Gustavo Vasconcellos Cavalcante

Projeto e Implementação de um Serviço de Licenciamento CORBA

Campina Grande - PB
Agosto de 1999

Gustavo Vasconcellos Cavalcante

Projeto e Implementação de um Serviço de Licenciamento CORBA

Dissertação apresentada à Coordenação de Pós-graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para a obtenção do Título de Mestre em Informática.

Orientador:Francisco Vilar Brasileiro

Campina Grande - PB

Agosto de 1999

C376p Cavalcante, Gustavo Vasconcellos
Projeto e implementacao de um servico de licenciamento CORBA / Gustavo Vasconcellos Cavalcante. - Campina Grande, 1999.
76 f.

Dissertaca (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Arquitetura de Computadores 2. Tolerancia a Falhas 3. Licenciamento 4. CORBA 5. Sistemas Distribuidos 6. Dissertacao - Informatica I. Brasileiro, Francisco Vilar II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.2(043)

PROJETO E IMPLEMENTAÇÃO DE UM SERVIÇO DE LICENCIAMENTO CORBA

GUSTAVO VASCONCELLOS CAVALCANTE

DISSERTAÇÃO APROVADA EM 18.08.1999

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador

PROF. JOSÉ ANTÃO BELTRÃO MOURA, Ph.D
Examinador

PROF. ARTURO HERNÁNDEZ DOMÍNGUEZ, Dr.
Examinador

CAMPINA GRANDE – PB

Dedico este trabalho aos
meus pais, Radjalma e Tania,
que me ensinaram o valor
do aprendizado contínuo.

Caminante, son tus huellas
el camino, y nada más;
caminante, no hay camino,
se hace camino al andar.
Al andar se hace camino,
y al volver la vista atrás
se ve la senda que nunca
se ha de volver a pisar.
Caminante, no hay camino,
sino estelas en la mar.

Antonio Machado
(Poeta Espanhol , 1875 - 1939)

Resumo

Este trabalho apresenta o projeto e a implementação de um Serviço de Licenciamento CORBA. CORBA é um padrão para uma arquitetura de desenvolvimento de aplicações distribuídas. O Serviço de Licenciamento controla o uso das aplicações segundo os termos contidos na licença. O projeto do nosso sistema enfatiza o aspecto de tolerância a falhas. Apresentamos nesta dissertação algumas técnicas que visam manter a continuidade do sistema, mesmo na ocorrência de falhas em seus componentes.

Abstract

This work presents the project and implementation of a CORBA Licensing Service. CORBA is a standard for an architecture for the development of distributed applications. The Licensing Service controls the use of applications in terms of a license. The focus of our project is fault tolerance. We present in this work some techniques to improve the system functioning even in the presence of faults on some components of the system.

Agradecimentos

A Deus por tudo.

Aos meus pais e a toda a minha família pelo amor, incentivo e ajuda em todas as fases de minha vida.

A Ivette pela força, carinho e motivação que me ajudaram a concluir esse trabalho.

Ao meu orientador, Prof. Francisco Brasileiro, pela confiança e incentivo.

Aos meus companheiros do LSD (Laboratório de Sistemas Distribuídos): Lívia, Érica, Fellipe, Tatiana, Luiza, Guga e Milton; pela amizade.

Aos funcionários da COPIN (Aninha e Vera) pelo seu excelente trabalho.

Aos funcionários da miniblio (Zeneide, Manuela e Josirene) e da Cantina (Romildo, Josenilda e D. Inês).

Aos amigos Daniel, Eloi, Guto, Hilmer e Tarig.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1 - Introdução

1.1 Motivação

A indústria da informática é atualmente um dos setores que mais recursos movimenta. Muito dinheiro é investido em pesquisa e desenvolvimento de novos produtos e grande parte dos rendimentos deste mercado é perdida com a pirataria de software.

Ao comprar um determinado software, na verdade está se comprando o direito ao uso deste. A licença é o contrato que regulamenta as condições do uso do software. A pirataria pode ser definida como o uso ou cópia ilegal do software de forma a violar esta licença.

A lei é clara: pirataria é crime. O Brasil inclui-se entre os países que possuem legislação específica de proteção à indústria do software. Segundo a Lei nº 9609/98 de 20 de fevereiro de 1998, os programas de computador ficam incluídos no âmbito dos direitos autorais, sendo proibidas a reprodução, a cópia, o aluguel e a utilização de cópias de programas de computador feitas sem a devida autorização do titular dos direitos autorais.

A legislação de software estabelece que a violação destes direitos é passível de ação criminal e de ação cível de indenização. O infrator fica sujeito a detenção de 6 meses a 2 anos e multas diárias pelo uso ilegal dos programas. Combinada com a Lei do Direito Autoral, a Lei de Software permite que as perdas e danos do titular do programa sejam ressarcidos pelo valor equivalente a 3.000 cópias de cada software ilegalmente produzido. Caso a infração seja feita com o intuito de comercialização, a pena passa a ser de reclusão de 1 a 4 anos.

Segundo dados da *Business Software Alliance* [BSA99], os prejuízos causados pela pirataria de software no mundo foram estimados em 11 bilhões de dólares em 1998. Este estudo mostra que quatro entre dez aplicativos comerciais usados no mundo

são cópias ilegais. No Brasil, é estimado que 61% dos softwares em uso no país são piratas.

A primeira vista, é possível pensar que a pirataria causa danos apenas aos produtores de software. Na verdade, os danos são muito maiores, causando um grande impacto na criação de empregos, salários, impostos e rendimentos das revendas.

Com o crescimento da redes de computadores e dos aplicativos executando em rede, é cada vez mais difícil o controle do uso do software segundo os termos contidos na licença. Um aplicativo instalado em uma máquina pode ser utilizado por diversos usuários, em diferentes máquinas, facilitando assim a pirataria.

