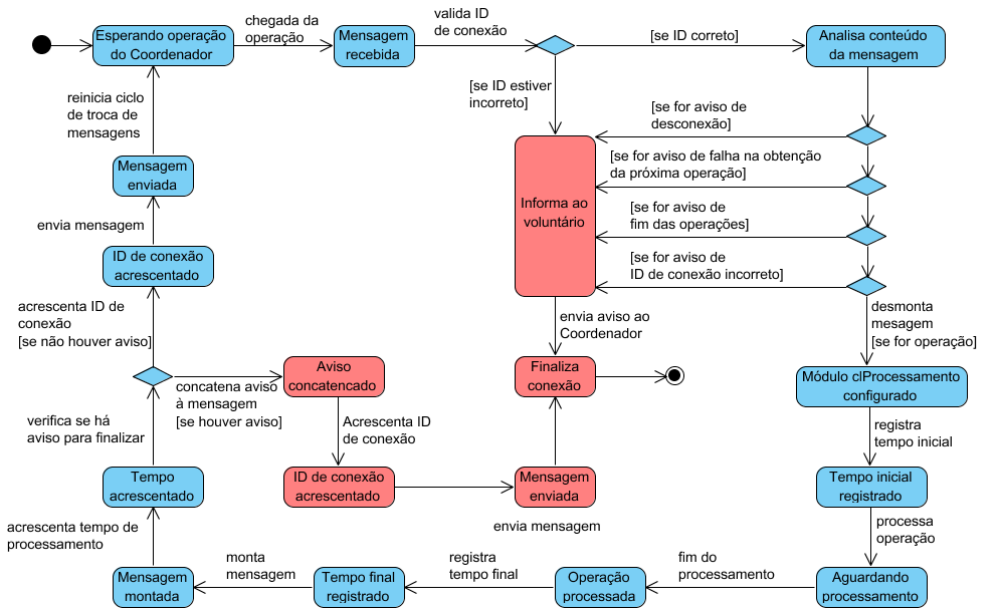


Figura 3.16. Diagrama de Estado do módulo clTrocaMsg

clConfiguracao

Este módulo é responsável por acessar o arquivo que contém as configurações da aplicação Agente. Estas configurações estarão armazenadas em um arquivo que poderá ter as diferentes formatações já apresentadas. É função desde módulo conhecer a forma correta de acessar estas informações e recuperá-las. As configurações relacionadas à aplicação Agente estão listadas a seguir:

- IP: endereço de Internet através do qual o Agente irá se conectar com o Coordenador;
- Porta: O número da porta na qual a aplicação Coordenador estará esperando por pedidos de conexão;
- Timeout Msg: especifica o tempo em que a aplicação Agente deve permanecer esperando para receber uma mensagem. Este parâmetro é importante para que o Agente não assuma, precipitadamente, que o Coordenador esteja impossibilitado de enviar operações. A rede ou o Coordenador podem estar sobrecarregados, implicando em um atraso para chegada da operação;
- *Buffer*: especifica o número máximo de bytes aceitável para as mensagens recebidas do Coordenador.

Caso haja algum erro ao obter as configurações, seja por não conseguir acessar o arquivo ou formatação incorreta deste arquivo, o método responsável pela obtenção das configurações deve informar ao módulo externo e evitar que a aplicação seja executada sem as devidas configurações. Além desta função, o módulo deve oferecer uma forma através da qual outros módulos possam acessar as configurações contidas nele.

clProcessamento

Apesar deste módulo ser totalmente dependente das características de cada tipo de problema, foram definidas funções básicas para ele. A primeira está relacionada com a configuração do processamento. As configurações podem ser algumas informações necessárias para que as operações sejam processadas. Em geral podem ser informações que são passadas ao módulo que irão auxiliá-lo no processamento. A principal função é relacionada ao processamento de uma determinada operação. Esta função é totalmente dependente das características de cada problema. Além destas funções, são propostas as funções para desmontar as operações e montar mensagem. Como sugerido pelo nome, a função de "desmontar operação" consiste em configurar o módulo *clProcessamento* com as operações que são passadas para este módulo na forma de mensagens modo texto que são recebidas do Coordenador pelo módulo clTrocaMsg. A função "montar mensagem" é responsável por criar uma mensagem de texto contendo o resultado do último processamento realizado para o módulo *clTrocaMsg* enviar o resultado para o Coordenador. Adicionalmente é definida uma outra função que informa, sempre que requisitado pelo módulo *clTrocaMsg* o estado em que a etapa de processamento da operação se encontra. As funções pré-definidas na metodologia para os módulos

adaptáveis são genéricas, o que implica na criação de funções específicas para cada módulo adaptável à medida que são necessárias para uma determinada solução.

clCripto

Este módulo será desenvolvido de acordo com o método de criptografia utilizado em cada solução. São definidas três funções básicas para este módulo. Diferente do módulo *clCripto* da aplicação Coordenador, este módulo geralmente não necessitará de métodos para criptografia de informações armazenadas localmente em arquivos. Suas duas principais funções serão codificar a mensagem de autenticação que será enviada ao Coordenador contendo o nome e senha do voluntário e descodificar a mensagem contendo o ID de conexão que o Coordenador irá enviar logo após a autenticação do voluntário. A terceira função é definir uma chave para o método de criptografia. Poderão ser definidas duas chaves distintas, uma para codificar a mensagem de autenticação, a outra para decodificar a mensagem contendo o ID de conexão.

clLog

Para que o voluntário possa ter acesso a informações relacionadas com sua participação no processamento de operações, são registradas algumas informações para consulta. As informações registradas em log estão listadas a seguir:

- Data e hora do registro do log – Durante o período em que o voluntário permanece conectado ao Coordenador vão sendo colhidas algumas informações que serão registradas ao final de sua participação. A data e hora registradas correspondem à data e hora do final da participação de um determinado Agente na sessão de processamento;
- Número de Mensagens enviadas – representa o número de mensagens (resultados) que o Agente enviou ao Coordenador;
- Número de Operações Processadas – No caso em que uma operação recebida do Coordenador represente um bloco de operações, ao invés de uma única operação, o número de operações processadas será relativa ao número de sub-operações de um bloco de operação, e não diretamente relativa ao número de mensagens enviadas. Neste caso o número total de operações processadas será o número de operações em um bloco multiplicado pelo número de mensagens transmitidas;

- Tempo Médio de Processamento por Mensagem Enviada – A cada mensagem que é enviada ao Coordenador, é registrado o tempo decorrido durante o processamento das operações. Estas informações são enviadas ao módulo *clLog* que acumula estes valores para ao final da execução da aplicação registrar o tempo médio de processamento.

clAcessoDados

Este módulo, que é opcional, é necessário em situações onde informações adicionais são necessárias para que a aplicação Agente possa realizar o processamento das operações. Em ocasiões em que a quantidade de informação possa prejudicar a eficiência do sistema caso seja necessário trafegá-la pela rede, estas poderão ser replicadas nos Agentes, caso sejam imutáveis. Neste caso, elas poderiam estar armazenadas em arquivos acessíveis através do módulo *clAcessoDados*, que apesar de ter o mesmo nome do módulos responsável por acessar operações no Coordenador, possui semântica diferente. A função básica para este módulo é configurar a forma de acesso ao arquivo, passando para este módulo algumas informações necessárias para acessar o arquivo de dados. Em problemas como simulações, informações relacionadas a um determinado ambiente base poderiam ser replicadas entre os Agentes e as mensagens de operações conteriam apenas as simulações a serem realizadas sobre o ambiente base, dispensando o envio de toda a informações que poderia comprometer o desempenho do sistema. Neste caso, o módulo *clAcessoDados* seria responsável por recuperar as informações relativas ao ambiente base quando uma simulação for realizada.

clSeguranca

A razão para este módulo ser opcional é que geralmente será necessário apenas a criptografia da mensagem contendo nome e senha, como recurso de segurança. Quando apenas esta criptografia for utilizada, o uso do módulo *clSeguranca* é dispensável. Neste caso o módulo *clConexao* iria acessar o módulo *clCripto* diretamente sem ter que invocar a criptografia e descriptografia de mensagens através do módulo *clSeguranca*. Em caso onde serão necessários outros recursos de segurança, o módulo *clSeguranca* poderá ser utilizado como ponto central dos recursos relacionados à segurança na aplicação e na comunicação com o Coordenador. Neste caso, este módulo será semelhante ao módulo responsável pela segurança na aplicação Coordenador.

clInterface

A razão deste módulo ser opcional é baseada no fato de que distribuição de processamento geralmente possui pouco ou nenhuma necessidade de intervenção ou interação com usuário. Projetos que utilizam computação voluntária geralmente configuram suas aplicações clientes para serem executadas nos períodos em que as máquinas dos voluntários estão ociosas, e assim deve ser para não prejudicar o uso destas máquinas já que estas aplicações consomem muito tempo de processador. Uma política utilizada por alguns projetos é configurar suas aplicações clientes para que sejam executadas quando o descanso de tela é ativado. Outros projetos configuram suas aplicações para executar em *background*. Em ambas as configurações a interação com o usuário é desnecessária, o que torna a utilização de um módulo que implemente uma interface sofisticada dispensável.

Porém uma interface mais elaborada pode ser necessária dependendo da política de interação com o voluntário adotada por um determinado projeto. Um exemplo é o projeto SETI@Home que optou por oferecer uma interface mais elaborada onde o voluntário possa acompanhar o andamento do processamento das informações recebidas. Na Figura 3.17 é apresentada uma tela capturada da aplicação cliente do projeto SETI@Home.

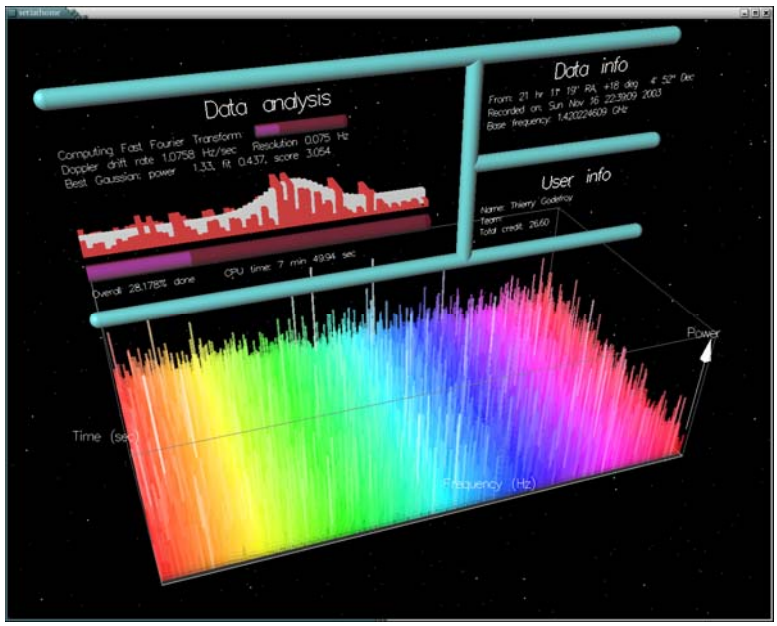


Figura 3.17. Interface da aplicação cliente do projeto SETI@Home.

Uma visão geral de todo o processo de execução que o Agente segue, com as interações entre módulos é apresentado através de um Diagrama de Seqüência no Anexo A.4. No diagrama do Anexo A.4 é representado um cenário em que o fluxo principal dos módulos da aplicação Agente é executado. Este fluxo principal é representado pelos estados na cor azul nos diversos Diagramas de Estado.

3.1.3. Protocolo de Comunicação

Uma característica importante e um desafio para Sistemas Distribuídos, apresentada na sessão 2.1.3, está relacionada com a documentação e com a especificação formal do sistema. Esta documentação possibilita que o sistema seja estendido, adaptado ou que esta formalidade possibilite que outros sistemas possam se comunicar com ele, como no caso da formalização do protocolo de comunicação entre aplicação Agente e Coordenador. Esta formalização possibilita até mesmo aplicações que não foram desenvolvidas seguindo a metodologia proposta, se comunicarem com uma aplicação que utilizou a metodologia. Um exemplo seria uma aplicação Agente, desenvolvida com a ajuda de uma outra ferramenta ou utilizando uma outra metodologia, possa se comunicar com uma aplicação Coordenadora que foi desenvolvida seguindo a metodologia proposta. Para tal, a aplicação Agente deve apenas conhecer o protocolo de comunicação, conhecer a formatação das mensagens e ter conhecimento sobre o processamento que deve ser realizado sobre as operações recebidas. Os estados que devem ser assumidos por uma aplicação Coordenador com relação ao protocolo a ser seguido, estão ilustrados no Anexo A.5, através de um Diagrama de Estado. Assim como os estados a serem assumidos por uma aplicação Agente estão ilustrados no Anexo A.6. Na Figura 3.18 são apresentadas as mensagens trocadas entre Coordenador e Agente em uma situação específica, onde as mensagens trocadas estão relacionadas ao fluxo principal de estados (apresentados no Anexo A.5 e A.6) assumidos por ambas as aplicações.

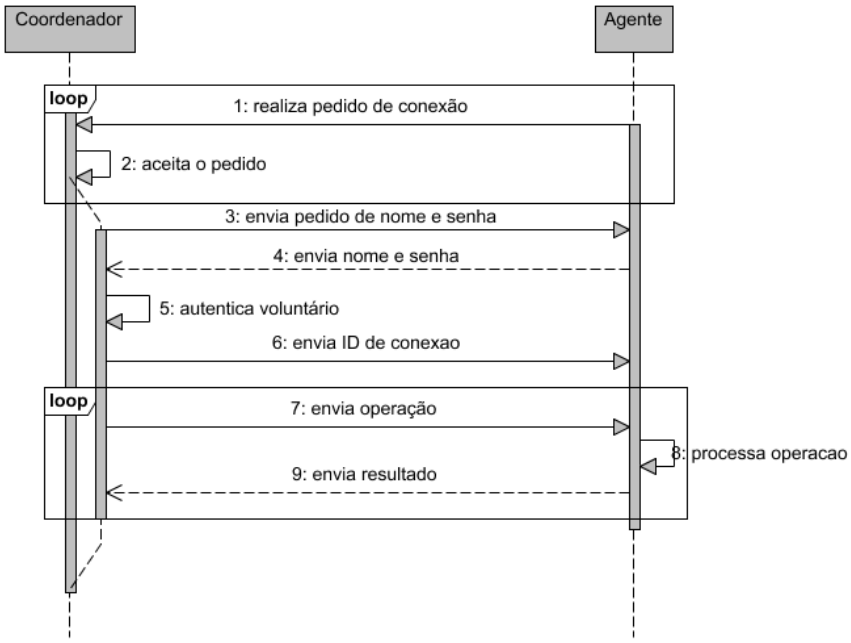


Figura 3.18. Mensagem trocadas entre Coordenador e Agente em uma situação ideal.

4. Estudo de Caso

Para que as especificações descritas na metodologia proposta neste projeto fossem validadas, foi desenvolvido um estudo de caso para uma aplicação de distribuição de processamento pela Internet. Houve uma análise a respeito dos problemas que se encaixavam na definição de problemas do tipo *Bag-of-Tasks* que implicassem em uma contribuição científica. Foram estudados problemas como (1) encontrar o maior (ou significativamente grande) número primo; (2) calcular o valor da constante PI com várias casas decimais; (3) quebra de criptografia, dentre outros. Como nenhum desses problemas era de fato um problema real, optou-se por um caso desafiador, com significativa contribuição científica, porém pouco estudado. Esse problema é a representação funcional de relevo, mais especificamente, a representação de uma superfície por meio de polinômios bidimensionais. O problema de representação de relevo e outras informações matriciais através de polinômio já havia sido estudado anteriormente, porém sem a abordagem na distribuição de processamento [Bajaj et al., 1993], que não permitia o uso de polinômio de alto grau limitava o tamanho da área a ser representada. Quanto maior for a informação a ser representada, maior será o poder computacional exigido. Além disto, uma representação com relativa fidelidade requer um polinômio de alto grau, que também demanda um grande poder computacional. Por esta razão o problema de representação de relevo através de polinômio foi escolhido para o estudo de caso e teste do sistema de distribuição proposto. A representação de informações por uma função polinomial possibilita que algumas funcionalidades matemáticas sejam utilizadas tais como, o cálculo de pontos máximos e mínimos, declividades, aplicação de derivadas e integrais, etc. Adicionalmente o espaço necessário para armazenar os coeficientes do polinômio (em média 8 bytes por coeficiente) é uma fração do espaço ocupado pela matriz de dados representativa do relevo, por exemplo.

4.1. Modelagem Matemática

Nesse trabalho apresentamos uma metodologia para representação de relevo utilizando o método de regressão polinomial bidimensional. A análise de regressão é um instrumento estatístico muito utilizado na ciência em geral. Sua grande utilização se deve ao fato de possibilitar a descrição de fenômenos por meio de modelos matemáticos a partir de uma amostra de dados. Graficamente, equivale a identificar a curva ou

superfície matemática que melhor se ajusta aos pontos de um diagrama de dispersão. Os modelos matemáticos de regressão fundamentam-se em três pressupostos estatísticos: a) a relação entre as variáveis dependente e independente é determinística ao invés de estocástica; b) os erros de medida são aleatórios com distribuição normal, média zero e variância constante; e c) as variáveis explicativas não apresentam correlação entre si [Seber e Wild, 1989].