Um grande número de vezes, o responsável pela pirataria é o comprador de uma cópia legal. No caso de um usuário doméstico, essa cópia é instalada em vários computadores, emprestada para os amigos etc. Em uma empresa, além das instalações ilegais, essa cópia é colocada na rede, ficando a disposição do uso de qualquer usuário desta.

Diante destes fatos, visando proteger seu investimento e sua propriedade intelectual, os desenvolvedores de software passaram a tomar atitudes para evitar a prática da pirataria. A forma mais comum de se realizar este controle é através do serviço de licenciamento, onde cada licença para o uso do software é fornecida por um meio técnico (baseado em mecanismos de hardware ou software), ou através de uma forma contratual (através da adoção de sanções legais ao usuário de uma cópia não autorizada).

A forma contratual isoladamente não tem se mostrado uma opção viável. Devido a isto, a utilização de meios técnicos para controle do uso do software são geralmente utilizados para complementar os contratos legais. O uso destes mecanismos possibilitam um controle mais efetivo do uso do software.

Uma das formas de se implementar este controle é através do uso do serviço de gerenciamento de licenças. Este serviço consiste em um sistema distribuído que controla a utilização de produtos de software segundo os termos contidos na licença.

1.1.1 CORBA e Serviço de Licenciamento

Uma característica importante das grandes redes de computadores tais como a Internet e as *intranets* das corporações, é o fato de elas formarem um ambiente heterogêneo de rede. Por exemplo, uma *intranet* pode ser formada de *mainframes*, estações e servidores UNIX e vários PCs utilizando Windows. Esta *intranet* poderá ter em sua rede vários protocolos de redes, tais como: ATM, Ethernet, FDDI, etc. O rápido crescimento destas redes tem como causa a necessidade de compartilhar informações entre todos os setores da corporação.

Infelizmente, lidar com essa heterogeneidade nos sistemas distribuídos não é tarefa fácil. Em particular, é difícil desenvolver aplicações que façam uso eficiente do ambiente de rede heterogêneo. Existem vários pacotes de software que auxiliam o desenvolvimento de sistemas em uma plataforma homogênea de rede. Entretanto, poucos lidam com a integração de sistemas desenvolvidos separadamente em um ambiente distribuído heterogêneo.

CORBA (*Common Object Request Broker Architecture)* é um padrão de arquitetura que permite a comunicação de aplicações independentemente das arquiteturas de computadores (hardware), sistemas operacionais e linguagens de programação (software), sobre as quais elas executam. A comunicação entre as aplicações é realizada através de chamadas a métodos de objetos.

Alem disso, CORBA torna a rede transparente para o programador, facilitando assim o desenvolvimento de aplicações distribuídas. Com isso, a chamada de um método em um objeto situado em uma máquina remota é feita de forma análoga à de um objeto situado em uma máquina local.

Corba padroniza uma série de serviços para auxiliar o desenvolvimento de aplicações distribuídas. Entre estes serviços encontra-se o serviço de licenciamento.

A especificação do Serviço de Licenciamento CORBA define as interfaces dos objetos que os produtores de software podem utilizar para interagir com o sistema de licenciamento.

1.1.2 Problemas que Podem Ocorrer no Serviço de Licenciamento

No serviço de licenciamento podem ocorrer comportamentos indesejáveis em vários de seus componentes. É necessário que o Serviço de Licenciamento seja capaz de se recuperar e continuar seu funcionamento, mesmo na presença destes problemas.

Entre estes comportamentos indesejáveis encontram-se: falhas nos clientes (o software que está sendo licenciado), falhas nos produtores (O elemento responsável por efetuar este controle), particionamento da rede da rede de comunicação, etc.

Existe um custo a ser pago pela inclusão de técnicas de tolerância a falhas em sistema. Este custo pode ser: financeiro, degradação do desempenho do sistema, etc. No projeto do gerenciador de licenças deve ser estabelecido qual o custo que se deseja pagar para aumentar a robustez do sistema.

1.2 Objetivo do Trabalho

O nosso trabalho teve como objetivo o projeto e a implementação de um Serviço de Licenciamento CORBA. Neste projeto o aspecto enfocado foi o de tolerância a falhas. O Serviço de Licenciamento sugerido é capaz de se recuperar de vários tipos de falhas que podem ocorrer no sistema. Entre os possíveis problemas encontram-se: falhas nos servidores, falhas nos clientes e problemas com a rede de comunicação.

No projeto do sistema, tentou-se minimizar o custo da implementação das técnicas de tolerância a falhas, permitindo assim, que a inclusão destas técnicas não onerasse o sistema em custo e desempenho.

Também foi implementado um protótipo de um Serviço de Licenciamento. Os objetivos principais da construção deste protótipo foram: validar as soluções apresentadas no projeto e o de levantar os problemas encontrados na implementação destas soluções.

1.3 Organização do Trabalho

Este trabalho encontra-se dividido da seguinte forma: no capítulo 2 é apresentado a Arquitetura CORBA. No capítulo 3, vamos enfocar o serviço de licenciamento e como CORBA especifica este serviço. Em seguida, no capítulo 4, vamos apresentar um projeto de um Serviço de Licenciamento CORBA com características de tolerância a falhas. No capítulo 5, discutiremos os aspectos envolvidos na implementação deste serviço. No capítulo 6, apresentaremos as considerações finais e perspectivas deste trabalho.

Capítulo 2 - Arquitetura e Serviços CORBA

2.1 Introdução

A diversidade das redes modernas torna a programação em rede muito difícil. As aplicações distribuídas consistem, muitas vezes, em vários programas escritos em diferentes linguagens de programação e rodando em diferentes sistemas operacionais de rede, formando assim um ambiente heterogêneo.

Essa heterogeneidade é causada por diversos fatores, dentre eles podemos citar [Vinoski97]:

Questões de Engenharia: Raramente existe uma única solução aceitável para um complexo problema de engenharia. Como resultado, diferentes pessoas dentro da corporação escolhem diferentes soluções para problemas similares.

Custo: Os vendedores variam em suas habilidades de prover a melhor solução pelo menor custo. Apesar de existir algum grau de fidelidade às marcas, muitos consumidores tendem a comprar sistemas que melhor atendam as suas necessidades pelo preço mais baixo, independente de quem os faça.

Sistemas Legados: Ao longo do tempo, são feitas várias decisões de compra. Sistemas comprados anteriormente podem ser muito críticos ou ter uma substituição muito onerosa. A corporação também pode ter investido muito dinheiro em seu sistema atual, e então esses sistemas devem ser utilizados até que tenham o seu custo pago.

Uma grande vantagem de um ambiente heterogêneo é permitir a melhor combinação de componentes de hardware e software em cada porção da corporação. Para que esses componentes interajam de maneira satisfatória necessitamos de padrões para que o sistema como um todo seja coerente e operacional.

Por outro lado, o desenvolvimento de aplicações que suportem e façam uso eficiente do ambiente de rede heterogêneo não é uma tarefa fácil. Existem vários pacotes de software que auxiliam o desenvolvimento de sistemas em uma plataforma homogênea de rede. Entretanto, poucos lidam com a integração de sistemas desenvolvidos separadamente em um ambiente distribuído heterogêneo.

Em resposta a estes problemas, foi fundada em 1989 a *Object Management Group* (OMG) . A OMG foi criada com o objetivo de desenvolver, adotar e promover padrões para o desenvolvimento de aplicações baseadas na tecnologia de objetos em um ambiente distribuído heterogêneo. Atualmente a OMG é o maior consórcio na área de tecnologia, sendo formada por mais de 700 empresas [Baker97].

No centro de todas as atividades desenvolvidas pela OMG está a *Object Model Architecture* (OMA) [OMA97]. A OMA foi criada com o objetivo de fomentar o crescimento de tecnologias baseadas em objetos e fornecer uma infra-estrutura conceitual para todas as especificações OMG. Como elemento fundamental da arquitetura OMA temos o CORBA *(Common Object Request Broker Architecture)* [OMG98].

CORBA é um padrão de arquitetura que permite a comunicação entre aplicações independentemente das arquiteturas de computadores (hardware), sistemas operacionais e linguagens de programação (software), sobre as quais elas executam. A comunicação entre as aplicações é realizada através de chamadas a métodos de objetos (de forma análoga à programação orientada a objeto convencional).

CORBA traz as vantagens das técnicas orientadas a objeto para o ambiente distribuído (como herança, encapsulamento, polimorfismo, reusabilidade de código, etc.), facilitando a construção de aplicações distribuídas. Este padrão permite o projeto e o desenvolvimento de aplicações distribuídas como um conjunto de objetos cooperantes e a reutilização destes objetos em novas aplicações.

Nas seções seguintes iremos abordar em maiores detalhes a arquitetura OMA, focalizando um dos seus componentes chaves, a especificação *CORBA*.

2.2 OMA (Object Management Architecture)

A arquitetura OMA é composta pelo modelo de objetos e o modelo de referência:

Modelo de Objetos: Quando um grande número de pessoas trabalha em um projeto é necessário que fiquem bem definidos conceitos sobre o que é um objeto, operação, tipos etc. Este modelo define uma semântica comum para objetos especificando as suas características visíveis externamente.

Modelo de Referência: Caracteriza as interações entre os objetos. Podemos ver na Figura 2.1 o Modelo de Referência OMA.

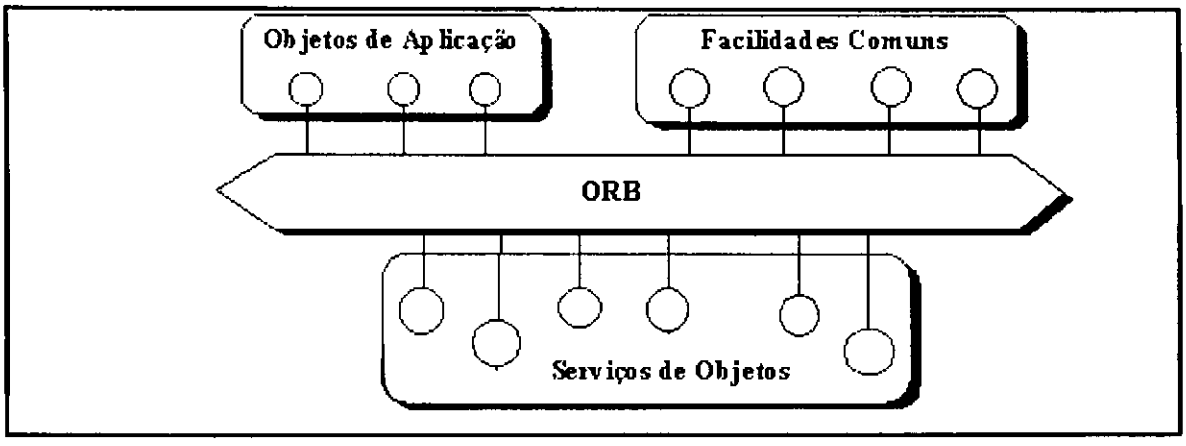

Figura 2.1 – O Modelo de Referência OMA

O Componente *Object Request Broker* (ORB) tem como função principal facilitar a comunicação entre todos os elementos que constituem a arquitetura.

Utilizando o ORB temos os seguintes componentes:

Serviços de Objetos: São serviços de propósito geral que são utilizados para o desenvolvimento de aplicações baseadas em um modelo de objetos distribuídos. Outra finalidade é o provimento de uma base universal, independente do domínio da aplicação, para a interoperabilidade. Esses componentes padronizam o gerenciamento

do ciclo de vida dos objetos. Aqui temos interfaces para a criação de objetos, controle de acesso aos mesmos, transações, segurança etc.