Quando se utiliza modelos matemáticos de regressão, o método de estimação dos parâmetros mais amplamente utilizado é o método dos mínimos quadrados ordinários que consiste em estimar uma função para representar um conjunto de pontos minimizando o quadrado dos desvios [Nobel e Daniel, 1986]. Considerando um conjunto de coordenadas geográficas (x,y,z), tomando a altitude como função estimadora destes pontos, um polinômio de grau r em x e de grau s em y pode ser dado conforme a Equação (3), com o erro ε_{ij} estimado pela Equação (4), em que $0 \leq i \leq m$ e $0 \leq j \leq n$.

$$\hat{z} = f(x_i, y_j) = \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l \tag{3}$$

$$\varepsilon_{ij} = z_{ij} - \hat{z}_{ij} \tag{4}$$

Os coeficientes a_{kl} (k=0,1,...,r ; l=0,1,...,s) que minimizam o erro da função estimadora f(x,y), podem ser obtidos solucionado a Equação (5) para c = 0, 1, ..., r e d = 0, 1, ..., s.

$$\frac{\partial \xi}{\partial a_{cd}} = 0 \tag{5}$$

onde,

$$\xi = \sum_{i=1}^{m} \sum_{j=1}^{n} \varepsilon_{ij}^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (z_{ij} - \hat{z}_{ji})^2 \tag{6}$$

Da Equação (7) até a Equação (13) tem-se o desenvolvimento da Equação (5).

$$\varepsilon_{ij}^2 = \left(z_{ij} - \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l \right)^2 \tag{7}$$

$$z_{ij} = \sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^k y_j^l + \varepsilon_{ij} \tag{8}$$

$$\xi = \sum_{i=0}^{m}\sum_{j=0}^{n}\left(Z_{ij} - \sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^k y_j^l \right)^2 \tag{9}$$

$$\frac{\partial \xi}{\partial a_{cd}} = 2\sum_{i=0}^{m}\sum_{j=0}^{n}\left[\left(z_{ij} - \sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^k y_j^l \right) x_i^c y_j^d \right] \tag{10}$$

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left[\left(z_{ij} - \sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^k y_j^l \right) x_i^c y_j^d \right] = 0 \tag{11}$$

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^{k+c} y_j^{l+d} = \sum_{i=0}^{m}\sum_{j=0}^{n} z_{ij} x_i^c y_j^d \tag{12}$$

$$\sum_{i=0}^{m}\sum_{j=0}^{n}\left[z_{ij} x_i^c y_j^d - \left(\sum_{k=0}^{r}\sum_{l=0}^{s} a_{kl} x_i^{k+c} y_j^{l+d} \right) \right] = 0 \tag{13}$$

Desenvolvendo a Equação (3) para o caso particular em que r=s=2, é obtido a Equação (14). Desenvolvendo a Equação (13) para o mesmo caso particular, é obtido o sistema de matrizes ilustrado na Figura 4.1, representada por AX = B, onde a matriz A é formada pelos termos x_{lc}, a matriz X é formada pelos termos a_{kl} e a matriz B é formada pelos termos b_l. Os termos x_{lc} e b_l são ilustrados nas Equação (15) e Equação (16), respectivamente como solução geral para qualquer r e qualquer s na Equação (13).

$$\widehat{Z}_{ij}(x_i, y_j) = a_{00}x^0y^0 + a_{01}x^0y^1 + a_{02}x^0y^2 + a_{10}x^1y^0 + a_{11}x^1y^1 + a_{12}x^1y^2 + a_{20}x^2y^0 + a_{21}x^2y^1 + a_{22}x^2y^2 \tag{14}$$

$$\begin{bmatrix}
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 \\
\{0,0\} & \{0,1\} & \{0,2\} & \{0,3\} & \{0,4\} & \{0,5\} & \{0,6\} & \{0,7\} & \{0,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 \\
\{1,0\} & \{1,1\} & \{1,2\} & \{1,3\} & \{1,4\} & \{1,5\} & \{1,6\} & \{1,7\} & \{1,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^0 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^4 \\
\{2,0\} & \{2,1\} & \{2,2\} & \{2,3\} & \{2,4\} & \{2,5\} & \{2,6\} & \{2,7\} & \{2,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 \\
\{3,0\} & \{3,1\} & \{3,2\} & \{3,3\} & \{3,4\} & \{3,5\} & \{3,6\} & \{3,7\} & \{3,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^3 \\
\{4,0\} & \{4,1\} & \{4,2\} & \{4,3\} & \{4,4\} & \{4,5\} & \{4,6\} & \{4,7\} & \{4,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^1 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^4 \\
\{5,0\} & \{5,1\} & \{5,2\} & \{5,3\} & \{5,4\} & \{5,5\} & \{5,6\} & \{5,7\} & \{5,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^0 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^2 \\
\{6,0\} & \{6,1\} & \{6,2\} & \{6,3\} & \{6,4\} & \{6,5\} & \{6,6\} & \{6,7\} & \{6,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^1 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^3 \\
\{7,0\} & \{7,1\} & \{7,2\} & \{7,3\} & \{7,4\} & \{7,5\} & \{7,6\} & \{7,7\} & \{7,8\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^2 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^3 y_j^4 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^2 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^3 & \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^4 y_j^4 \\
\{8,0\} & \{8,1\} & \{8,2\} & \{8,3\} & \{8,4\} & \{8,5\} & \{8,6\} & \{8,7\} & \{8,8\}
\end{bmatrix}
\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{20} \\ a_{21} \\ a_{22} \end{bmatrix} =
\begin{bmatrix}
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^0 y_j^0 \\ \{0,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^0 y_j^1 \\ \{1,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^0 y_j^2 \\ \{2,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^1 y_j^0 \\ \{3,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^1 y_j^1 \\ \{4,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^1 y_j^2 \\ \{5,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^2 y_j^0 \\ \{6,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^2 y_j^1 \\ \{7,0\} \\
\sum_{i=0}^{m}\sum_{j=0}^{n} Z_{ij} x_i^2 y_j^2 \\ \{8,0\}
\end{bmatrix}$$

Figura 4.1. Sistema de matrizes geradas pelo desenvolvimento da Equação (13) considerando o caso em que r = s = 2.

$$x_{lc} = \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^{\,l\,div\,(s+1)+c\,div\,(s+1)}\, y_j^{\,l \bmod (r+1)+c \bmod (r+1)} \tag{15}$$

$$b_l = \sum_{i=0}^{m}\sum_{j=0}^{n} z_{ij}\, x_i^{\,l\,div\,(s+1)}\, y_j^{\,l \bmod (r+1)} \tag{16}$$

A Equação (15) e Equação (16) desenvolvidas deram uma vantagem significativa para a distribuição do processamento para estimativa dos coeficientes do polinômio bidimensional. Estas equações representam genericamente qualquer elemento da matriz A e matriz B. Qualquer termo da matriz A e da matriz B pode ser obtido através das Equação (15) e Equação (16), respectivamente. Na distribuição de processamento o tempo de transmissão deve ser uma fração do tempo total de processamento da informação, caso contrário a distribuição deste processamento se mostra ineficiente. O uso destas equações possibilitou que cada elemento de ambas as matrizes, que são as informações a serem processadas, fosse referenciadas apenas através da linha e coluna onde se encontram, no caso da matriz B, através da linha apenas.

Este fato implica em uma diminuição significativa na quantidade de informações a serem trafegadas na rede. Ao invés de se enviar toda a equação matemática referente a um elemento da matriz, é enviado apenas a posição deste elemento na matriz. De posse das Equação (15) e Equação (16), a aplicação Agente é capaz de deduzir a equação específica a ser processada. Outro fato interessante é que estas equações são válidas para qualquer informação matricial que se queira representar e para qualquer grau de polinômio bidimensional que se queira ajustar.

4.2. Características do Problema

O problema trabalhado neste estudo de caso é o poder computacional exigido para se estimar os coeficientes de um polinômio bidimensional que represente o relevo do estado de Minas Gerais. O relevo do estado de Minas Gerais, área escolhida, é representado por um conjunto de pontos obtido pelo Modelo Digital de Elevação (MDE) do projeto GTOPO30 [GTOPO30] na forma de uma grade regular, com 1343 linhas por 1043 colunas, com espaçamento aproximadamente 900m nas coordenadas geográficas. Como comentando na sessão 2.5.2, regressão polinomial é um tipo de interpolação global e como tal, exige um poder computacional significativo para um conjunto de dados muito grande [Câmara e Felgueiras]. O MDE do estado de Minas Gerais não pode ser considerado um conjunto pequeno, o poder computacional exigido

para cálculo do polinômio para representa-lo com significativa precisão torna inviável o processamento centralizado. As informações relativas ao relevo estão armazenadas em um arquivo modo texto. Para cada medição realizada existe um valor número do arquivo que corresponde à altitude na respectiva posição. As linhas e colunas do arquivo representam a latitude e longitude onde a medição foi realizada. As latitudes e longitudes não pertencentes ao estado de Minas Gerais, isto é, fora dos limites do estado, são representadas pelo valor -9999. Na Figura 4.2 está ilustrada uma parte do arquivo contendo as altitudes do estado.

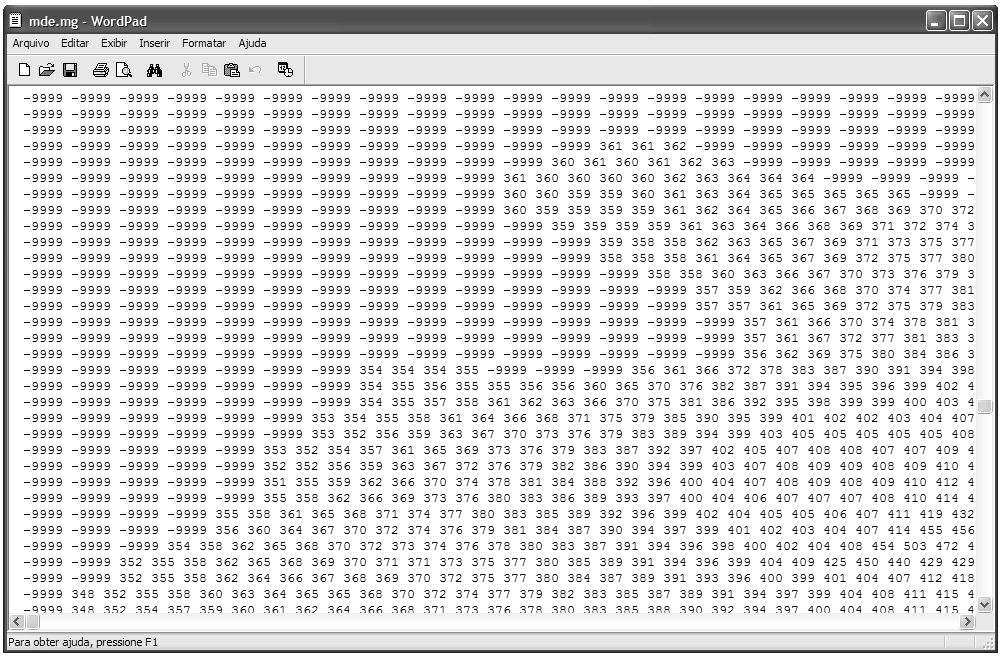


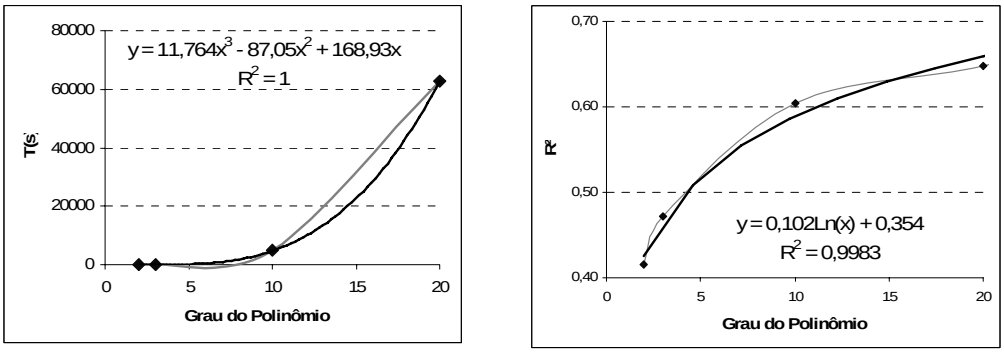
Figura 4.2. Parte do MDE do Estado de Minas Gerais

O MDE, com 1.400.749 pontos, possui apenas 722.294 pontos pertencentes ao estado de Minas Gerais, os demais se situam fora do contorno do estado. A partir dos pontos do estado utilizando-se a Equação (15) e Equação (16), foram estimados, inicialmente, coeficientes para os polinomiais de graus r = s = 2, 3, 10 e 20 utilizando-se uma máquina comum com placa Asus M5200AE, Windows XP, Intel Pentium M Processor 740, 1.73 GHz, 512MB RAM. Os valores dos tempos de processamento em segundos t(s) e os coeficientes de regressão R^2 dos polinômios ajustados estão apresentados na Tabela 4.1. Sendo que R^2 pode ser entendido como a fidelidade da representação.

Tabela 4.1. Duração do processamento em segundos t(s) e R² do polinômio ajustado de graus r em x e s em y, sendo r = s.

r=s	t(s)	R²
2	28	0,41547
3	86	0,47168
10	4.746	0,60450
20	62.675	0,64767

Os dados da Tabela 4.1 foram utilizados para estimar os outros valores de t(s) e o R², respectivamente, em função de r=s, para outros graus do polinômio. Estas equações estão apresentadas na Figura 4.3.



(a) (b)

Figura 4.3. (a) Tempo de processamento para se estimar polinômios em função de r=s. (b) R² dos polinômios estimados.

As equações apresentadas na Figura 4.3 (a) e Figura 4.3 (b), ilustram a duração do processamento em segundos t(s) e o coeficiente de regressão do polinômio ajustado R² para r=s variando de 0 a 20. Usando o polinômio obtido a partir do dado da Tabela 4.1, foram estimados os valores para o tempo de processamento para r=s variando de 30 a 500, conforme mostrado na Tabela 4.2. Para r=s=500, por exemplo, o modelo de representação de relevo para o estado de Minas Gerais é ajustado com R²=0,988 e seriam necessários 45,9 anos para gerar os coeficientes polinomiais, utilizando-se a mesma máquina que gerou os dados da tabela 4.1. O gráfico da Figura 4.4 ilustra a taxa de crescimento polinomial do tempo de processamento requerido para estimar o polinômio. Analisando as informações contidas na Tabela 4.2 é possível notar que, para gerar um polinômio de alta fidelidade para representar relevo, é necessário estimar um polinômio de grau elevado, o que implica em maior tempo de processamento. Ou seja, a fidelidade da representação é diretamente proporcional ao grau do polinômio e ao tempo de processamento requerido para estimá-lo.

Tabela 4.2. Estimativa da duração do processamento em segundos t(s) e R^2 do polinômio de graus r em x e s em y, sendo r = s variando de 30 a 500.

r=s	t(s)	R^2		r=s	t(s)	R^2
30	244.351	0,70102		225	129.630.416	0,9065
40	620.373	0,73037		250	178.414.108	0,9173
50	1.261.322	0,75313		275	238.117.737	0,927
60	2.237.780	0,77172		300	309.844.179	0,9359
70	3.620.332	0,78745		325	394.696.309	0,9441
80	5.479.562	0,80107		350	493.777.001	0,9516
90	7.886.055	0,81308		375	608.189.130	0,9587
100	10.910.393	0,82383		400	739.035.572	0,9652
125	21.637.523	0,84659		425	887.419.202	0,9714
150	37.770.215	0,86518		450	1.054.442.894	0,9772
175	60.411.344	0,88091		475	1.241.209.523	0,9828
200	90.663.786	0,89453		500	1.448.821.965	0,988

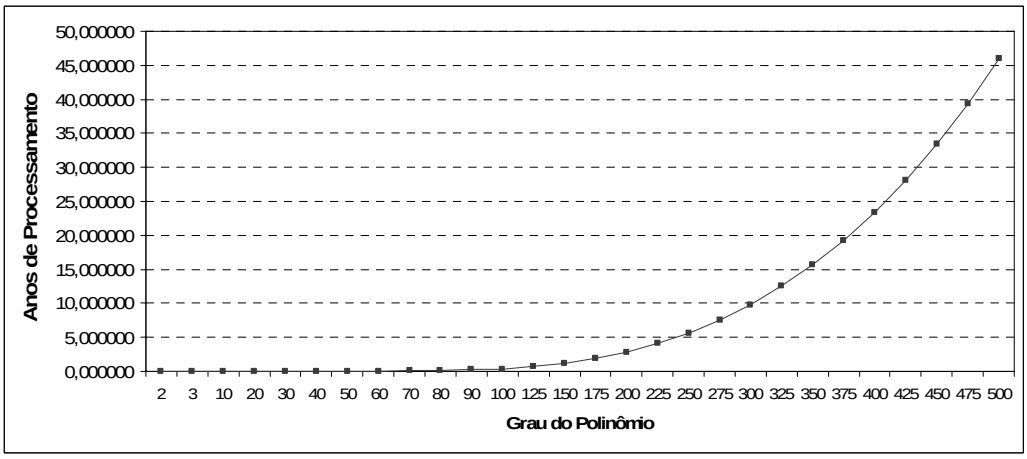


Figura 4.4. Tempo de processamento para se estimar polinômios em função de r=s.

4.3. Aplicação

No sistema desenvolvido serão comentados algumas modificações, quando houver, realizadas sobre os módulos pertencentes ao núcleo do sistema. Será comentado a forma com que os módulos adaptáveis foram configurados para atender o problema do estudo de caso e os detalhes de suas implementações. Não serão discutidos aspectos relacionados à robustez, tolerância a falhas, segurança, modelo de comunicações e outros aspectos de sistemas de processamento distribuído apresentados no detalhamento da metodologia.

A aplicação Coordenador foi implementado em C++ e o sistema operacional Linux foi escolhido por apresentar maior segurança, robustez e desempenho. As rotinas foram desenvolvidas para executar de forma distribuída. Foi utilizado o ambiente de desenvolvimento KDevelop 3.3.5 (http://www.kdevelop.org/), que é disponibilizado junto à distribuição do Debian 4.0 (http://www.debian.org).

A aplicação Agente foi desenvolvida também em linguagem C++. Poderia ter sido utilizada outra linguagem de programação já que diversas linguagens possuem acesso às funcionalidades da API *Socket* oferecem possibilidades de desenvolvimento utilizando *threads* e outras funcionalidades presentes no desenvolvimento do sistema. Microsoft Windows XP foi o sistema operacional escolhido para a implementação da aplicação Agente. Como a aplicação desenvolvida seria para utilizar o poder computacional de voluntários e o número de computadores conectados à Internet que utilizam o Windows XP representa em torno de 72% dos computadores [W3Schools], é mais conveniente disponibilizar aplicações que executam neste ambiente.