Os Serviços de Objetos padronizados pela OMG são chamados de *CORBAservices*. Os serviços já padronizados pela OMG são: Serviço de nomes, Eventos, Ciclo de Vida, Objetos persistentes, Transações, Controle de Concorrência, Relacionamentos, Externalização, Consulta, Propriedades, Segurança, Tempo, Coleções, Negociação e Licenciamento [OMGserv99] .

Sendo este último, o tema central desta dissertação. No capítulo 3, analisaremos com maiores detalhes o serviço de licenciamento.

Facilidades Comuns: Enquanto os Serviços de Objetos fornecem serviços para objetos individuais ou grupos de objetos, as Facilidades Comuns disponibilizam seus serviços para as aplicações finais [Baker97]. A primeira destas Facilidades adotadas pela OMG foi a *Distributed Document Component Facility (DDCF)*, baseado no OpenDoc. DDCF permite a apresentação e o intercâmbio de documentos baseados em um modelo de documento, facilitando, por exemplo, que uma planilha seja conectada a um relatório [Vinoski97]. Entre as Facilidades Comuns encontram-se: Facilidades para Internacionalização, tempo, Intercâmbio de dados, Facilidade para agentes móveis, impressão, etc .[Vogel98].

Objetos de Aplicação: Correspondem à noção tradicional de aplicações de usuários, que por esse motivo, não são padronizadas pela OMG.

É importante notar que as aplicações precisam apenas suportar ou usar interfaces compatíveis com as padronizadas pela OMG para participarem do modelo OMA. Estas aplicações não necessitam terem sido desenvolvidas utilizando o paradigma da orientação a objeto.

No centro do Modelo de Referência OMA está o ORB. O ORB vem sendo um dos elementos que tem recebido mais atenção por parte da OMG. Isto é necessário pois todos os componentes no modelo OMA dependem do ORB. Na próxima seção analisaremos o funcionamento do ORB e sua especificação.

2.3 Common Object Request Broker Architecture (CORBA)

Uma das primeiras especificações adotadas pela OMG foi a especificação CORBA. Esta especificação detalha as interfaces e características do ORB (componente da arquitetura OMA). A especificação CORBA mais recente lançada pela OMG é a versão 2.3 [OMG99].

Uma arquitetura CORBA consiste principalmente de um tratador de requisições chamado ORB e de uma linguagem de definição de interfaces chamada IDL (*Interface Definition Language)*. Um sistema típico é feito de um conjunto de programas **clientes** que fazem uso dos objetos distribuídos através do sistema. CORBA define que os objetos existem nos **servidores** (freqüentemente é utilizado o termo *implementação* no lugar de servidor). Por isso, dizemos que um cliente CORBA invoca um método de um servidor. Este servidor CORBA pode desempenhar o papel de cliente se este invocar um método em outro servidor. Da mesma forma, o cliente pode desempenhar o papel de servidor.

O ORB é um software que serve de intermediário entre as transferência de mensagens de um objeto para outro. O ORB entrega as requisições aos objetos e retorna as respostas aos clientes. O papel do ORB é o de esconder a complexidade da rede para o programador. Um ORB permite que os objetos possam ser invocados de qualquer ponto da rede. Quando um cliente invoca uma função de um objeto CORBA, o ORB intercepta este chamado e o redireciona através da rede em direção ao objeto alvo, como podemos ver na Figura 2.2.

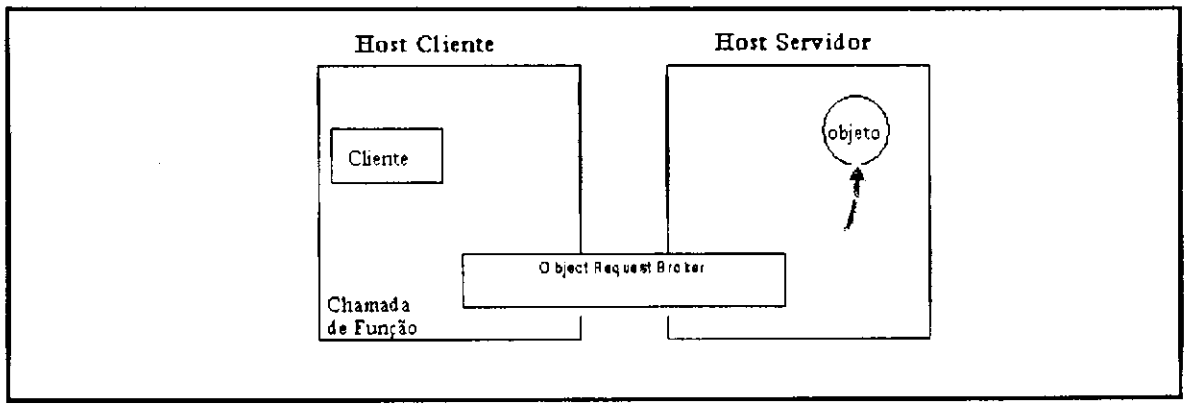

Figura 2.2 – O ORB (Object Request Broker)

A IDL (*Interface Definition Language)* possibilita a transparência em relação às linguagens de programação. Um objeto cliente pode chamar métodos em um objeto servidor de diferentes classes, não importando a linguagem em que estas estão implementadas (Figura 2.3).

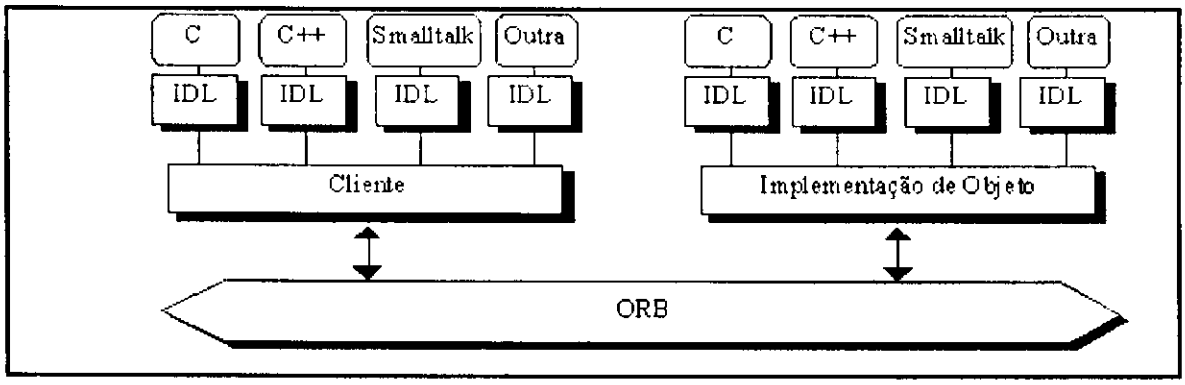

Figura 2.3 – IDL provê independência de linguagem de programação

Os objetos devem saber os nomes dos métodos que o objeto servidor expôs. Neste ponto é que entra a IDL. CORBA IDL é uma maneira neutra (independente de uma linguagem específica) de especificar tipos de dados, atributos, operações, interfaces etc.

```
1 modulo  Exemplo {
2       interface ServidorBanco {
3                       boolean verifiqueSenha (in long num_conta, in long senha);
4                       void obtenhaDetalhesConta (in long num_conta, in string nome_cliente,
5                                               out double balanco, out boolean foiChecada);
6                       boolean processeTransacao (in Transacao t, in long num_conta);
7       }
8 }
```

Figura 2.4 – Exemplo de uma Especificação em CORBA IDL

Na Figura 2.4 podemos ver um exemplo de uma especificação em CORBA IDL. Neste exemplo é mostrada a especificação de um Servidor Bancário simplificado. Esta especificação indica que o objeto ServidorBanco terá três métodos: um para verificar a senha em relação à conta (linha 3), um para obter detalhes sobre a conta (linha 4), e um para processar uma transação sobre uma conta (linha 6).

A linguagem IDL é bastante parecida com C++ na forma de definição de classes e seus métodos. Entretanto, como IDL trata de objetos distribuídos, esta força a definição de informações adicionais sobre a interface do objeto, tais como: quais

argumentos dos métodos são somente de entrada, somente de saída, ou de entrada e saída. Isto é feito através da inclusão das palavras *in*, *out* e *inout*. Na interface ServidorBanco, os dois argumentos do método verifiqueSenha() são declarados como *in*, pois eles são utilizados somente como entrada neste método e não necessitam serem retornados no fim da execução do método. O método obtenhaDetalhesConta() tem dois parâmetros *in* e dois *out*. Os dois argumentos *out* são retornados para o cliente no retorno do método. O argumento *inout* usa ambas as abordagens anteriores. Este serve de entrada ao método e retorna seu valor no retorno da chamada [Farley98]. No Apêndice A podemos ver um exemplo mais completo de uma especificação em CORBA IDL. No caso, a especificação do Serviço de Licenciamento CORBA.

Uma definição de interface escrita em OMG IDL define completamente a interface e especifica cada parâmetro da operação. É importante ressaltar que os objetos não são escritos em OMG IDL, que é uma linguagem puramente descritiva. Eles são escritos em linguagens de programação que possuam mapeamentos definidos dos conceitos existentes em OMG IDL. Em [OMG98] são descritos mapeamentos para C, C++, Smalltalk, Cobol, ADA e Java.

Na prática, após a descrição do objeto em OMG IDL, é utilizada alguma ferramenta (que geralmente é adquirida junto com ORB), que compilará OMG IDL para a linguagem de programação de preferência do programador.

2.4 Estrutura de uma Aplicação CORBA

Para a construção de aplicações CORBA podem ser utilizadas duas abordagens: estática e dinâmica.

A abordagem estática é utilizada quando, em tempo de compilação, o cliente sabe quais os métodos que podem ser invocados no servidor. A abordagem dinâmica é utilizada quando em tempo de execução o cliente necessita chamar métodos em um servidor, os quais não eram conhecidos em tempo de compilação. Esta abordagem permite que seja adicionada novas funcionalidades a um servidor sem que seja necessária a realização de alterações no cliente. Desta forma, os clientes podem descobrir em tempo de execução quais são os serviços oferecidos pelo servidor.

2.4.1 Estrutura de uma Aplicação CORBA Usando a Abordagem Estática

Esta abordagem é usada para escrever uma aplicação onde o cliente e o servidor serão escritos a partir da compilação da IDL e implementados a partir deste código gerado. Isto significa que o programa cliente só poderá chamar um método do servidor nos objetos cujas interfaces são conhecidas em tempo de compilação.

O primeiro passo no desenvolvimento desta aplicação CORBA é a definição das interfaces para os objetos do sistema, usando CORBA IDL. Em seguida estas interfaces são compiladas usando um compilador de IDL.

Este compilador irá produzir código de programação na linguagem de programação escolhida, gerado a partir da IDL. Dentre os programas gerados teremos o código *stub* do cliente, que irá permitir o desenvolvimento de clientes *e o skeleton* do servidor, que irá permitir a implementação dos objetos CORBA. O *stub* do cliente intercepta as chamadas dos clientes e os transfere ao ORB enquanto o *skeleton* entrega as chamadas à implementação do objeto CORBA [Iona99].

Na Figura 2.5 podemos notar que quando o cliente chama um método de um objeto CORBA, esta chamada é transferida através do código *stub* do cliente para o ORB. Se o cliente não acessou este objeto antes, o ORB procura um banco de dados chamado Repositório de Interfaces para determinar exatamente qual objeto deve receber a chamada ao método. O Repositório de Interfaces é um banco de dados que armazena informações sobre as interfaces IDL que são implementadas pelos objetos na rede. O ORB então passa a chamada do método através do *skeleton* em direção ao objeto alvo.

Quando o objeto que implementa o servidor concluir o processamento deste pedido, é enviada a resposta ao cliente. A resposta segue pelo caminho contrário ao do envio: esta é entregue ao *skeleton*, a seguir é enviada ao ORB. O ORB entrega ao *stub* do cliente e este entrega ao cliente.