Porém, o sistema desenvolvido será testado em uma rede interna. Devido ao fato da computação voluntária depender de voluntários, o que implica na divulgação do projeto para atrair tais colaboradores. Por questões de tempo, não será possível executar o sistema com computação de voluntários espalhados pela Internet. Para isto, seria necessário criar uma página e desenvolver outras técnicas para divulgar o projeto. A resposta a esta divulgação poderia ser demorada, o que não é interessante para o projeto tendo em vista o espaço de tempo planejando para sua finalização. Desta forma, o processamento será distribuído entre as máquinas de uma rede local. A diferença básica entre os dois cenários é que na rede local o tempo de transmissão das mensagens é menor e existe confiabilidade entre as partes e na comunicação. Porém o sistema, ainda assim, possui mecanismos de segurança propostos da metodologia.

4.3.1. Aplicação Coordenador

Como previsto na metodologia de desenvolvimento, nenhum módulo pertencente ao núcleo do sistema necessitou de grandes modificações. De acordo com a metodologia apresentada, os módulos pertencentes ao núcleo deveriam manter suas funcionalidades, independente o tipo do problema. Os demais módulos foram devidamente adaptados ao problema específico de regressão não-linear.

clCoordenador

O módulo *clCoordenador* permaneceu sem nenhuma modificação em relação à metodologia e manteve suas funcionalidades que são: (a) fazer interface funcional com o usuário; (b) ativar as configurações através do módulo *clConfiguracao* e (c) iniciar a execução do sistema, que é feita ativando o módulo *clConexaoPrimaria*. No estudo de

caso não houve a necessidade da implementação do módulo opcional *clInterface*, já que a interface com o usuário seria simples e poderia ser implementada como uma linha de comando dentro do próprio módulo *clCoordenador*. A interface com o usuário é basicamente para iniciar, finalizar e visualizar o *status* da aplicação. Na Figura 4.5 está ilustrada a tela da aplicação capturada quando o *status* simples foi exibido. Na Figura 4.6 é ilustrada a tela relacionada ao *status* completo, que mostra as portas utilizadas em cada conexão secundária e algumas informações sobre cada uma delas.

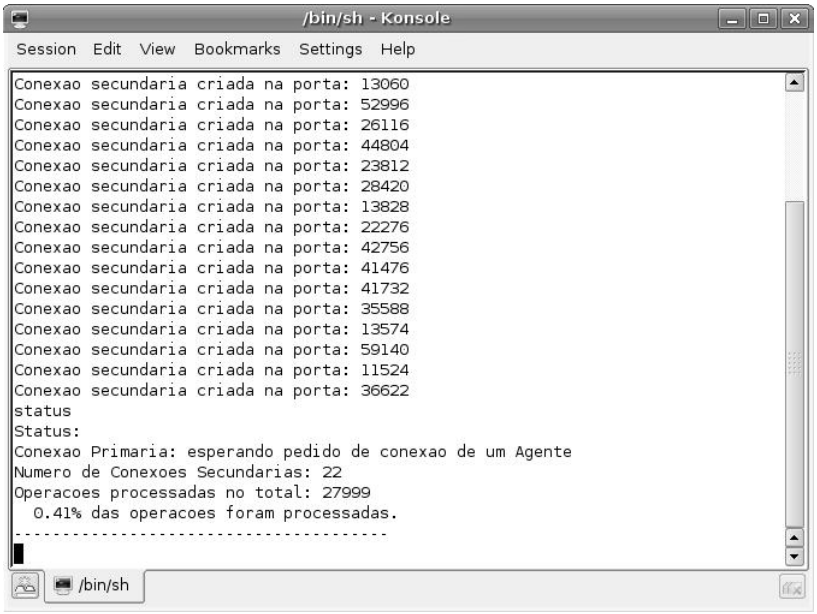


Figura 4.5. Tela da aplicação Coordenador exibindo seu *status* simples

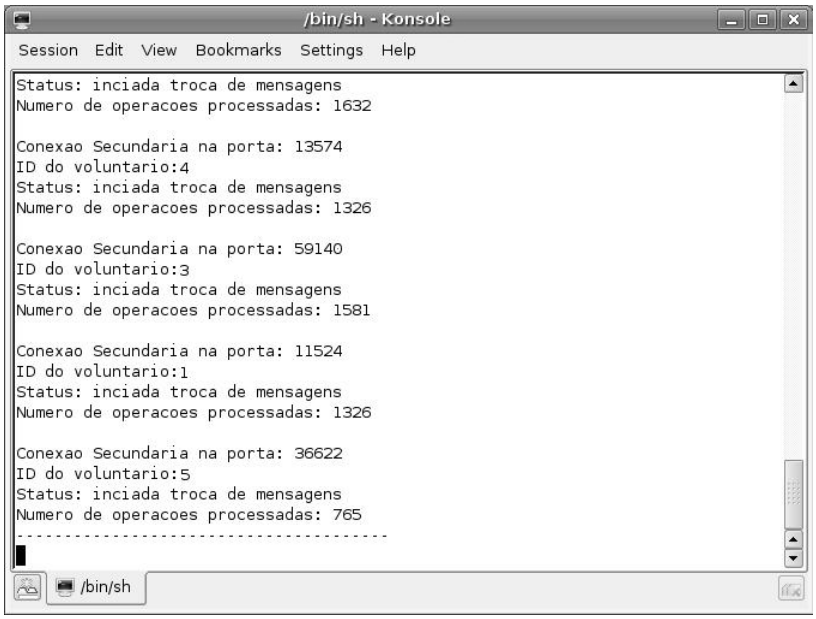


Figura 4.6. Tela da aplicação Coordenador exibindo seu *status* completo

clConexaoPrimaria

Como a metodologia de comunicação adotada no desenvolvimento do sistema é utilizando a API *Socket*, logo que o módulo *clConexaoPrimaria* é ativado, a sua configuração corresponde à configuração do *socket* utilizado na comunicação com as aplicações Agentes. Este *socket* foi configurado para usar o endereço IP corrente da máquina. A porta utilizada para a conexão primária é 35.600 (escolhida uma porta alta para evitar possíveis problemas de conflitos entre portas). O *socket* é configurado de acordo com as informações presente no arquivo de configurações, inclusive configurando-o com uma fila de espera de no máximo 50 pedidos de conexão. Depois do qüinquagésimo pedido enfileirado, os demais pedidos de conexão enviados pelas aplicações Agentes serão negados. Mais informações sobre as funcionalidades relativas à API *Socket* podem ser encontradas em Loosemore et al. (2007). Após a configuração é invocado, através do módulo *clAcessoDados*, a correção do arquivo de operações, que será detalhado na sessão relacionada ao módulo *clAcessoDados*. Após a correção, a próxima etapa é a verificação de pedidos de conexão. Depois que um pedido de conexão é efetuado por uma aplicação Agente e, se aceito, é realizada a verificação com relação ao número de conexões secundárias já criadas e o número máximo permitido. O número máximo foi definido para esta aplicação como sendo 60 conexões, para o caso específico do estudo de caso. Caso este número seja alcançado, todos os demais Agentes que tentarem estabelecer conexão serão informados que não há *slots* disponíveis para atendê-los. Caso exista *slots* disponíveis, este pedido é aceito e um novo módulo *clConexaoSecundaria* é criado. A estrutura de dados utilizada para representar as conexões secundárias foi uma lista encadeada, implementada com ponteiro que possibilita a inserção e retirada de elementos de forma simples e rápida.

clConexaoSecundaria

A execução do módulo *clConexaoSecundaria* permanece exatamente como especificado na metodologia. A única especificação definida no módulo é o número máximo de tentativas que o usuário terá para realizar a autenticação. Como comentado anteriormente, é este módulo que realiza a troca de mensagens entre Coordenador e Agente para a realização da autenticação. Por esta razão cabe a este módulo limitar o número de tentativas de autenticação, que neste caso foi definida como 5 tentativas. Na

Figura 4.7 está ilustra a tela da aplicação Coordenador exibindo o aviso de estouro do número de tentativas de autenticação de um possível voluntário.

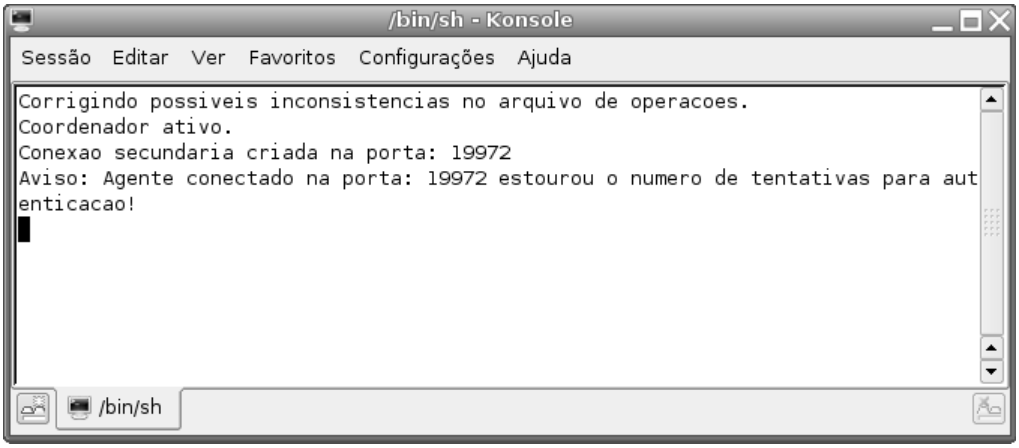


Figura 4.7. Tela da aplicação Coordenador com aviso sobre o estouro do número de tentativas de autenticação de um suposto voluntário.

clTrocaMsg

Depois de realizada a autenticação do voluntário, o método que realiza a troca de mensagens do módulo *clTrocaMsg* é invocado. Assim que for ativado, é realizada a configuração através dos parâmetros passado as módulo na sua inicialização. Além desta configuração, é realizada a configuração do módulo *clAcessoDados* que é feito com base em três informações: (a) mutex para acesso ao arquivo de operações; (b) grau do polinômio em x; e (c) grau do polinômio em y. Detalhes sobre a configuração do módulo *clAcessoDados* serão apresentados posteriormente. Uma modificação realizada neste módulo é a inclusão do envio do grau do polinômio para o Agente antes que a troca de mensagem propriamente dita se inicie. Este recurso foi acrescentado para possibilitar que a aplicação Agente possa processar operações relacionadas a polinômios de diferentes graus sem a necessidade de ser reconfigurado ou recompilado.

Com base na modelagem matemática desenvolvida é possível realizar o processamento de uma operação cujo grau do polinômio é definido em tempo de execução. Por esta razão, o grau do polinômio é enviado antes, para que a aplicação Agente possa ser configurada em tempo de execução. As próximas etapas da execução deste módulo seguem as especificações da metodologia proposta. A mensagem correspondente a um bloco de operações enviada pelo módulo *clTrocaMsg* está ilustrada abaixo. As informações relativas ao ID de conexão e ao bloco de operações

são separadas pelo caractere “#”. O campo relativo ao bloco de operações será detalhado posteriormente.

```
ID_Conexão#Bloco_Operações
```

clConfiguracao

Este módulo foi implementado para obter as configurações relativas à aplicação que estão armazenadas em um arquivo modo texto. O formato modo texto foi escolhido para que o usuário possa visualizar e editar as configurações do Coordenador diretamente no arquivo. Duas informações foram acrescentadas ao arquivo de configurações. Ambas as configurações definem o grau do polinômio cujos coeficientes serão estimados. Uma configuração define o grau do polinômio em x, a outra define o grau do polinômio em y. Uma imagem capturada do arquivo de configurações está ilustrada na Figura 4.8. Nesta figura é possível visualizar as configurações adotadas para esta aplicação. A configuração relativa ao tamanho de bloco de operações não foi implementada no arquivo de configurações, pois o tamanho do bloco é dependente do grau do polinômio em x e em y. Por esta razão, o tamanho do bloco é calculado dinamicamente em tempo de execução.

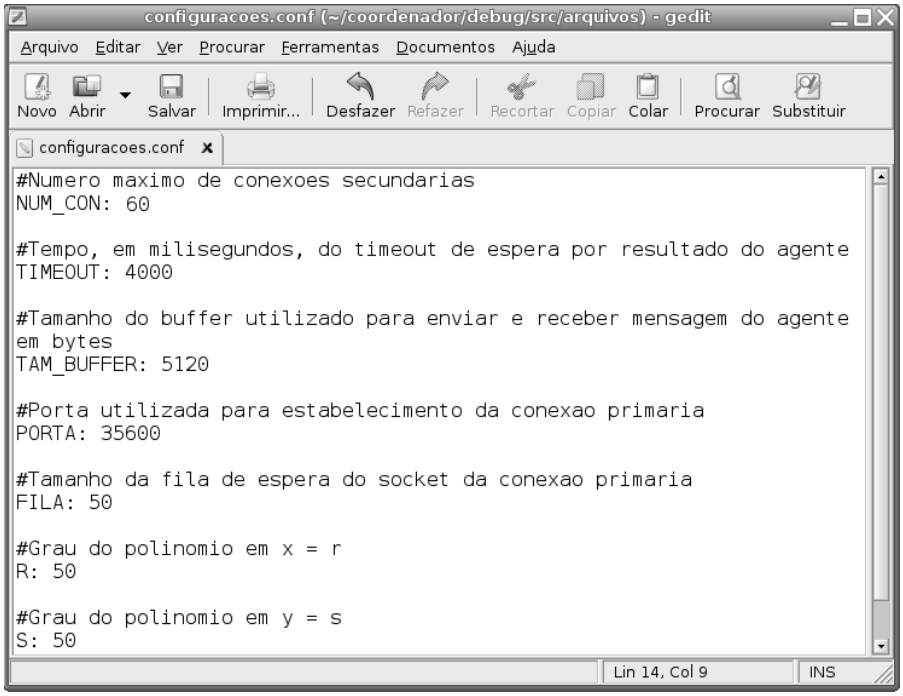


Figura 4.8. Arquivo de configurações: “configuracoes.conf”.

clSeguranca

Como havia sido comentado anteriormente, o módulo *clSeguranca* é composto por mais dois módulos, *clCripto* e *clAutenticacao*. Por esta razão as funcionalidades relacionadas à criptografia de informações e autenticação de voluntários são feitas pelos dois módulos auxiliares. A única função desempenhada por este módulo é gerar o ID de conexão com base em uma função da linguagem C++ para geração de números aleatórios. A cada requisição para geração de um número aleatório um inteiro entre 0 e 32.767 é gerado e transmitido para o módulo que o requisitou. As demais requisições para criptografia, descriptografia e autenticação são passadas para os módulos específicos. Neste módulo estão definidas ambas as chaves utilizadas para codificar e decodificar mensagens e a chave para codificar e decodificar informações do arquivo de voluntários.

clCripto

Como previsto na definição da metodologia, os funções especificadas para este módulo poderiam ser substituídas pelas funções de algum algoritmo de criptografia. No estudo de caso foi utilizado o método de criptografia de chave simétrica chamado AES [IAIK, 2008]. O algoritmo utilizado foi desenvolvido por Joan Daemen and Vincent Rijmen, anunciando em 2001 pelo National Institute of Standards and Technology [NIST] disponível em [AES]. Apesar de não utilizar as funções especificadas na metodologia, o algoritmo possui funções semelhantes, o que não comprometeu a implementação do módulo *clSeguranca*.

clAutenticacao

A autenticação da aplicação é baseada em nome e senha dos voluntários, como especificado na metodologia. Além destas informações, cada voluntário possui um ID que o identifica de forma única. Estas informações ficam armazenadas em um arquivo texto. A senha dos voluntários é armazenada criptografada para que não fique exposta e visível. O método de criptografia utilizado é o mesmo utilizado para criptografia das mensagens, porém utilizando uma chave diferente. No método para busca do nome de um voluntário, para que a autenticação seja realizada, é feita uma busca linear no arquivo com as informações dos voluntários. Caso o nome seja encontrado, é realizada a verificação da senha, que é descriptografada assim que é recuperada do arquivo. Os demais procedimentos são os mesmos especificados na metodologia.

clAcessoOperacoes/clSessao

No estudo de caso, ambos os módulos não foram implementados. Como os testes foram realizados em uma rede interna, com a quantidade de máquinas limitada, não seria interessante limitar ainda mais a capacidade de processamento, tendo em vista que o módulo *clAcessoOperacoes* reduz pela metade a capacidade de processamento total disponível. A intenção dos testes realizados é solucionar um problema real que necessite de grande poder de processamento. Por esta razão não houve o processamento redundante implementado pelo módulo *clAcessoDados* em conjunto com o módulo *clSessao*.

clAcessoDados

Este módulo é um dos mais adaptáveis da metodologia, pois a forma como será desenvolvido é dependente do problema específico e do formato no qual as operações são armazenadas. No estudo de caso, o arquivo de operações foi dividido em dois, um contendo os elementos da matriz A e o outro contendo os elementos da matriz B. O formato escolhido para armazenar as operações foi arquivo binário, por três motivos: (a) arquivos binários ocupam pouco espaço em relação a outros formatos como arquivos texto e arquivos XML, o que representa uma vantagem já que estes arquivos podem chegar a Gigabytes quando forem estimados polinômios de alto grau (grau acima de 200 em x e em y); (b) o acesso a arquivos binários é mais rápido que o acesso a informações em um banco de dados com arquivo texto ou XML, já que através de apontamento, qualquer posição de um arquivo binário pode ser acessado de forma direta sem busca linear e (c) por não possuir relação entre as operações não seria necessário um armazenamento mais sofisticado como um banco de dados.

Neste arquivo, a estrutura de dados representada pelo módulo *clOp* é gravada diretamente no arquivo. Não há distinção entre a representação das operações de ambos os arquivos (detalhes dos módulos *clOperacao* e *clOp* serão apresentados nas próximas seções). Os arquivos de operações de ambas as matrizes são criados estaticamente e antes da primeira execução da aplicação. Esta política foi adotada para que não ocorra problema com alocação de espaço em disco durante a execução da aplicação. Desta forma os arquivos de operações são criados previamente contendo todos os elementos. No início, os campos relativos ao resultado e ao ID do voluntário que processou cada operação estão em branco, porém o espaço para armazená-los já está alocado. À medida

que as operações vão sendo processadas, os espaços relativos a cada elemento vão sendo atualizados, ao invés de criados.