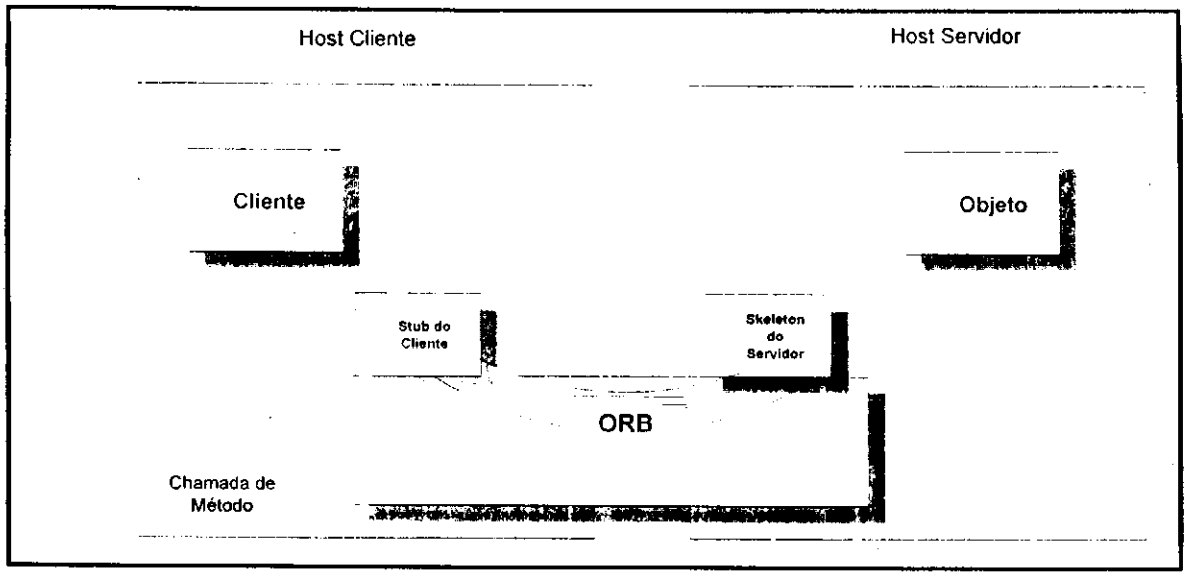

Figura 2.5 – Chamando um método em um Objeto CORBA

2.4.2 Estrutura de uma Aplicação CORBA Usando a Abordagem Dinâmica

Na abordagem anterior, o cliente podia apenas chamar métodos no servidor que eram conhecidos em tempo de compilação. Usando a abordagem estática mostrada anteriormente, as interfaces IDL que o programa cliente podia utilizar eram determinadas quando este foi compilado. Isto é uma séria limitação para alguns tipos de aplicações como *browsers* por exemplo. Estes tipos de aplicações necessitam utilizar um número ilimitado de interfaces: interfaces que talvez não tivessem sido ainda concebidas quando o programa cliente estava em desenvolvimento. Quando o cliente necessita obter informação sobre uma interface IDL em tempo de execução, teremos o uso da abordagem dinâmica.

Um programa cliente pode consultar o Repositório de Interfaces em tempo de execução para obter informações sobre estas interfaces. Este cliente, pode então, chamar os métodos dos objetos usando um componente do ORB chamado *Dynamic Invocation Interface* (DII). Podemos ver na figura 2.6 com isto acontece [Iona99].

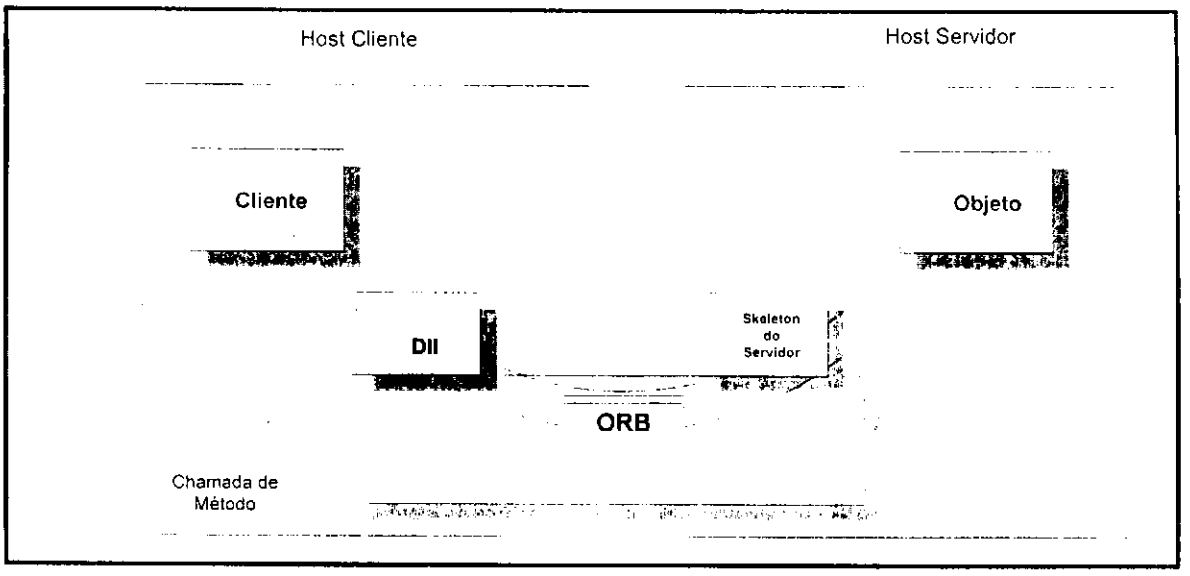

Figura 2.6 – Cliente chamando um método usando o DII

2.5 Interoperabilidade entre ORBs

Como visto anteriormente, o ORB faz com que a comunicação entre os objetos seja feita de forma transparente aos programadores de rede. Para que isto seja atingido é necessário que o ORB se comunique com outros ORBs.