Com base no grau em x e em y do polinômio, valores passados ao módulo na sua configuração, é possível definir o número de elementos em cada arquivo, ou seja, o número de linhas multiplicado pelo número de colunas da matriz A (arquivo da matriz A) e o número de linhas da matriz B (arquivo da matriz B). Como estes valores são usados para definir o tamanho do bloco de operações, eles são passados para o módulo *clOperacao* para que este possa ser configurado. Nos arquivos de operações, a primeira informação, representada por um número inteiro ("long int" em C++), corresponde à "etapa_completa" que informa a posição, no arquivo, na qual todas as operações anteriores foram processadas ininterruptamente. Com base nesta informação, um método, acrescentado a este módulo, realiza uma busca à posição da próxima operação cujo estado seja "não processada". Se o método não encontrar nenhuma operação neste estado, significa que todas as operações do arquivo atual já foram processadas. Caso este arquivo seja referente à matriz A, o processo é executado novamente para a matriz B, caso contrário é assumido que todas as operações de ambos os arquivo já foram processadas.

Caso o método para busca da posição da próxima operação retorne a posição de uma operação ainda não processada, o número de sub-operações correspondente a um bloco de operações é lido e passado para o módulo *clOperacao* para que ele monte o bloco de operações. Todas as operações lidas são rotuladas como "em processamento" no arquivo de operações, o módulo *clOperacao*, contendo o bloco de operações é retornado para o módulo *clTrocaMsg* e o arquivo é fechado, permitindo que outros módulos *clAcessoDados* acessem este arquivo.

A posição da primeira sub-operação do bloco que foi lida é mantida no módulo para que no momento de gravar o resultado das sub-operações do bloco, o acesso seja feito de forma direta. Ao gravar o resultado de cada sub-operação do bloco, seus estados são rotulados como "processados". Caso haja algum erro no processamento do bloco, este erro é reportado ao módulo *clAcessoDados* que modifica o estado de todas as sub-operações do bloco para "não processada", o que possibilita que sejam processadas por outros Agentes.

Quando o módulo é requisitado para atualizar o índice "etapa_completa", ele realiza uma verificação a partir do atual índice. Um contador de posição é atualizado para toda operações consecutiva que o módulo encontrar após o índice e que tenha sido

processada. Na primeira operação que encontre que ainda não tenha sido processado, o módulo encerra a atualização e o índice é atualizado com base no contador e o valor atualizado é gravado no arquivo.

No método responsável pela correção dos arquivos de operações, é realizada uma busca linear em ambos os arquivos de operações a fim de identificar as operações que estão rotuladas como “em processamento”. Todas estas operações são rotuladas para “não processadas”. Como este método é executado pelo módulo *clConexaoPrimaria* antes de iniciar a distribuição de processamento e no início da execução da aplicação, toda operação “em processamento” é fruto de interrupção inesperada da execução do Coordenador, e por esta razão devem ser rotuladas como “não processadas”.

Outros dois métodos foram acrescentados para auxiliar a realização do processamento final. São métodos para carregar os resultados dos elementos de ambos os arquivos de operações na memória para que o processamento final seja realizado. O Diagrama de Classe que define a estrutura do módulo está representado na Figura 4.9.

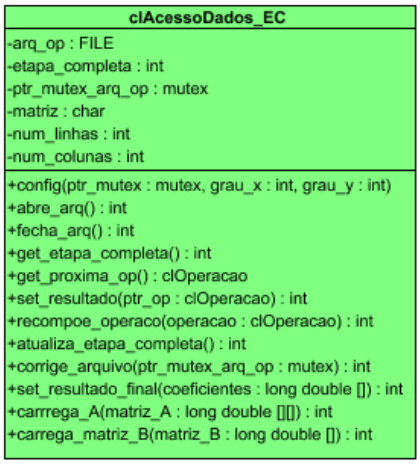


Figura 4.9. Diagrama de Classe do módulo clAcessoDados

clOp

Como será utilizado o processamento em blocos de operações, este módulo representa uma única operação (ou sub-operação). Sua tarefa é apenas armazenar as informações necessárias para cada operação única. Cada operação a ser processada corresponde a um elemento da matriz A ou da matriz B. Ambos os elementos são representados pelo mesmo módulo, não há distinção entre a estrutura de dados que representam os elementos de ambas as matrizes. As informações relacionadas a cada operação estão

listadas abaixo. Estas informações são gravadas nos arquivos de operações para cada elemento das matrizes A e B e estão detalhadas a seguir:

- ID da operação: identifica unicamente uma operação em cada arquivo de operações;
- status: indica uma operação (0) ainda não foi processada, (1) operação em processamento ou (2) já processada;
- linha: indica qual linha a operação pertence na respectiva matriz;
- coluna: indica qual linha a operação pertence na matriz A. Caso o módulo represente um elemento da matriz B, este campo será igual a zero;
- resultado: guarda o resultado a operação;
- ID voluntário: caso a operação já tenha sido processada, este campo guarda a ID do voluntário responsável pelo processamento.

clOperacao

Para esta aplicação, o tamanho do bloco de operações é definido com base no grau do polinômio. Como o número de linha e colunas da matriz A e o número de linhas da matriz B são dependentes do grau do polinômio, então o tamanho é calculado em função do grau em x e em y do polinômio a ser estimado. O número de operações por bloco deve ser um divisor exato do número de linhas e de colunas da matriz A e ao mesmo tempo do número de linhas da matriz B para que sejam montados blocos sempre do mesmo tamanho. A equação que define o número de operações em um bloco está ilustrada na Equação (19) que foi deduzida com base nas Equações (17) e (18).

$$MA\_L = (grau\_r + 1) \times (grau\_s + 1) \tag{17}$$

$$MA\_L = MA\_C = MB\_L \tag{18}$$

$$TAM\_BLOCO = grau\_r + 1 \tag{19}$$

Onde:

- grau_r: grau do polinômio em x
- garu_s: grau do polinômio em y

- MA_L: número de linhas da matriz A
- MA_C: número de colunas da matriz A
- MA: número de elementos da matriz A
- MB_L: número de linhas da matriz B
- TAM_BLOCO: número de elementos de um bloco de operações.

Como o espaço necessário para armazenar um bloco de operação é alocado na criação do módulo *clOperacao*, a tarefa de criar um bloco de operações com as operações passada pelo módulo *clAcessoDados* se resume a armazenar cada sub-operação em uma posição específica no vetor (bloco) de operações.

Antes que um bloco de operações seja enviado a um Agente, o módulo *clTrocaMsg* requisita a este módulo a montagem da mensagem texto a ser enviada, pois apenas este módulo conhece a formatação dos blocos de operações. Pelas Equação (15) e Equação (16) é possível enviar apenas a referência à linha e coluna de um elemento da matriz (A ou B) a ser processada. No caso da representação de operações através de bloco de operações, a referência, ao invés de ser relativa a um único elemento da matriz, é relativa à coluna inicial e final do bloco, e a linha deste bloco quando o bloco pertencer à matriz A, e linha inicial e final quando este ele pertencer à matriz B, cadeia de caracteres gerada pelo módulo para representar um bloco de operações é ilustrada abaixo.

Elementos pertencentes à matriz A:

- `Matriz#Coluna_inicial#Coluna_final#Linha`

Exemplo:

- `A#50#101#50`

Elementos pertencentes à matriz B:

- `Matriz#Linha_inicial#Linha_final`

Exemplo:

- `B#0#50`

Exemplo de uma mensagem completa enviada a um Agente contento um bloco de operação com elementos da matriz A:

- `1299#A#305#356#1784`

Quando o módulo *clTrocaMsg* recebe uma mensagem contendo o resultado de um bloco de operações, esta mensagem é passada a este módulo para ser desmontada. Nas mensagens contendo resultados de processamento, o resultado de cada sub-operação do bloco é separada pelo símbolo "#" . A função do módulo é obter cada resultado e registrá-lo nos respectivos módulos *clOp*. Ao final do conjunto de resultados estará o tempo de processamento gasto pelo bloco de operações, que foi acrescentado à mensagem pelo Agente. Um aviso de "finalizar conexão" pode estar concatenado à mensagem caso o voluntário deseje encerrar a aplicação Agente. Um exemplo de uma mensagem recebida pelo módulo *clTrocaMsg* contendo os resultados de um bloco de operações de tamanho 10 com aviso de finalização está ilustrado abaixo. A primeira informação, relativa ao ID de conexão, as duas últimas informações são relativas ao tempo de processamento (em segundos) e o aviso de finalizar, que não são passadas para o método que desmonta a mensagem.

- `2387523#3.66106e+08#2.6992e+11#2.25938e+14#2.03779e+17#1.93909e+20#1.92234e+23#1.96816e+26#2.06785e+29#2.21901e+32#2.42351e+35#7#finalizar`

clLog

Neste módulo não foi acrescentado nenhuma informação ou métodos além do que foi proposto na metodologia. As informações a serem registradas que foram propostas são suficientes para acompanhar a execução do sistema. As informações do log foram gravadas em um arquivo texto. Um trecho do arquivo de registro, gravado durante a execução do sistema, está ilustrado na Figura 4.10. Como era previsto, por ter sido executado em uma rede interna, o tempo gasto na transmissão das mensagens foi menor que 1 segundo.

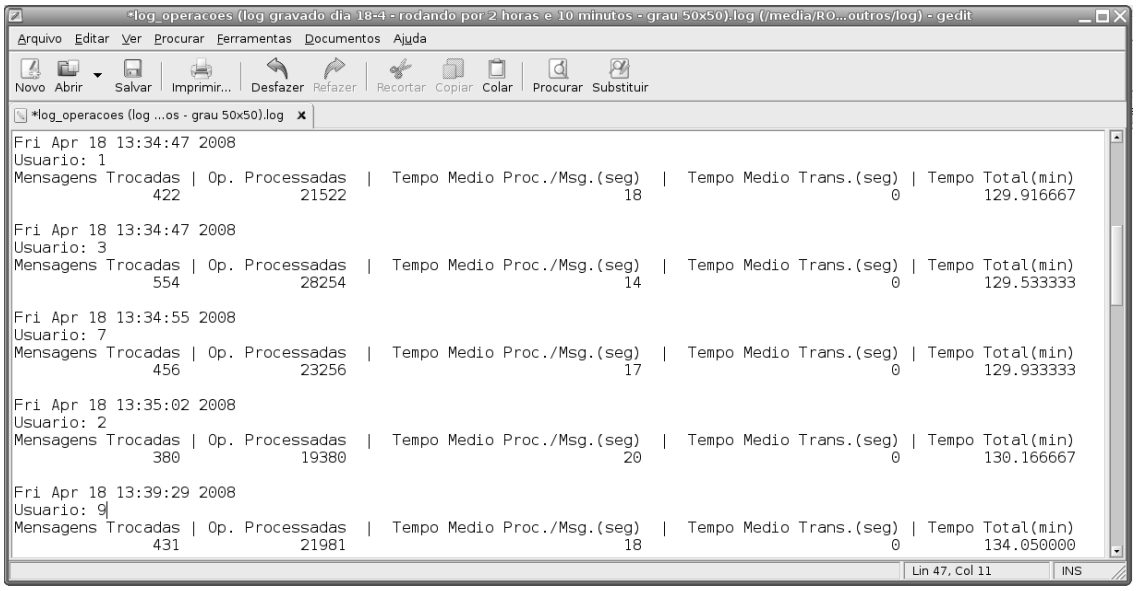


Figura 4.10. Trecho do arquivo de log da aplicação Coordenador

clProcessamentoFinal

O processamento demandado pela representação de relevo através de polinômio é fruto dos cálculos dos elementos da matriz A e matriz B. Porém este cálculo não é o único necessário para se estimar o polinômio. Depois que todos os elementos de ambas as matrizes forem calculados é necessário resolver o sistema matricial para se chegar aos coeficientes do polinômio, já que os elementos da segunda matriz representam tais coeficientes. Por esta razão, este problema específico necessita de um processamento final, que é a resolução do sistema de matrizes.

O método, responsável por configurar este módulo, recebe apenas duas informações, o grau do polinômio em x e em y. Estas informações são necessárias para executar o processamento final, pois o módulo tem acesso, através do módulo *clAcessoDados*, aos arquivos de operações processadas. Porém o método principal é o método que realiza o processamento final. Sua função básica é alocar espaço na memória para carregar os resultados das operações de ambos os arquivos de operações. As operações são carregadas na memória pelos métodos do módulo *clAcessoDados* que realizam esta função. Depois de carregados os resultados, o método responsável para resolver o sistema de matrizes é executado.

O método “sistema()”, responsável pela resolução do sistema de matrizes, realiza os cálculos necessários para se obter os coeficientes. Depois que os coeficientes são calculados, o método “processamento_final gera um arquivo texto com os coeficientes calculados através do método “set_resultado_final()” do módulo *clAcessoDados*. Na

Figura 4.11 está ilustrada parte de um arquivo contendo os coeficientes estimados do polinômio de grau 10 em x e em y.

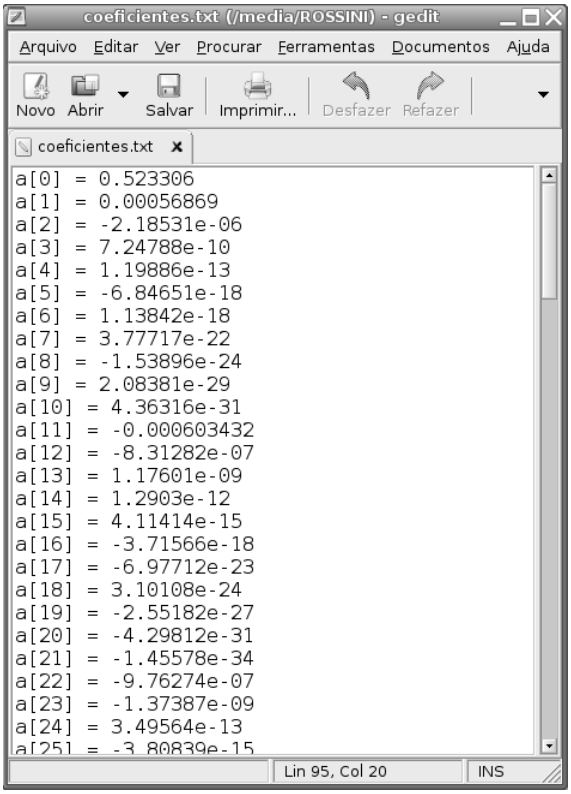


Figura 4.11. Arquivo com coeficientes do polinômio de grau 10 em x e em y.

4.3.2. Aplicação Agente

Houve a necessidade de replicação de informações junto às aplicações Agentes. O MDE do estado de Minas Gerais foi replicado em cada aplicação Agente. Desta forma se evita que toda a informação relativa ao relevo do estado tenha que ser trafegado pela rede. Como está informação é estática, ou seja, não mudará no decorrer da execução do sistema, é possível replicá-la sem afetar o processamento das operações já que esta informação será enviada junta com a aplicação Agente.

clAgente

De acordo com o que foi especificado na metodologia, este módulo permanece muito simples. Suas funções básicas são ativar o carregamento das configurações através do módulo *clConfiguracao* e em seguida iniciar o módulo *clConexao*. Diferente do módulo *clCoordenador*, este módulo não é o único responsável pela interface com o usuário. A cargo do módulo *clAgente* estão as funcionalidades relacionadas à ativação da

aplicação, obtenção do *status* e finalização da aplicação. Por não ter sido necessário a implementação de uma interface mais sofisticada, o módulo opcional *clInterface* não foi utilizado, e a interface com o usuário foi implementado via linha de comando.

clConexao

A API *Socket* foi a forma utilizada por este método para estabelecer conexão com o Coordenador, seguindo a mesma política adotada na aplicação Coordenador. Por esta razão, o *socket* utilizado é configurado com o endereço IP e porta, que são fixos e predefinidos, utilizados pela aplicação Coordenador. As demais funcionalidades são as mesmas especificadas na metodologia. Nenhum método ou funcionalidades precisou ser acrescentado na implementação deste módulo. Caso o voluntário exceda o número máximo de tentativas de autenticação definidas como sendo 5, ele recebe um aviso sobre o estouro e a conexão é finalizada. Na Figura 4.12 está ilustrada a tela da aplicação Agente exibindo o aviso de finalização de conexão quando o número de tentativas de autenticação foi ultrapassado.

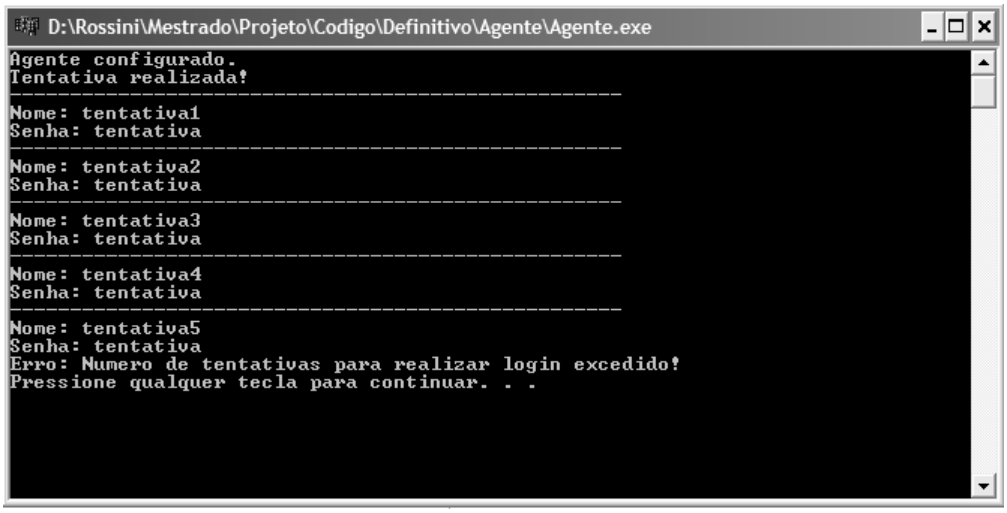


Figura 4.12. Tela da aplicação Agente com aviso sobre o estouro do número de tentativas de autenticação.

clTrocaMsg

Este módulo, também pertencente ao núcleo do sistema necessitou apenas de uma pequena modificação. Foi acrescentado um método responsável por receber uma configuração do Coordenador antes que as mensagens começassem a ser trocadas. Esta configuração está relacionada ao grau do polinômio a ser processado. A modelagem matemática e as fórmulas desenvolvidas Equação (15) e Equação (16), possibilitaram

que polinômios sejam estimados independente do seu grau, pois as equações são independentes do grau do polinômio.

As Equação (15) e Equação (16) foram implementadas na aplicação Agente (módulo clProcessamento) e para o Agente realizar o processamento corretamente, ele deve conhecer o grau do polinômio. Por esta razão, este grau é enviado pelo Coordenador antes que as primeiras operações sejam enviadas. O método "recebe_grau()" é responsável por receber o grau do polinômio e passá-lo ao módulo clProcessamento para que este seja devidamente configurado. A mensagem contendo o grau do polinômio segue o formato descrito abaixo. Após esta etapa, o fluxo de execução segue como especificado na metodologia.

Mensagem:

- `Id de conexão#grau em x#grau em y`

Exemplo

- `4234#50#50`

clProcessamento

Este é o módulo principal da aplicação Agente, sendo responsável pelo processamento das operações recebidas do Coordenador. Alguns métodos e atributos foram acrescentados para auxiliar no processamento das operações. Os acréscimos são listados e comentados abaixo. Na Figura 4.13 está ilustrado um Diagrama de Classe deste módulo que possui os seguintes atributos:

- status: armazena o estado atual do módulo;
- matriz: especifica a matriz a do elemento a ser processado;
- coluna inicial: especifica a primeira coluna do bloco de operações;
- coluna final: especifica a última coluna do bloco de operações. No caso do elemento pertencer à matriz B, os atributos relacionados à coluna não serão utilizados;
- linha inicial: especifica a primeira linha do bloco de operações. No caso do elemento pertencer à matriz A, este atributo será utilizado para indicar a linha do bloco;
- linha inicial: especifica a última linha do bloco de operações, caso o elemento a ser processado pertença à matriz B;

- número de elementos: especifica o número de elementos contido em um bloco;
- número de altitudes: número total de altitudes contidas no MDE (1343x1043);
- x, y e z: vetores que armazenam a linha, coluna e valor de cada altitude, respectivamente, contida no MDE;
- resultados: vetor que contém os resultados de todos os elementos de um bloco de operações;
- r: grau do polinômio em x;
- s: grau do polinômio em y

Os métodos do módulo clProcessamento são os seguintes:

- set_grau_polinômio(): método que configura o módulo a partir da mensagem contendo o grau do polinômio, enviada pelo Coordenador;
- mde(): carrega as informações contidas no MDE nos vetores X, Y e Z;
- getA(): implementa a Equação (15). Este método é responsável por calcular um determinado elemento da matriz A, com base na linha e coluna passadas como parâmetros;
- getB(): implementa a Equação (16). Este método é responsável por calcular um determinado elemento da matriz B, com base na linha passada como parâmetro;
- processa(): responsável por calcular todos os elementos de um bloco de operações através da chamada ao método correspondente a matriz do bloco, getA() ou getB();
- desmonta_msg(): responsável por configurar o módulo através das informações da mensagem contendo um bloco de operações enviada pelo Coordenador;
- monta_msg(): responsável por montar uma mensagem texto contendo todos os resultados de todas as sub-operações de um bloco de operações. O formato desta mensagem foi apresentado sessão relativa ao módulo *clOperacao* da aplicação Coordenador.

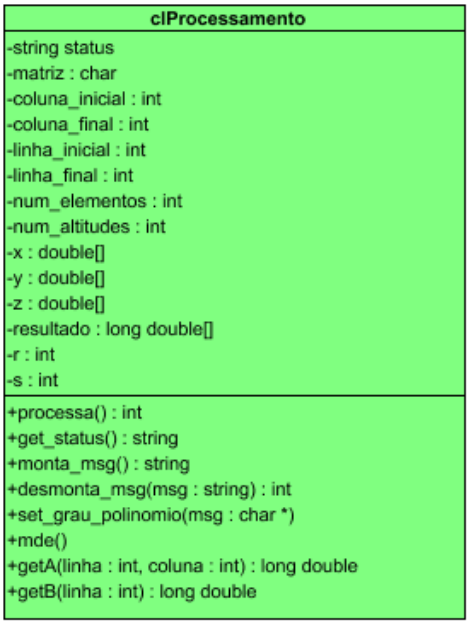


Figura 4.13. Diagrama de Classe do módulo clProcessamento.

clConfiguracao

Neste módulo não houve modificação nas funcionalidades propostas na metodologia. As informações relativas à configuração da aplicação continuaram as mesmas. Estas informações foram armazenadas em arquivo texto para o voluntário visualizar e modificá-las com facilidades. As configurações da aplicação Agente são apresentadas na Figura 4.14.

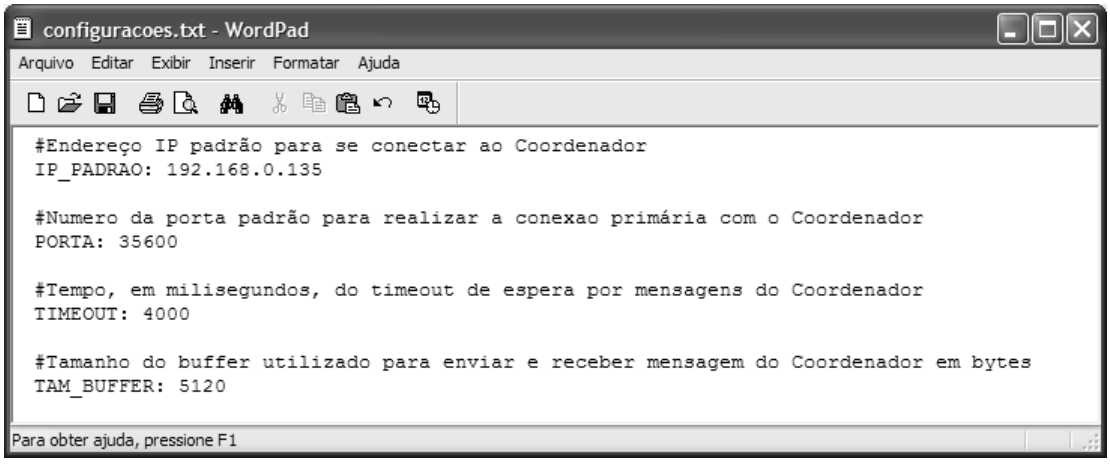


Figura 4.14. Arquivo de configurações da aplicação Agente

clCripto

Como a única funcionalidade relacionada à segurança implementada na aplicação Agente foi a criptografia da mensagem contendo as informações para realizar a

autenticação, não houve necessidade de implementar o módulo *clSeguranca*. A implementação deste módulo é aconselhada para aplicações que exijam outras formas de segurança, que conseqüentemente exigem outros módulos. Desta forma, o módulo *clSeguranca* seria o intermediário entre os módulo que utilizariam os métodos de segurança e os módulos que implementariam estes métodos. Como foi utilizada apenas a criptografia e a descriptografia de mensagens, o módulo *clConexao* interage diretamente com o módulo *clCripto*. O método de criptografia e a chave para criptografar e descriptografar mensagens utilizado neste módulo é o mesmo utilizado na aplicação Coordenador.

clLog

Este é outro método no qual não houve necessidade de modificações em relação ao que foi proposto pela metodologia. As informações registradas são as mesmas especificadas assim como as funcionalidades. Um trecho registra durante algumas execuções de uma aplicação Agente está ilustrado na Figura 4.15.

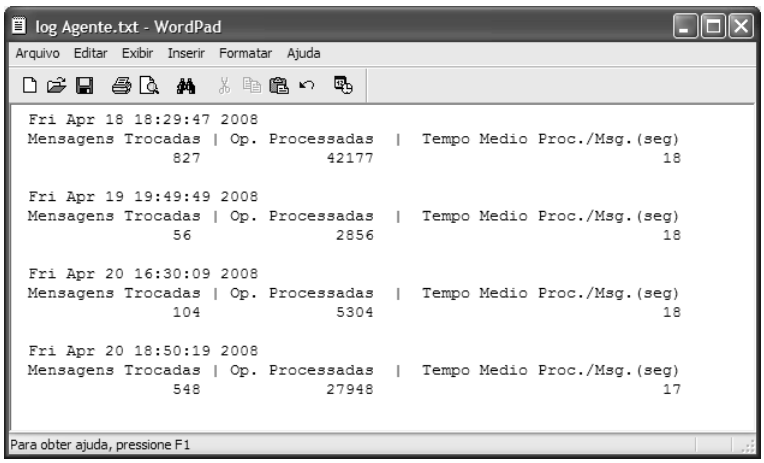


Figura 4.15. Arquivo de Log da aplicação Agente.

Nos anexos A.7 e A.8 estão representados, através de Diagramas de Componentes a ligação entre os componentes da aplicação Coordenador e da aplicação Agente, respectivamente. Os componentes, códigos fontes e arquivos auxiliares, representados nos diagramas foram agrupados de acordo com suas funcionalidades. Os componentes "clConfig.cpp", "clConfig.h" e "configuracoes.conf" foram agrupados relativemente à configuração de cada aplicação, por exemplo. No anexo A. 9 é apresentado um Diagrama de Execução que ilustra as interações das aplicações com informações externas, assim como o ambiente de execução de cada aplicação.

4.3.3. Execução da Aplicação

Alguns testes, mais simples, foram realizados durante o desenvolvimento do sistema com o auxílio de uma simulação em máquina virtual. O software utilizado foi VMware Workstation 5.5.2 que simula uma máquina virtual em ambiente Windows. Nesta máquina virtual podem ser instalados outros sistemas operacionais como Windows ou Linux. Nos primeiros testes realizados, foi instalado o sistema operacional Linux, distribuição Debian 4.0. Como a máquina virtual simulada pelo aplicativo se comunica com o sistema operacional hospedeiro (Windows) como se estivessem interligados por uma rede, foi possível realizar alguns testes relativos à comunicação entre as aplicações Agente e Coordenador durante o período de desenvolvimento.

Após finalizado o sistema, optou-se por testá-lo em um ambiente real, em uma rede de computadores onde diversas aplicações Agentes poderiam se conectar à aplicação Coordenador. O sistema desenvolvido foi testado em dois laboratórios do Departamento de Informática da Universidade Federal de Viçosa. Na Tabela 4.3 estão as informações relativas aos laboratórios utilizados. Todos os computadores de ambos os laboratórios estavam interligados por uma rede interna. Nas tabelas estão representadas apenas as configurações que são relevantes para o sistema.

Tabela 4.3. Configurações das máquinas do Laboratório 416 e 408

Identificador do grupo de configurações	Sala	Quantidade de máquinas	Processador	Memória Principal	Sistema Operacional	Coordenador / Agente
A	416	1	AMD Duron™ 1.19 Ghz	512 MB	Linux Suse 10.1	Coordenador
B	416	17	AMD Duron™ 1.19 Ghz	512 MB	Windows XP	Agente
C	416	5	AMD Sempron™ 1.49 Ghz	512 MB	Windows XP	Agente
D	408	15	Intel Core 2 Duo 2.20 GHz	1 GB	Windows XP 64 bits	Agente

Foram realizados quatro testes para estimar os coeficientes de quatro polinômios de diferentes graus em x e em y. Nas próximas seções, serão apresentadas as informações relativas a cada teste realizado. O grau referido em cada sessão equivale ao grau do polinômio em x e em y. É pertinente salientar que, além da execução do sistema para cálculo dos coeficientes dos polinômios, foram realizados testes sobre o sistema para avaliar aspectos como robustez, segurança e consumo do processador pela aplicação Coordenador. Todas as funcionalidades descritas na metodologia relacionadas

aos aspectos citados foram testadas. Os testes de robustez e segurança foram os testes de validação das técnicas propostas e descritas na metodologia.

Polinômio de grau 5

Por não requerer grande poder de processamento para ser estimado, neste teste foram utilizadas apenas a máquina do grupo A (Coordenador) e cinco máquinas do grupo B (Agentes). As informações relativas à distribuição de processamento para obtenção do polinômio estão listadas na Tabela 4.4. Na Figura 4.16 (a) está ilustrada a imagem do relevo de Minas Gerais, gerada a partir do polinômio estimado. Nesta imagem, assim como nas demais, os tons claros representam altitude mais elevadas. Na Figura 4.16 (b) está ilustrada a imagem do mapa de altitudes original, para termos de comparação.

Tabela 4.4. Informações relativas à execução do sistema para estimativa dos coeficientes do polinômio do grau 5 em x e em x.

Tempo de Processamento	40 segundos
Aplicação Coordenador	
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz A	35,4 KB
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz B	1.012 Bytes
Memória ocupada pela aplicação	1,2 MB
Consumo Médio de Processador	2%
Tamanho médio das mensagens enviadas	12 Bytes
Tráfego de rede médio (mensagens recebidas)	450 B/s
Aplicação Agente	
Memória ocupada pela aplicação	19 MB
Consumo Médio de Processador	100%
Tamanho médio das mensagens enviadas	90 Bytes
Número de máquinas utilizadas	5 (Grupo B)
Tempo médio de processamento de um bloco de operações	1 segundo
Tráfego de rede médio (mensagens enviadas)	90 B/s

(a) (b)

Figura 4.16. (a) Imagem gerada a partir do polinômio de grau 5 em x e em y. (b) Imagem gerada a partir do MDE de Minas Gerais

Polinômio de grau 10

Neste teste foram utilizadas todas as máquinas do Grupo B e C para executar as aplicações Agentes, o Coordenador foi executado na máquina do Grupo A. Na Tabela 4.5 estão listadas as informações relativas à execução do sistema. Na Figura 4.17 está ilustrada a imagem do relevo de Minas Gerais, gerada a partir do polinômio estimado.

Tabela 4.5. Informações relativas à execução do sistema para estimativa do polinômio do grau 10 em x e em y.

Tempo de Processamento	4 min 30 s
Aplicação Coordenador	
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz A	400,3 KB
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz B	3,3 KB
Memória ocupada pela aplicação	1,6 MB
Consumo Médio de Processador	3%
Tamanho médio das mensagens enviadas	14 Bytes
Tráfego de rede médio (mensagens recebidas)	1,39 KB/s
Aplicação Agente	
Memória ocupada pela aplicação	19 MB
Consumo Médio de Processador	100%
Tamanho médio das mensagens enviadas	155 Bytes
Número de máquinas utilizadas	20 (Grupo B)
Tempo médio de processamento de um bloco de operações	3 segundos
Tráfego de rede médio (mensagens enviadas)	51 B/s
Número de máquinas utilizadas	5 (Grupo C)
Tempo médio de processamento de um bloco de operações	2 segundos
Tráfego de rede médio (mensagens enviadas)	77 B/s

Figura 4.17. Imagem gerada a partir do polinômio de grau 10 em x e em y.

Polinômio de grau 20

Neste teste foram utilizadas todas as 25 máquinas da Sala 416 para executar as aplicações Agente. O Coordenador foi executado na máquina do Grupo A, as demais máquinas pertencem ao Grupo B e C. Na Figura 4.18 está ilustrada a imagem do relevo de Minas Gerais, gerada a partir do polinômio estimado. Na Tabela 4.6 estão listadas as informações relativas à execução do sistema.

Tabela 4.6. Informações relativas à execução do sistema para estimativa do polinômio do grau 20 em x e em y.

Tempo de Processamento	50 min 20 s
Aplicação Coordenador	
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz A	5,2 MB
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz B	12,1 KB
Memória ocupada pela aplicação	1,7 MB
Consumo Médio de Processador	3%
Tamanho médio das mensagens enviadas	14 Bytes
Tráfego de rede médio (mensagens recebidas)	933 B/s
Aplicação Agente	
Memória ocupada pela aplicação	19 MB
Consumo Médio de Processador	100%
Tamanho médio das mensagens enviadas	280 Bytes

Número de máquinas utilizadas	20 (Grupo B)
Tempo médio de processamento de um bloco de operações	8 segundos
Tráfego de rede médio (mensagens enviadas)	35 B/s
Número de máquinas utilizadas	5 (Grupo C)
Tempo médio de processamento de um bloco de operações	6 segundos
Tráfego de rede médio (mensagens enviadas)	47 B/s

Figura 4.18. Imagem gerada a partir do polinômio de grau 20 em x e em y.

Polinômio de grau 50

Neste teste foram utilizadas todas 17 máquinas do Grupo B, 5 máquinas do Grupo C e 14 máquinas do Grupo D para executar as aplicações Agente, o Coordenador foi executado na máquina do Grupo A. Na Figura 4.19 está ilustrada a imagem do relevo de Minas Gerais, gerada a partir do polinômio estimado. Na Tabela 4.7 estão listadas as informações relativas à execução do sistema. Como as máquinas do Grupo D possuem dois núcleos de processamento, foram executados duas aplicações Agente em cada máquina, desta forma totalizando 52 aplicações Agente sendo executadas.

Tabela 4.7. Informações relativas à execução do sistema para estimativa do polinômio do grau 50 em x e em y.

Tempo de Processamento	25 h 48 min
Aplicação Coordenador	
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz A	206,5 MB
Espaço ocupado pelo arquivo binário contendo informações sobre os elementos da matriz B	81,3 KB
Memória ocupada pela aplicação	2,4 MB
Consumo Médio de Processador	10%
Tamanho médio das mensagens enviadas	16 Bytes
Tráfego de rede médio (mensagens recebidas)	4.03 KB/s
Aplicação Agente	
Memória ocupada pela aplicação	19 MB
Consumo Médio de Processador	100%
Tamanho médio das mensagens enviadas	670 Bytes
Número de máquinas utilizadas	17 (Grupo B)
Tempo médio de processamento de um bloco de operações	20 segundos
Tráfego de rede médio (mensagens enviadas)	33 B/s
Número de máquinas utilizadas	5 (Grupo C)
Tempo médio de processamento de um bloco de operações	16 segundos
Tráfego de rede médio (mensagens enviadas)	42 B/s
Número de máquinas utilizadas	15 (Grupo D)
Tempo médio de processamento de um bloco de operações	6 segundos
Tráfego de rede médio (mensagens enviadas)	223 B/s

Figura 4.19. Imagem gerada a partir do polinômio de grau 50 em x e em y.

5. Resultados e Discussão

5.1. Metodologia

No estudo de caso desenvolvido ficou claro que a metodologia proposta consegue descrever as funcionalidades necessárias para se desenvolver uma aplicação de distribuição de processamento utilizando computação voluntária. O estudo de caso representou uma contribuição significativa para a metodologia por duas razões. Durante o desenvolvimento do estudo de caso verificou-se que a metodologia estava de acordo com a realidade da distribuição de processamento quando se utiliza computação voluntária. A outra contribuição foi ilustrar, de modo objetivo, as características da distribuição de processamento. Vários projetos semelhantes foram estudados para se extrair os pontos em comum e para se conhecer os mecanismos necessários para distribuir processamento entre voluntários através da Internet. Porém, nestes estudos não foram possíveis obter informações detalhadas sobre este tipo de sistema.