Em uma rede poderemos ter muitos ORBs se comunicando entre si. Cada um deles desenvolvido por empresas diferentes. É necessário que aplicações desenvolvidas para um determinado ORB se comuniquem com programas desenvolvidos para ORBs de outros fabricantes. Por isso é necessário que haja uma padronização sobre como um ORB irá se comunicar com outro. Para assegurar que isto aconteça, CORBA especifica que o ORB deve se comunicar com outro usando um protocolo padronizado o GIOP (*General Inter-ORB Protocol).* O *Internet Inter-ORB Protocol* (IIOP) é a especificação de como o GIOP é implementado sobre TCP/IP.

2.6 Exemplo Completo de Implementação de uma Aplicação CORBA

Será mostrada agora a implementação de uma aplicação CORBA muito simples.

Este exemplo mostra a implementação do programa conhecido como "Hello World" na versão Cliente-Servidor [OOC99b]. Neste caso o cliente simplesmente chama uma operação no servidor para imprimir o texto "Hello World". Este exemplo será utilizado para mostrar os passos da implementação de uma aplicação CORBA.

2.6.1 A IDL

O passo inicial no desenvolvimento de uma aplicação CORBA é a definição da interface em IDL. A figura 2.7 mostra o código em IDL desta interface.

```
1    interface Hello
2    {
3            void hello();
4    };
```

Figura 2.7 – Código IDL da Aplicação "Hello-World" na Versão Cliente-Servidor

É visto nesta aplicação a necessidade de apenas uma interface e que esta possui apenas uma operação, a operação `hello()` (linha 3).

2.6.2 Compilação da IDL

Agora que a interface já foi definida, esta será compilada e será gerado código-fonte na linguagem de programação escolhida, neste caso Java. Essa compilação irá gerar os seguintes arquivos: `Hello.java`, `HelloHelper.java`, `HelloHolder.java`, `StubForHello.java` e `_HelloImplBase.java`. Todos estes arquivos são armazenados em um diretório de nome `Hello`.

2.6.3 Implementação do Servidor

Agora será implementado o servidor de nossa aplicação. Na Figura 2.8 podemos visualizar a implementação deste servidor:

```
1   package hello;
2
3   import org.omg.CORBA.*;
4
5   public class Hello_impl extends _HelloImplBase
6   {
7           public void hello()
8           {
9                   System.out.println("Hello World!");
10          }
11  }
```

Figura 2.8 – Implementação do Servidor (Arquivo `Hello_impl.java`)

Podemos ver que este programa herda as definições do programa `_HelloImplBase` (linha 5), um dos arquivos que foram gerados automaticamente pela compilação da IDL. Em seguida é definida a única operação do método `hello()` (linha 7), imprimir a cadeia de caracteres "Hello Word" (linha 9). Este arquivo será salvo com o nome de *Hello_impl.java.*

Agora será escrita a classe que possui o método `main()` e que é a responsável pela ativação do Servidor. Esta classe será chamada `Server.java`. Na Figura 2.9, podemos ver a implementação desta classe.

Este código parece complexo à primeira vista, mas é possível observar que sua função principal é iniciar a classe `Hello_impl` (linha 27). Este arquivo pode ser divido nas seguintes etapas: inicialização do ORB (linhas 10 a 26), em seguida inicialização da classe `Hello_impl` (linha 27), armazenamento de sua referência em um arquivo para que o Cliente possa encontrá-lo (linhas 28 a 48), indicação que o servidor está pronto para receber pedidos (52) e finalização (53 a 62).

```
1   package hello;
2
3   import org.omg.CORBA.*;
4   import java.io.*;
5   import java.util.*;
6
7   public class Server
8   {
9   public static void main(String args[]) {
10  Properties props = System.getProperties();
11  props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
12  props.put("org.omg.CORBA.ORBSingletonClass",
13            "com.ooc.CORBA.ORBSingleton");
14  System.setProperties(props);
15
16
17
18    //
19    // Criando ORB e BOA
20    //
21     ORB orb = ORB.init(args, props);
22     BOA boa = ((com.ooc.CORBA.ORB)orb).BOA_init(args, props);
23
24     //
25     // Cria a implementação do Objeto
26     //
27     Hello_impl p = new Hello_impl();
28
29     //
30     // Armazena referência
31     //
32     try
33     {
34           String ref = orb.object_to_string(p);
35           String refFile = "Hello.ref";
36           FileOutputStream file = new FileOutputStream(refFile);
37           PrintWriter out = new PrintWriter(file);
38           out.println(ref);
39           out.flush();
40           file.close();
41     }
42     catch(IOException ex)
43     {
44           System.err.println("Can't write to " +
45                              ex.getMessage() + """);
46           System.exit(1);
47     }
48
49     //
50     // Executa Implementação
51     //
52     boa.impl_is_ready(null);
53  }
```

Figura 2.9 – Implementação da Classe `Server.Java`

```
54 catch(SystemException ex)
55 {
56          System.err.println(ex.getMessage());
57          ex.printStackTrace();
58          System.exit(1);
59       }
60
61       System.exit(0);
62    }
63 }
92
93
94
```

Continuação da Figura 2.9

2.6.4 Implementação do Cliente

Agora será mostrado na Figura 2.10 o código do Cliente. Este código pode ser dividido em 3 partes:

1) Inicialização do ORB – Linhas 9 a 16.
2) Descoberta da referência para o servidor que implementa a interface `Hello` – Linhas 18 a 40.
3) Chamada o método `hello()` (linha 43).