Grande parte do material pesquisado estava relacionado à apresentação de resultados e não na apresentação da metodologia adotada. Por esta razão, desenvolver o sistema e aplicá-lo a um estudo de caso real foi fundamental para descobrir funcionalidades não descritas na literatura, mas que poderiam ser acrescentadas à metodologia para enriquecê-la. Um exemplo é o processamento em bloco. Foi no estudo de caso que se notou a importância de possibilitar que operações fossem enviadas aos Agentes em blocos. Esta funcionalidade é importante na metodologia já que é uma característica desejável e utilizada em outros projetos. Projetos como SETI@Home utilizam uma forma de envio de operações que podem ser entendidas como blocos. A aplicação coordenadora envia faixas de freqüência, ao invés de freqüências únicas, a serem analisadas, o que diminui a freqüência que a aplicação cliente se comunica com a aplicação coordenadora por ficar mais tempo realizando o processamento.

Deve-se notar que a metodologia proposta é independente de linguagem de programação ou sistema operacional possibilitando que aplicações Coordenador e Agentes, desenvolvida em diferentes linguagens, interajam sem problemas de incompatibilidade. Toda linguagem de programação que possua suporte para desenvolvimento de aplicações para Internet e implementação de *threads* pode ser utilizada para desenvolver ambas as aplicações. A metodologia não é limitante em relação ao sistema operacional também. Sistema operacional que implemente recursos

necessários para aplicações Internet, poderá ser utilizado como plataforma para o desenvolvimento das aplicações. E aplicações desenvolvidas em sistemas operacionais diferentes, como no estudo de caso, poderão interagir já que as informações trocadas entre elas são em formato textos. Troca de mensagens em formato texto é uma sugestão da metodologia, não uma limitação. Outras formas de comunicação proprietária poderão ser utilizadas, como alguns *middleware* de comunicação. A utilização de tecnologia proprietária poderá implicar em algumas restrições ao sistema, como a impossibilidade de utilizar sistemas operacionais ou linguagens de programação diferentes para desenvolver as aplicações.

5.2. Estudo de Caso

O estudo de caso escolhido e implementado foi fundamental para o sucesso deste trabalho em vários aspectos, desde a adequação da metodologia proposta até o desempenho do sistema na distribuição de processamento. Os requisitos do sistema desenvolvido foram atendidos pela metodologia proposta e poucas funcionalidades foram acrescentadas à metodologia incentivada pelo estudo de caso. Ambas as aplicações desenvolvidas ocupam pouco espaço de memória, como apresentado na sessão 4.3.3. O espaço ocupado pela aplicação Agente chegou a 19 MB. Por uma opção estratégica, optou-se por fazer que a aplicação Agente trabalhasse com toda informação relativa ao MDE do estado é carregada na memória principal para minimizar acesso ao disco rígido. Caso seja utilizado um MDE que inviabilize o carregamento das informações na memória principal, a aplicação Agente passará a acessar tais informações no disco rígido.

As equações desenvolvidas para o estudo de caso possibilitaram que o tamanho das mensagens trafegadas fosse reduzido significativamente. Com ambas as aplicações gerando pouco tráfego de rede, o desempenho do sistema de forma geral não é afetado quando os Agentes estão conectados ao Coordenador através da Internet. O tamanho das mensagens enviadas aos Agentes, por serem pequeno (16 bytes para o polinômio de grau 50 em x e em y), é fundamental para evitar que o Coordenador se torne o gargalo do sistema e sobrecarregar a rede. Enviando mensagens pequenas possibilita que o Coordenador atenda a um número maior de aplicações Agentes sem sobrecarregar a rede.

A aplicação Coordenador desenvolvida tem a característica de necessitar pouco poder de processamento para realizar a distribuição do processamento. No estudo de

caso realizado para se estimar os coeficientes do polinômio de grau 50 em x e y e distribuindo operações para 36 máquinas, a carga do processador estabilizou em pouco mais de 3% de uso. Deve-se considerar que havia outros aplicativos, que também consomiram tempo de processamento, sendo executados simultaneamente com Coordenador, inclusive o gerenciador do sistema que exibe a utilização do processador. Levando em consideração a configuração de máquina em que a aplicação Coordenador foi instalada, pode-se considerar que a aplicação necessita de poucos recursos para realizar a distribuição de processamento. Se for utilizada uma máquina mais potente, é possível distribuir processamento para um grande número de Agentes sem sobrecarregar a máquina que executa a aplicação Coordenador.

5.3. Artigos

Durante a realização do trabalho, com o objetivo de expor à comunidade científica o formalismo matemático desenvolvido e a metodologia proposta para implementação de um sistema de distribuição de processamento, foram submetidos quatro artigos a congressos, sendo três artigos para congressos internacionais e um artigo para congresso nacional. Os artigos estão listados a seguir.

- **Mathematical Formulation of a Model for Landform Attribute Representation for Application in Distributed Systems.** Publicado no 4th International Conference on Web Information Systems and Technologies (Webist 2008) [Bastos et al., 2008a]. O artigo apresenta a modelagem matemática desenvolvida para a representação de relevo por um polinômio com aplicação em Sistemas Distribuídos;
- **A Methodology for a Distributed Processing Using a Mathematical Model for Landform Attribute Representation.** Publicado no IADIS International Conference on Applied Computing 2008 [Bastos et al., 2008b]. Este artigo apresenta a metodologia para o desenvolvimento do sistema para distribuição do processamento através da Internet para estimar o polinômio que represente o relevo do estado de Minas Gerais. Porém, a metodologia apresentada neste arquivo ainda não abordava alguns aspectos relacionados com a computação voluntária;
- **A Model of Landform Attributes Representarion for Application in Distributed Systems.** Também publicado no IADIS International Conference on Applied Computing 2008 [Bastos et al., 2008c]. Este artigo exemplifica o uso

das equações desenvolvidas no primeiro artigo na distribuição de processamento, necessário para se estimar polinômios de alto grau;

- **Implementação de uma Metodologia para Processamento Distribuído com Aplicação de Representação do Relevo do Estado de Minas Gerais.** Publicado no Workshop de Teses e Dissertações ocorrido no IV Simpósio Brasileiro de Sistemas de Informação (SBSI 2008) [Bastos et al., 2008d]. Este artigo descreve toda a metodologia necessária para implementação de um sistema distribuído e mostrada como parte deste projeto de Dissertação de mestrado.

Além dos artigos publicados, pretende-se produzir os artigos conclusivos do projeto e um artigo extra. O primeiro artigo conclusivo seria sobre a metodologia desenvolvida, que sofreu modificações significativas desde as últimas publicações, por abordar aspectos relacionados à robustez e a computação voluntária. Um artigo sobre o sistema desenvolvido no estudo de caso para apresentar os resultados obtidos com a execução do sistema desenvolvido e apresentar os coeficientes encontrados. E um artigo sobre a formulação matemática, que foi demasiadamente simplificada, por questão de exigência de número máximo de páginas dos congressos, nos artigos publicados. Essa formulação matemática será generalizada para representar qualquer informação matricial.

5.4. Trabalhos Relacionados

Existem alguns trabalhos dedicados a simplificar o desenvolvimento de sistemas de distribuição de processamento utilizando computação voluntária, que ao contrário deste trabalho, apostam na simplificação através da reutilização de código. Um exemplo é o toolkit chamado Globus (http://www.globus.org/) que utiliza o conceito de Web Service para a distribuição de processamento. Apesar de ser uma ferramenta muito robusta, exige conhecimentos em diversas áreas como programação de *Web Services*, descrição de serviço, criação, gerenciamento e utilização de certificados, XML, etc. além do conhecimento sobre instalação, configuração e utilização da ferramenta, que não são tarefas triviais. Este fato pode restringir a utilização da ferramenta a grandes empresas e grupos de pesquisas consolidados e inviabilizar sua utilização para pequenos e médios projetos.

Além desta ferramenta, existe uma ferramenta denominada Berkeley Open *Infrastructure For Network Computing* - BOINC (http://boinc.berkeley.edu) que é semelhante à ferramenta Globus com a vantagem de possibilitar que aplicações clientes possam processar informações de diferentes projetos. No BOINC cada tarefa que é enviada é constituída de um conjunto de arquivos input e uma aplicação. Neste projeto o enviou de tarefas para os clientes é computacionalmente mais cara e menos segura já que envolve mobilidade de código.

Existe um projeto brasileiro chamado OurGrid (http://www.ourgrid.org/) cujo principal objetivo é simplificar o desenvolvimento deste tipo de sistema e possibilitar que pequenas empresas e pequenos projetos possam desenvolver seus sistemas de forma menos complexa. Este projeto utiliza o modelo de comunicação *Peer-to-Peer* onde não há uma aplicação que coordena a distribuição de processamento. Ao invés disto, cada aplicação da rede *Peer-to-Peer* é responsável por distribuir sua demanda de processamento para outros participantes da rede e ao mesmo tempo atender a demanda de processamento de outros projetos. Para utilizar esta ferramenta o projeto deve disponibilizar computadores para que outros projetos os utilizem. Isto é devido à política adotada pelo projeto OurGrid baseada em um sistema de créditos, onde o grupo (que pode ser um laboratório de computação de um projeto, uma rede de computadores de um projeto) que mais processa demanda de outros grupos acumula créditos. Estes créditos são utilizados quando este grupo necessitar de poder de processamento de outros grupos. Grupos com mais créditos são privilegiados no momento de requisitar poder de processamento dos demais grupos.

6. Conclusões e Trabalhos Futuros

A metodologia proposta se mostrou satisfatória em descrever aspectos importantes para o desenvolvimento de um sistema de distribuição de processamento utilizando computação voluntária. Não houve grandes dificuldades em adaptar a metodologia para o problema de estimativa de um polinômio para representação do relevo do estado de Minas Gerais, pois esta metodologia foi projetada com o intuito de ser facilmente utilizada e adaptada para solucionar problemas do tipo *Bag-of-Tasks*. Não houve necessidade de se criar novas classes de módulos, sendo todos os módulos propostos enquadrados em uma das três classes inicialmente propostas (módulos do núcleo, adaptáveis e opcionais). Com o estudo de caso realizado foi possível constatar que a metodologia é suficiente para servir como base para o desenvolvimento de aplicações que solucionam problemas do mundo real. Com o sistema desenvolvido e com base nos resultados obtidos, verifica-se que a metodologia cumpre o objetivo de possibilitar que sistemas de alto desempenho sejam desenvolvidos, dado a eficiência que a aplicação Coordenador distribuiu o processamento, sem sobrecarregar a máquina onde estava instalado.

A metodologia supera os principais desafios da área de Sistemas Distribuídos identificados neste trabalho: (a) o desafio da heterogeneidade das redes onde os computadores estarão localizados, heterogeneidade de hardware, sistema operacional e linguagem de programação utilizada para implementar a metodologia; (b) através da documentação da metodologia e arquitetura do sistema, o desafio da extensibilidade foi superado, possibilitando que o projeto possa ter extensão e modificação para a arquitetura; (c) com a utilização de criptografia, processamento redundante e ID de conexão foi possível oferecer um nível de segurança adequado para o sistema; (d) a partir do estado de cada operação registrada no status de cada operação no arquivo de operações, e outras técnicas desenvolvidas, obteve-se um nível de resistência a falhas significativo e; (e) além de contribuir para a escalabilidade, a comunicação com os Agentes realizado em paralelo e com a implementação de exclusão mútua, o desafio do controle de concorrência foi eliminado por completo.

Desafios específicos da área de *Volunteer Computing* também foram superados pela metodologia. No processo de restabelecer o estado anterior das operações enviadas para um determinado Agente, cujos resultados não foram retornados devido à finalização abrupta do Agente e, com a uso de um *timeout* ajustável do recebimento de

mensagens, foi possível superar os desafios da volatilidade que os voluntários entram e saem da sessão de processamento. Com o baixo *overhead* gerado na criação e destruição de *threads* para atender os Agentes dinamicamente, chegou-se a um sistema que se adapta, eficientemente, ao ambiente dinâmico da computação voluntária. A redundância de processamento foi a solução adotada para a falta de confiança entre as partes envolvidas na distribuição de processamento. Com esta redundância não se evita o recebimento de resultados incorretos, mas possibilita a identificação destes, sem sobrecarregar o sistema. A solução proposta para se obter um sistema escalável foi comentado anteriormente. Como os testes foram realizados em rede interna, não houve a necessidade de implantar uma política de incentivo a participação voluntária no projeto. Contudo, acreditamos que a divulgação do projeto a voluntários e grupos de pesquisa que têm interesse nos resultados é a melhor forma de estimular a participação voluntária, apontada como um desafio da área.

Como o principal objetivo da metodologia desenvolvida neste trabalho é facilitar o desenvolvimento de sistemas distribuídos, trabalhos futuros relacionados poderão completá-la e complementá-la, entre os quais sugerimos:

- dar continuidade ao projeto desenvolvido no estudo de caso para possibilitar o uso de computação voluntária. Para isto, deve ser criada uma metodologia de divulgação do projeto através de sites, por exemplo, emailsa etc., para que voluntários possam ser incentivados a colaborar com o projeto;
- utilizar o sistema desenvolvido para estimar polinômios de grau maior que 50 pois, graus elevados darão melhor fidelidade da representação do relevo porém demandam maior poder computacional, que poderia ser atendido com o auxílio de voluntários;
- depois de estimado um polinômio de certo grau que significasse uma representação satisfatória do relevo do estado de Minas Gerais, seria possível utilizar funcionalidades matemáticas sobre o polinômio para obter informações ricas sobre o relevo como pontos máximos e mínimos, cálculo de escoamento de água, cálculos de visibilidade, etc.;
- com pequenas modificações do sistema desenvolvido no estudo de caso, seria possível representar não só o relevo de outras áreas (outros estados ou países), como também outras informações matriciais;
- para a representação de relevo, poderia ser utilizado um outro MDE, de outro projeto, cujas medições foram feitas a cada 90 metros, enquanto o MDE do

projeto GTOPO30 foi feito a cada 900 metros. Esta mudança possibilitaria uma representação ainda mais representativa do relevo de uma determinada área;

- acreditamos que a metodologia proposta, além de auxiliar do desenvolvimento de sistemas sem ajuda de middlewares, também poderia ser útil caso o sistema para distribuição de processamento fosse desenvolvido utilizando alguma ferramenta como Globus ou BOINC. Neste caso, não seria necessário desenvolver algumas funcionalidades, já implementadas nas ferramentas, porém a metodologia proposta auxiliaria na forma como a ferramenta seria utilizada. Desta forma, sistemas poderiam ser desenvolvidos utilizando uma ferramenta disponível em conjunto com a metodologia proposta;
- um trabalho futuro significativo seria a extensão desta metodologia para possibilitar o desenvolvimento da aplicação Coordenador ainda mais resistente a falhas. Para isto seria descrito na metodologia o desenvolvimento de Coordenadores auxiliares, ou o desenvolvimento de réplicas de Coordenadores para impedir que a falha de um Coordenador comprometesse todo o sistema;
- aplicar as equações matemáticas desenvolvidas para representar qualquer tipo de imagem por um polinômio e comparar a eficiência em tempo e espaço para transmissão desta imagem. O modelo matemático apresentado pode ser utilizado para representar, via polinômio, qualquer informação que estiver em forma matricial.

Referencias Bibliográficas

ACM. ACM Computing Classification System. Disponível em: <http://www.acm.org/about/class/ccs98-html>. Acessado em: 20 jun. 2008.

GLADMAN, Brian. AES and Combined Encryption/Authentication Modes. Brian Gladman's Home Page. Disponível em: <http://fp.gladman.plus.com/AES/>. Acessado em: 13 jun. 2008.

ANDERSON, D. P. Public Computing: Reconnecting People to Science. In: CONFERENCE ON SHARED KNOWLEDGE AND THE WEB. 2004a. Madri, Espanha.

ANDERSON, D. P. BOINC: A System for Public-Resource Computing and Storage. In: 5TH IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING. 2004b. Pittsburgh, Pennsylvania, E.U.A, p. 365- 372.

ANDERSON, D. P.; COBB, J.; KORPELA, E.; LEBOFSKY, M. & WERTHIMER, D. SETI@home An Experiment in Public-Resource Computing., **Communication of the ACM,** ACM. 2002. No. 11.

ANDERSON, D. P. & WALTON, R. High-Performance Task Distribution for Volunteer Computing. In: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON E-SCIENCE AND GRID COMPUTING. 2005. IEEE Computer Society, Melbourne, Australia.

ANDREWS, G. R. Paradigms for Process interaction in Distributed Programs. **ACM Computing Surveys**. 1991. vol. 23. p. 49 - 90.

ARGAEZ, Enrique de. Internet World Stats. Disponível em: <http://www.internetworldstats.com/stats.htm>. Acessado em: 20 maio 2008.

BAJAJ, C.; IHM, I. & WARREN, I. Higher-Order Interpolation and Least-Squares Approximation Using Implicit Algebraic Surfaces. **ACM Transactions on Graphics**. 1993. vol. 12, p. 327-347.

BASTOS, L. N.; ABRANTES, R. P. & LEAL, B. G. A Mathematical Formulation of a Model for Landform Attributes Representation for Application in Distributed Systems. In: WEB INFORMATION SYSTEMS AND TECHNOLOGIES. 2008A. Funchal, Madeira, Portugal, p. 259.

BASTOS, L. N.; ABRANTES, R. P. & LEAL, B. G. (2008b), A Methodology for a Distributed Processing Using a Mathematical Model for Landform Attribute Representation. In: PROCEEDINGS OF IADIS INTERNATIONAL CONFERENCE ON APLLIED COMPUTING 2008. 2008B. p. 441-445.

BASTOS, L. N.; ABRANTES, R. P. & LEAL, B. G. A Model of Landform Attributes Representation for Application in Distributed Systems. In: PROCEEDINGS OF IADIS INTERNATIONAL CONFERENCE ON APLLIED COMPUTING 2008. 2008b. p. 460-465.

BASTOS, L. N.; ABRANTES, R. P., LEAL, B. G. & Goulart, C. C. (2008d), Implementação de uma Metodologia para Processamento Distribuído com Aplicação de Representação do Relevo do Estado de Minas Gerais. In: WORKSHOP DE TESES E DISSERTAÇÕES 2008. 2008c. Rio de Janeiro, Brasil, p. 326-331.

BOLTON, F. **Pure Corba**. E.U.A. Sams, 2001.

BRAMAN, S. Transformations of the Reserach Enterprise. EDUCAUSE. 2006.

CÂMARA, Gilberto; DAVIS, Clodoveu; MONTEIRO, Antônio Miguel Vieira. Introdução à Ciência da Geoinformação. Cap. 7. Instituto Nacional de Pesquisas Espaciais. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/>. Acessado em: 12 maio 2008.

CÂMARA, G.; DAVIS, C. & MONTEIRO, A. M. V. **Introdução à Ciência da Geoinformação**. Instituto Nacional de Pesquisas Espaciais, São José dos Campos, São Paulo. Brasil. 2001.

CHOI, S.; KIM, H.; BYUN, E.; BAIK, M.; KIM, S.; PARK, C. & HWANG, C. Characterizing and Classifying Desktop Grid. In: SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID. 2007. Rio de Janeiro, Brasil, p. 743-748.

CHRISTENSEN, C.; AINA, T. & STAINFORTH, D. The Challenge of Volunteer Computing With Lengthy Climate Model Simulations. In: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON E-SCIENCE AND GRID COMPUTING. 2005. p. 8.-.

COULOURIS, G.; DOLLIMORE, J. & KINDBERG, T.WESLEY, A. **Distributed System: Concepts and Design**. E.U.A. Addison Wesley. 2005.

DOWNING, T. B. **Java RMI: Remote Method Invocation**. Wiley Publishing. 1998.

FELGUEIRAS, C. A. & CÂMARA, G. (2001), **Introdução à Ciência da Geoinformação.** Instituto Nacional de Pesquisas Espaciais. 2001. cap 7. p. 172-209.

FOLDING@HOME. Stanford University. http://folding.stanford.edu/

FOWLER, D. A supercomputer in every home?. NetNews. 2005. vol. 9, p. 5 - 8.

GEEK. PS3 entra para o Guinness. Disponível em: <http://www.geek.com.br/modules/noticias/ver.php?id=14986&sec=7>. Acessado em: 23 jun. 2008.

GOVINDARAJU, M.; SLOMINSKI, A.; CHIU, K.; LIU, P.; VAN ENGELEN, R. & LEWIS, M. J. Toward Characterizing the Performance of SOAP Toolkits. In: FIFTH IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING. 2004. P. 365-372.

GOVINDARAJU, M.; SLOMINSKI, A.; CHOPPELLA, V.; BRAMLEY, R. & GANNON, D. Requirements for and Evaluation of RMI Protocols for Scientific

Computing. In: PROCEEDINGS OF THE 2000 ACM/IEEE CONFERENCE ON SUPERCOMPUTING. 2000. p. 61.

GTOPO30. U.S. Geological Survey. Disponsível em: <http://edc.usgs.gov/products/elevation/gtopo30/gtopo30.html>. Acessando em 03 jul. 2008.

HARIRI, S. E PARASHAR, M. **Tools and Environments for Parallel and Distributed Computing. Wiley Series on Parallel and Distributed Computing**. New Jersey, E.U.A., John Wiley & Sons. 2004.

IAIK. AES Lounge. Disponível em: <http://www.iaik.tu-graz.ac.at/research/krypto/AES/>. Acessado em: 19 jun. 2008.

INTERNET SYSTEMS CONSORTIUM. Internet Domain Survey. Disponível em: <http://www.isc.org/index.pl?/ops/ds/>. Acessado em: 19 maio 2008.

KONDO, D.; TAUFER, M.; BROOKS, C. L.; CASANOVA, H. & CHIEN, A. A. Characterizing and Evaluating Desktop Grids: An Empirical Study. In: PROCEEDINGS OF THE 18TH INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM. 2004. p. 26-.

KOTSOVINOS, E. **Global Public Computing**. Technical report. University of Cambridge. 2005.

LARSON, S. M.; SNOW, C. D.; SHIRTS, M. & PANDE, V. S. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. Modern Methods em Computational Biology. Horizon Press. 2003.

LOOSEMORE, S.; STALLMAN, R. M.; MCGRATH, R.; ORAM, A. & DREPPER, U. **The GNU C Library Reference Manual**. Free Software Foundation. 2007.

LYMAN, P.; VARIAN, H. R. How Much Information 2003?. Disponível em: <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/>. Acessado em 30 jun. 2008.

MAHESWARAN, M.; MANIYMARAN, B.; ASADUZZAMAN, S. & MITRA, A. Towards a Quality of Service Aware Public Computing Utility. In: PROCEEDINGS OF THE THIRD IEEE INTERNATIONAL SYMPOSIUM ON NETWORK COMPUTING AND APPLICATIONS. 2004. p. 376- 379.

MATTMANN, C. A.; CRICHTON, D. J.; MEDVIDOVIC, N. & HUGHES, S. A Software Architecture-Based Framework for Highly Distributed and Data Intensive Scientific Applications. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. 2006. Shanghai, China, p. 721 - 730.

NAMIKAWA, L. M. (1995) Um método de ajuste de superfície para grades triangulares considerando linhas características. Tese de Mestrado, Instituto Nacional de Pesquisas Espaciais. 1995.

NAMIKAWA, L. M.; FELGUEIRAS, C. A.; MURA, J.; ROSIM, S. & LOPES, E. S. S.

Modelagem Numérica de Terreno e Aplicações. Instituto Nacional de Pesquisas Espaciais. São José dos Campos, Brasil. 2003.

NELSON, B. J. Remote procedure call. Carnegie Mellon University. Pittsburgh, E.U.A. 1981.

NIST. National Institute of Standards and Technology. http://www.nist.gov/

NOBLE, B. & DANIEL, J. W. **Álgebra Linear Aplicada**, Rio de Janeiro, Brazil. Prentice/Hall do Brasil. 1986.

QUEIROZ, C.; NETTO, M. A. S. & BUYYA, R. Message Passing over Windowsbased Desktop Grids. In: 4TH INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING. ACM. 2006. Melbourne, Austrália.

RAWLINGS, J. O.; PANTULA, S. G. & DICKEY, D. A. **Applied Regression Analysis: A Research Tool**. Springer-Verlag. 1998.

REDMOND, F. E. **Dcom: Microsoft Distributed Component Object Model.** IDG Books Worldwide. 1997.

RMI - Remote Method Invocation Home. Sun Developer Network. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

SALTZER, J. H.; REED, D. P. & CLARK, D. D. End-to-end arguments in system design. In: ACM TRANSACTIONS ON COMPUTER SYSTEMS. 1984. Vol. 2. p. 277-288.

SCHOLLMEIER, R. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING. 2002. IEEE.

SEBER, G. A. F. & WILD, C. J. (1989), *Nonlinear Regression*, E.U.A. Wiley, 1989.

SINHA, A. Client-Server Computing. Comunication of ACM. 1992. vol. 35.

STEINMETZ, R. & WEHRLE, K. Peer-to-Peer Systems and Applications. 2005. vol. 3485. Springer-Verlag.

STOCKINGER, H. Defining the grid: a snapshot on the current view. The Journal of Supercomputing. Springer Netherlands. 2007. vol. 42, 3-17.

STOKES, J. Understanding Moore's Law. Ars Technica. Disponível em: <http://arstechnica.com/articles/paedia/cpu/moore.ars/6>. Acessado em: 20 maio 2008.

TANENBAUM, A. S. **Computer Networks**. Prentice Hall. 2003a.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. Prentice-Hall. 2003b.

TANENBAUM, A. S., STEEN, M. VAN. **Distributed Systems: Principles and Paradigms**. Prentice Hall. 2002.

TAUFER, M.; AN, C.; KERSTENS, A. & III, C. B. Predictor@Home: A "Protein Structure Prediction Supercomputer" Based on Public-Resource Computing. In: 19TH IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM. 2005. Workshop 7. p. 200b.

TAUFER, M.; KERSTENS, A.; ESTRADA, T. P.; FLORES, D. A.; ZAMUDIO, R.; TELLER, P. J.; ARMEN, R. & BROOKS, C. L. Moving Volunteer Computing towards Knowledge-Constructed, Dynamically-Adaptive Modeling and Scheduling. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM. 2007. p. 478.

W3SCHOOLS. (2008). OS Platform Statistics. http://www.w3schools.com/browsers/browsers_os.asp

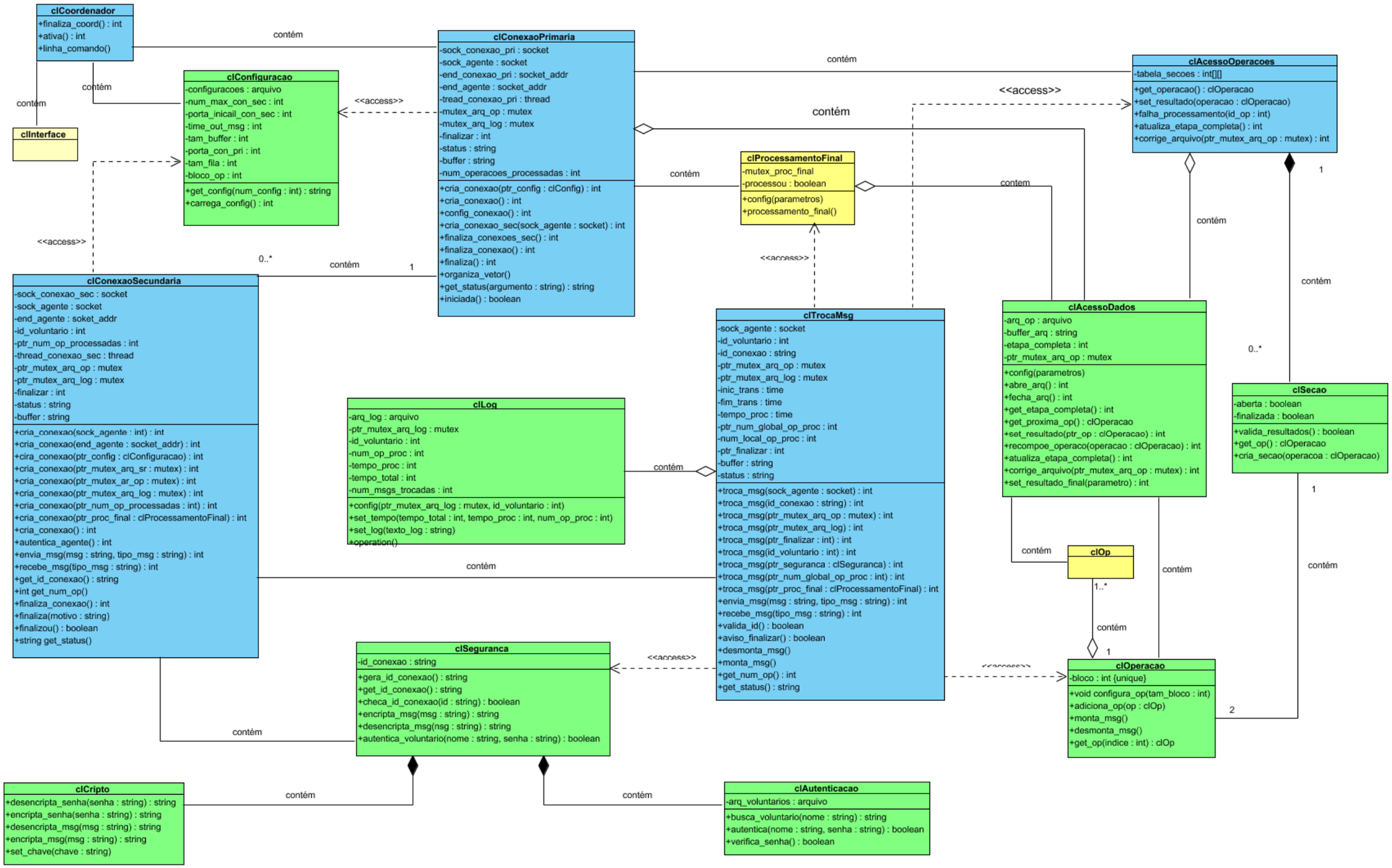
WIKIPEDIA (2008a). Application Programming Interface. Disponível em: <http://en.wikipedia.org/wiki/API>. Acessado em: 20 jun. 2008.

WIKIPEDIA. (2008b). Delaunay Triangulation. Disponível em: <http://en.wikipedia.org/wiki/Delaunay_triangulation>. Acessado em: 20 jun. 2008.

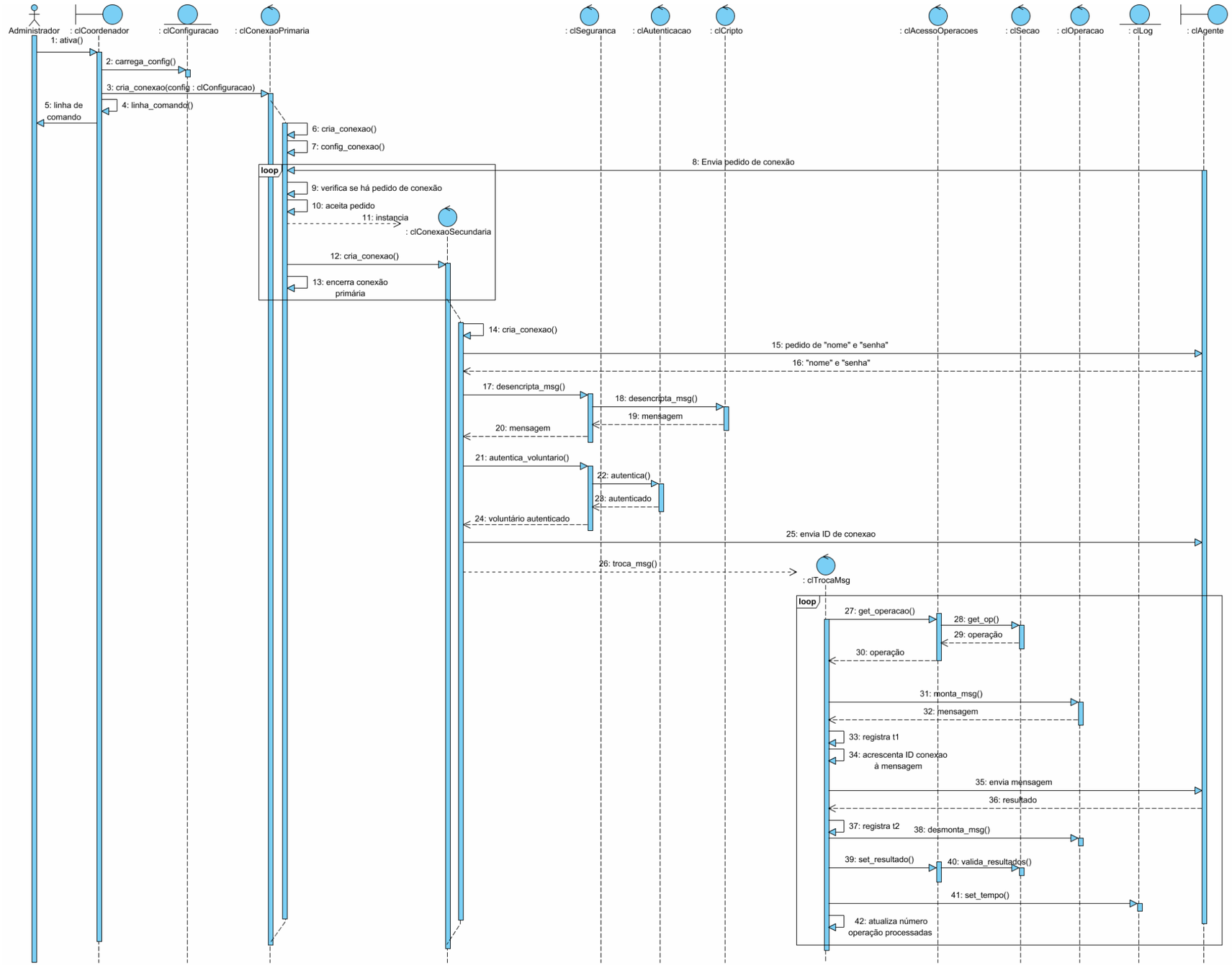
ZAKON, R. H. (2006). Hobbes' Internet Timeline. [Internet]. Disponível em: <http://www.zakon.org/robert/internet/timeline/#Growth>. Acessado em: 8 jun. 2008.

A. Diagramas do Modelo Genérico de Distribuição de Processamento

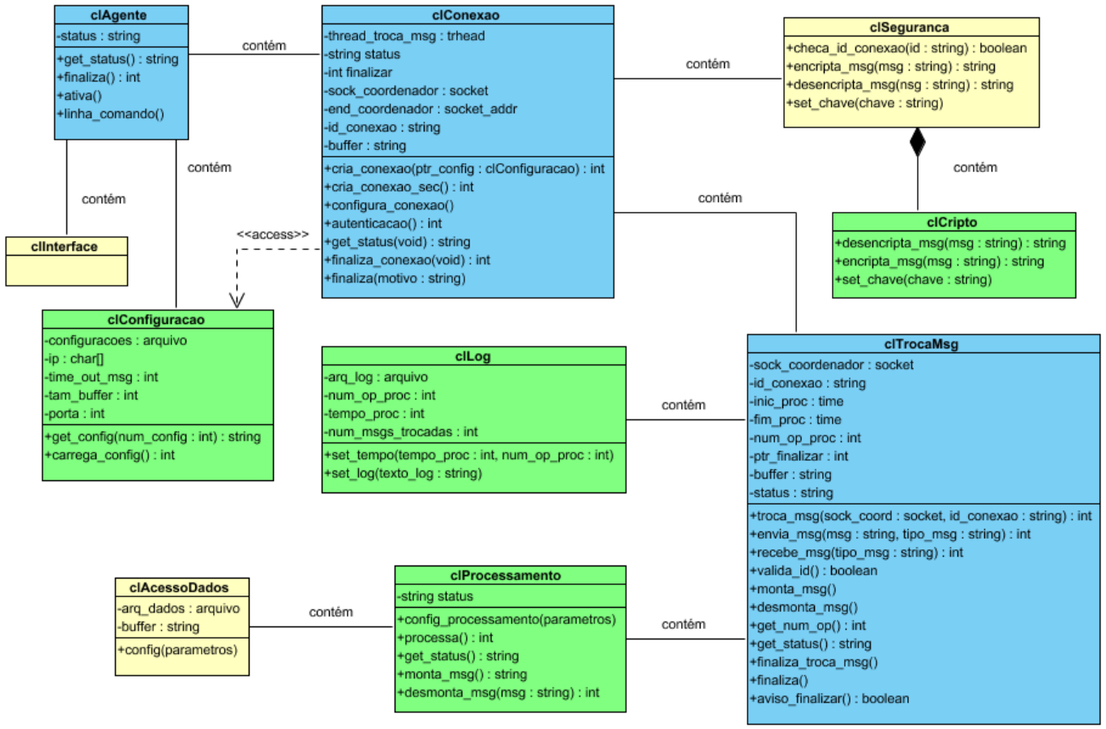
A.1. Diagrama de Classe – Aplicação Coordenador (Os parâmetros da função *cria_conexao()* do módulo clConexaoSecundaria e da função *troca_msg()* da classe clTrocaMsg foram divididas em diversas linhas por serem vários parâmetros. Na realidade, ao invés de existirem diversas funções com o mesmo nome, existe apenas uma função cujos parâmetros são apresentados nas demais funções de mesmo nome)



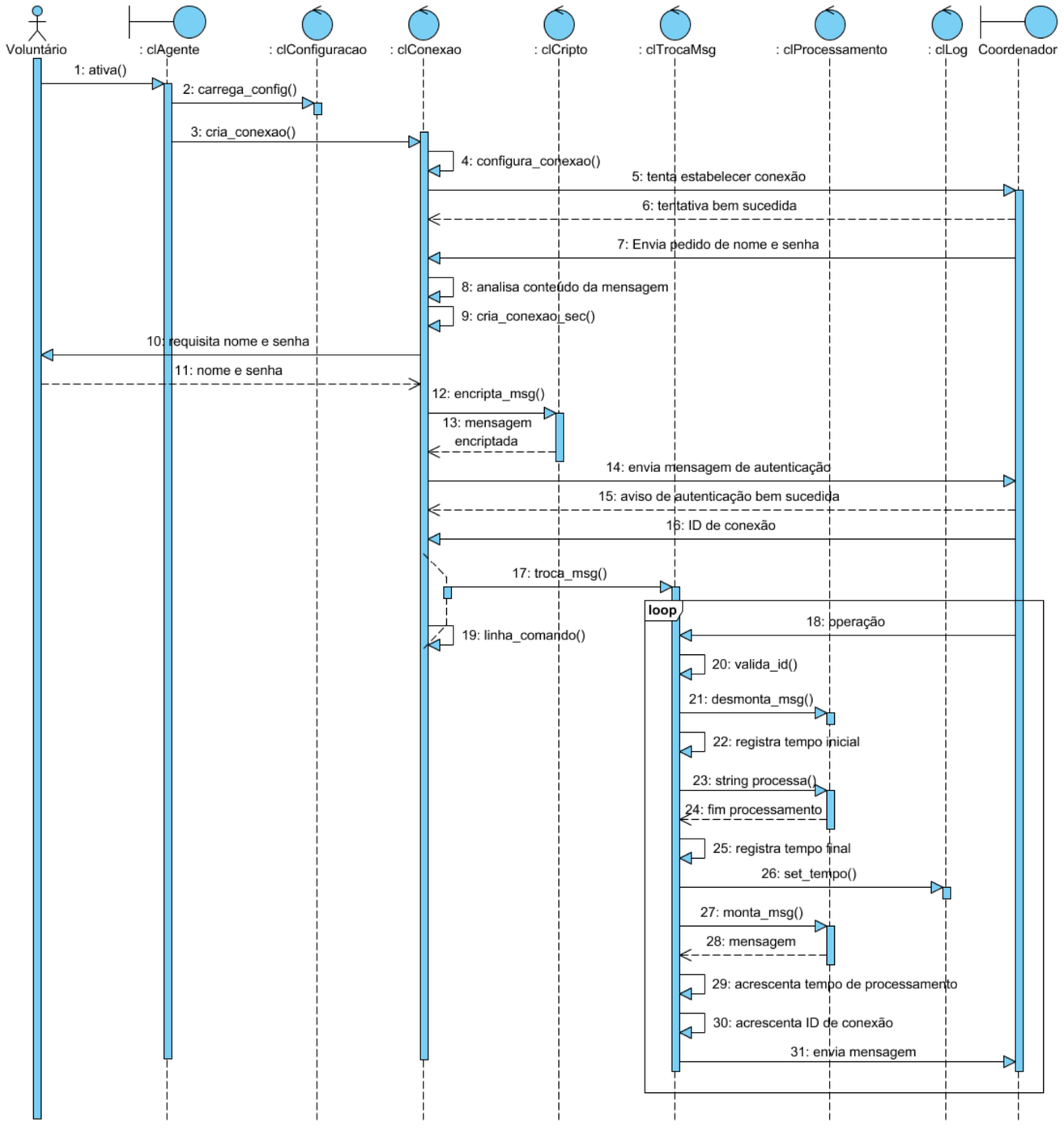
A.2. Diagrama de Seqüência que ilustra o fluxo principal de execução do Coordenador e a interação entre seus módulos. O fluxo principal apresentado aqui corresponde aos estados representados nos diversos Diagramas de Estados pela cor azul.



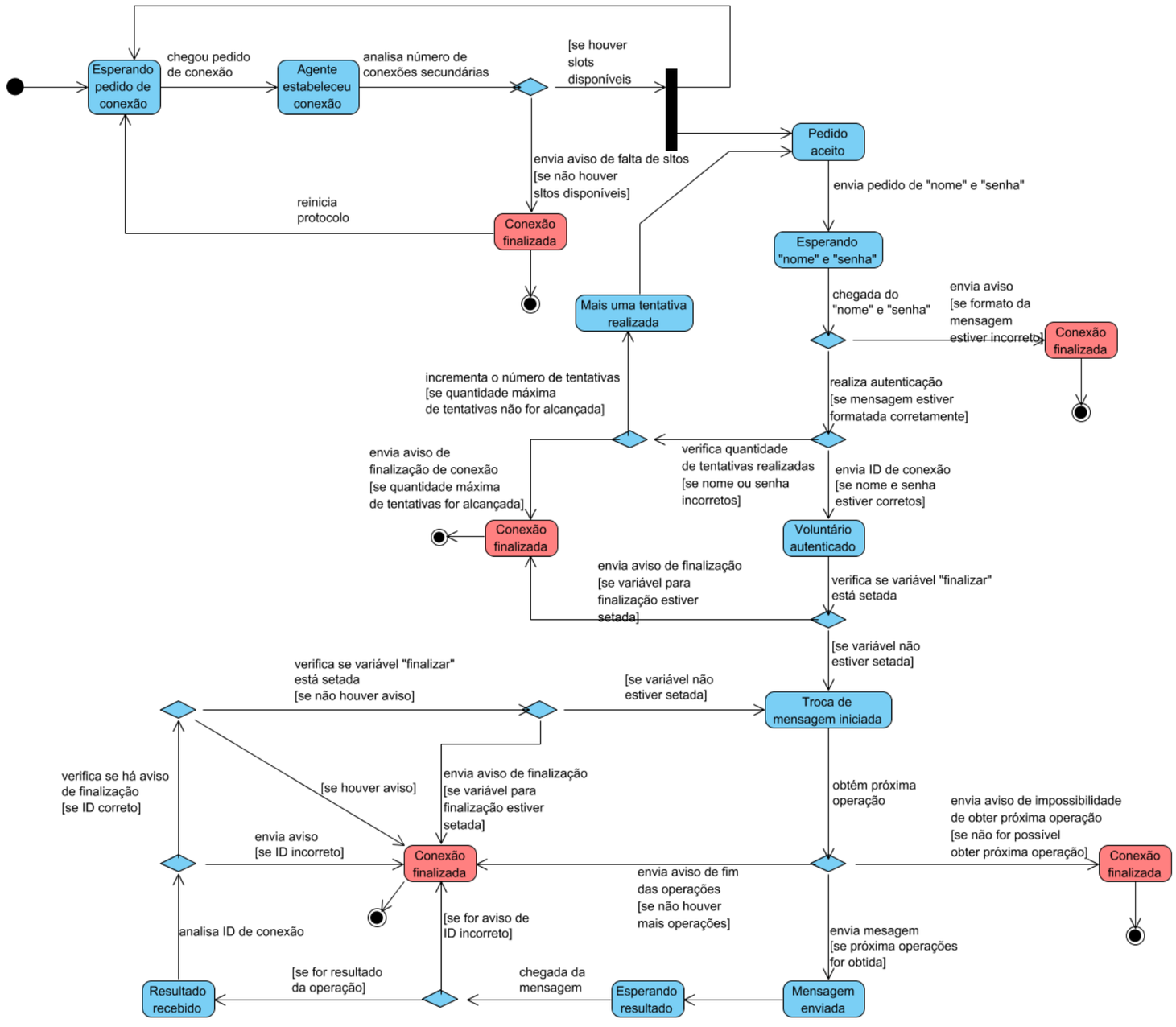
A.3. Diagrama de Classe – Aplicação Agente.



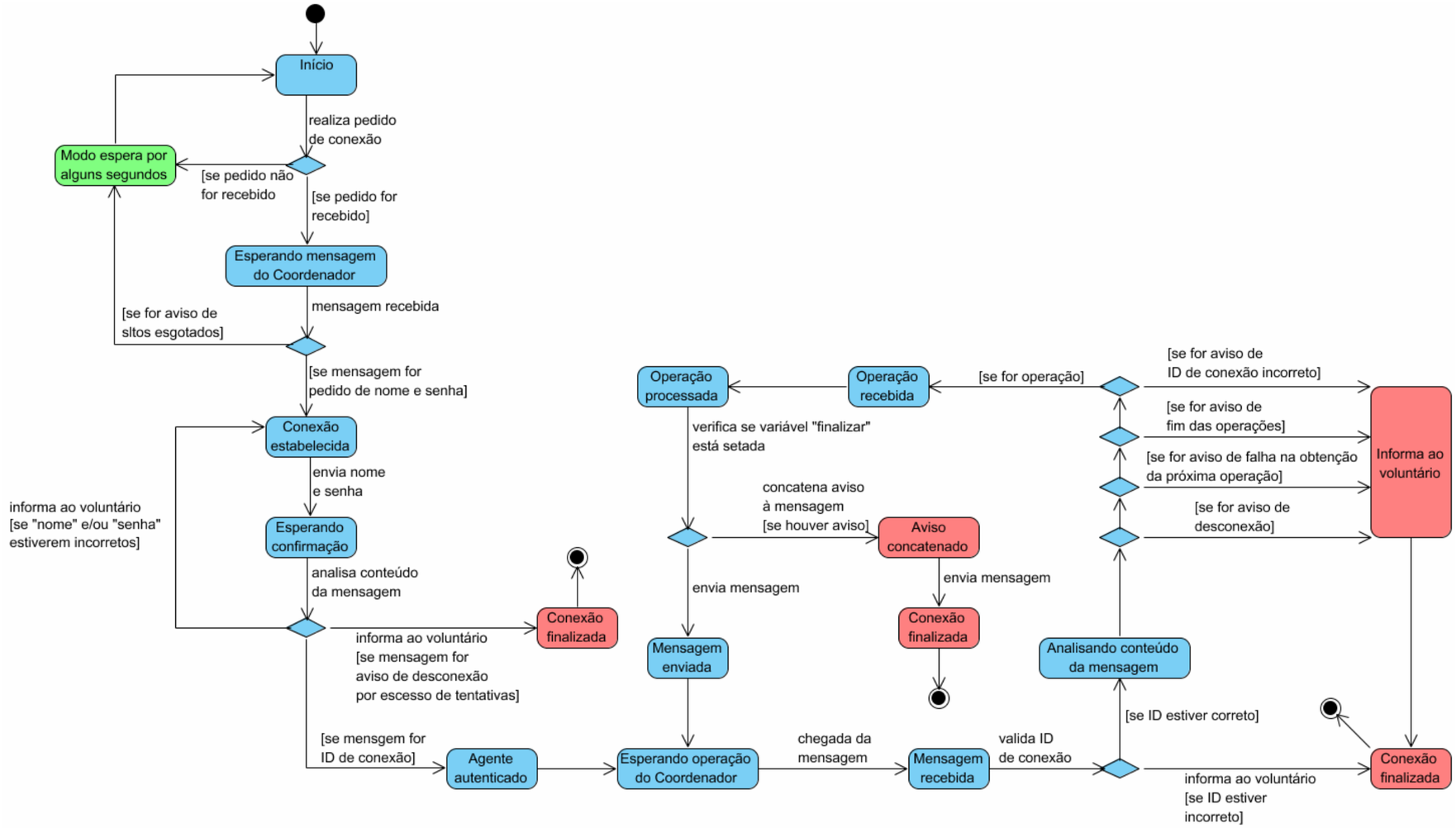
A.4. Diagrama de Seqüência que ilustra o fluxo principal de execução do Agente e a interação entre seus módulos. O fluxo principal apresentado aqui corresponde aos estados representados nos diversos Diagramas de Estados pela cor azul.



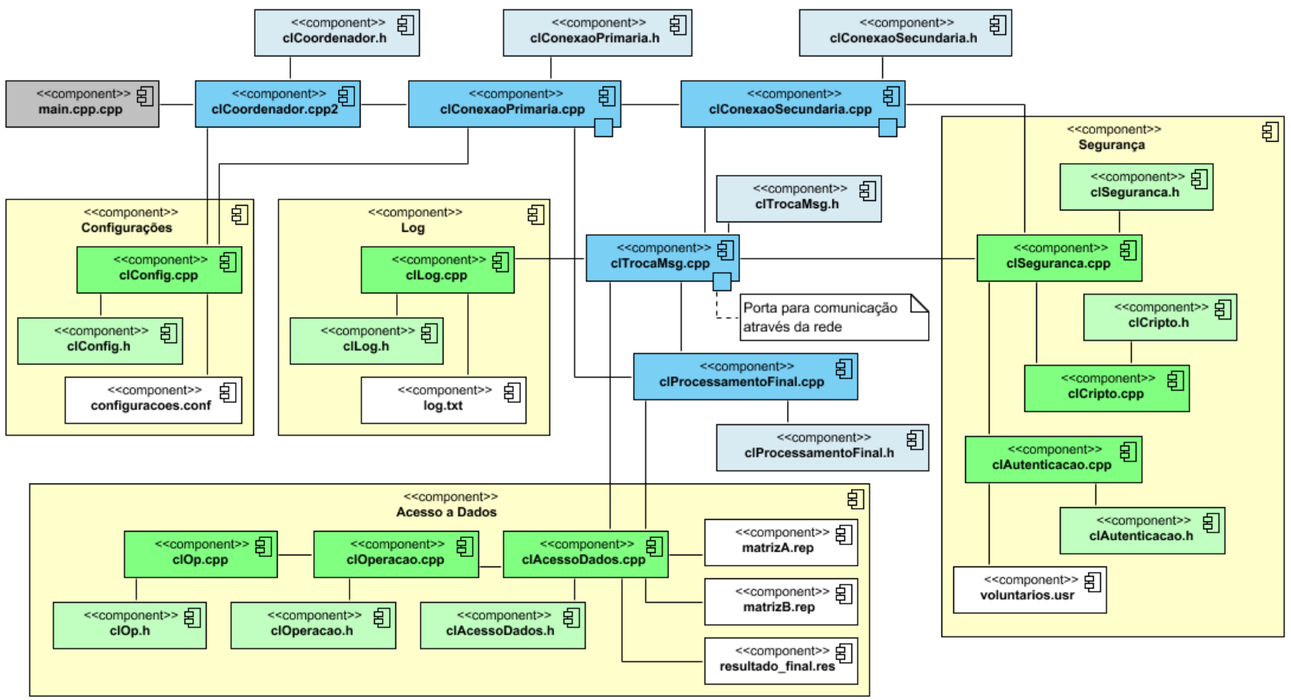
A.5. Diagrama de Estado que descreve o protocolo de comunicação para aplicação Coordenador e que representa os estados assumidos por uma aplicação Coordenador, para que ela possa se comunicar corretamente com outra aplicação Agente.



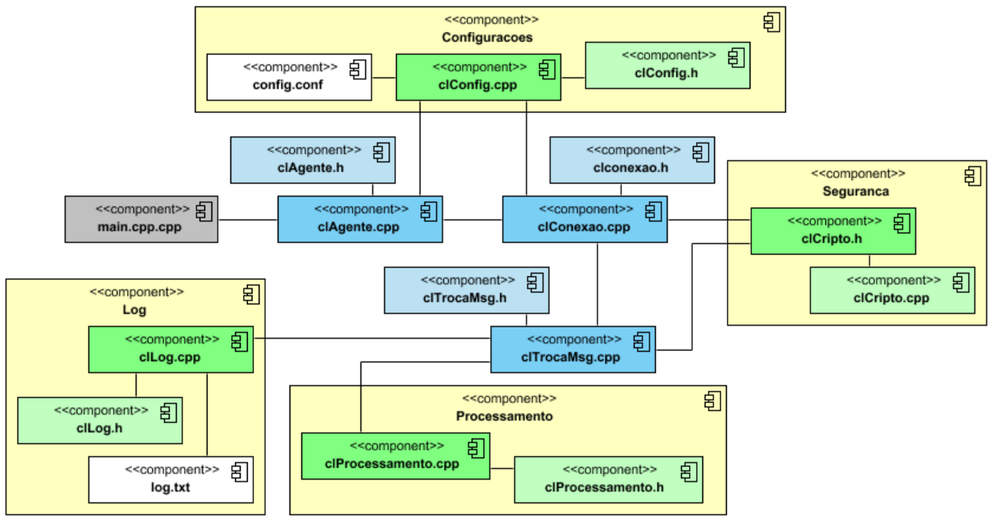
A.6. Diagrama de Estado que descreve o protocolo de comunicação para aplicação Agente e que representa os estados assumidos por uma aplicação Agente, para que ela possa se comunicar corretamente com uma aplicação Coordenador.



A.7. Diagrama de Componentes da aplicação Coordenador



A.8. Diagrama de Componentes da aplicação Agente



A. 9 – Diagrama de Execução do sistema desenvolvido no estudo de caso.