2.6.5 Compilação dos Arquivos e Execução das Classes

O próximo passo será a compilação de todos os arquivos, usando o compilador Java. Todos os arquivos mostrados anteriormente serão compilados e com isto teremos a geração das classes. O passo seguinte será a execução destas. O servidor será executado e em seguida o cliente.

```
1       package hello;
2       import org.omg.CORBA.*;
3       import java.io
4       import java.util.*;
5       public class Client
6       {
7       public static void main(String args[])
8       {
9       Properties props = System.getProperties();
10      props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
11      props.put("org.omg.CORBA.ORBSingletonClass",
12                "com.ooc.CORBA.ORBSingleton");
13      System.setProperties(props);
14
15        // Criando o  ORB
16        ORB orb = ORB.init(args, props);
17        //
18        // Descobrindo a Referência para o objeto "hello"
19        //
20        String ref = null;
21        try
22        {
23              String refFile = "Hello.ref";
24              FileInputStream file = new FileInputStream(refFile);
25              BufferedReader in =
26                 new BufferedReader(new InputStreamReader(file));
27              ref = in.readLine();
28              file.close();
29        }
30        catch(IOException ex)
31        {
32              System.err.println("Can't read from `" +
33                                 ex.getMessage() + "'");
34              System.exit(1);
35        }
36        org.omg.CORBA.Object obj = orb.string_to_object(ref);
37        if(obj == null)
38              throw new RuntimeException();
39
40        Hello hello = HelloHelper.narrow(obj);
41
42
43        hello.hello();
44
45
46
47
48      }
49  }
```

Figura 2.10 – Implementação do Cliente

2.7 Sumário

Neste capítulo descrevemos a *Common Object Request Broker Architecture (CORBA)* integrante da *Object Management Architecture* (OMA).

CORBA é um padrão de arquitetura que permite a comunicação entre aplicações independentemente de hardware e software. A comunicação entre as aplicações é realizada através de chamadas a métodos de objetos (de forma análoga à programação orientada a objeto convencional).

O modelo de referência OMA é composto do ORB, Objetos de Aplicação, Serviços de Objetos e Facilidades Comuns.

O ORB é o elemento central deste modelo, responsável pela comunicação entre os outros elementos. Os Objetos de Aplicação são os aplicativos do usuários. As Facilidades Comuns fornecem seus serviços para as aplicações finais. Os Serviços de Objetos fornecem serviços mais básicos ao objetos. Entre os Serviços de Objetos, encontra-se o Serviço de Licenciamento, que será detalhado no próximo capítulo.

Capítulo 3 - Gerenciamento de Licenças de Software

3.1 Introdução

Quando um software é comprado de um vendedor legal, existe um contrato entre o vendedor e o comprador. A este contrato chamamos de licença. A pirataria de software pode ser definida como o uso ou cópia de software de forma a violar essa licença.

Com o avanço das redes de computadores temos também o crescimento de aplicativos executando em rede, dificultando assim, cada vez mais, o controle do uso do software. Os produtores destes aplicativos necessitam de meios de controlar o acesso aos seus produtos, assegurando assim, uma justa compensação pelo seu uso. A forma mais comum de se realizar este controle é através do serviço de licenciamento, onde cada licença para o uso do software é fornecida por um meio técnico (baseado em mecanismos de hardware ou software), ou através de alguma forma contratual (através da adoção de sanções legais ao usuário de uma cópia não autorizada).

Apesar da forma contratual ser uma opção viável, ela não assegura um nível de controle tão satisfatório quanto um meio técnico. Por esta razão, os fornecedores de software necessitam de métodos de licenciamento técnicos para complementar os contratos legais.

Uma das formas de resolver este problema é através do uso do serviço de gerenciamento de licenças. Este serviço consiste em um sistema distribuído que controla a utilização de produtos de software segundo os termos contidos na licença.

Atualmente, a maioria dos softwares que fazem gerenciamento de licenças realizam também uma função adicional, a medição do uso de licenças (*License Metering*). A medição do uso de licenças permite que seja avaliado o grau de utilização dos softwares dentro das corporações. Com isto, estas terão dados confiáveis

para a compra de novas licenças, sabendo assim, quais são as suas reais necessidades de aquisição de software [Kotopoulis98].

3.2 Políticas de Licenciamento

A política de licenciamento irá definir como será feito o controle do uso do software. Existem vários atributos que podem ser usados na definição da política de licenciamento. Alguns exemplos de políticas de licenciamento:

Quantidade de ativações acumuladas: Indica o número pré-determinado de vezes que o usuário pode executar a aplicação. Toda vez que a aplicação é executada, o servidor de licenças decrementa o número máximo de ativações permitidas. Isto é feito até que esse número chegue a zero.

Data da Expiração: Estabelece uma data de validade para a utilização da aplicação. Uma vez expirada esta data, não mais será permitida a ativação da aplicação. Esta é uma política de licenciamento bastante difundida na Internet para o uso de cópias de demonstração.

Reserva de Nó : Permite a aplicação ser utilizada somente no nó da rede em que a licença foi criada.

Licenciamento de Sítio (*Site Licensing):* Permite o uso ilimitado de cópias de um determinado software dentro de determinado domínio. Esse domínio pode ser um departamento de uma grande empresa, uma corporação, etc.

A evolução dos sistemas computacionais tem trazido mudanças nos mecanismos de licenciamento. Métodos tradicionais de licenciamento, como os de *Reserva de nó* e *Licenciamento de Sítio*, que eram utilizados no passado, especialmente nos sistemas de tempo compartilhado e nas estações de trabalho não ligadas em rede, têm se mostrado insuficientes nos ambientes de computação atuais. Estes mecanismos são caracterizados por um alto custo para o cliente (devido a este muitas vezes ter que pagar por uma licença que não utiliza completamente) e por falta de flexibilidade [OMG96].

Com o desenvolvimento das redes de computadores, passou-se a introduzir a idéia do **licenciamento concorrente de software**. Esta forma de licenciamento define o número de usuários que podem acessar uma determinada aplicação em um dado momento. Essas licenças estão disponíveis pela rede e são temporariamente apropriadas pelos usuários, assim que as aplicação são executadas. Quando as aplicação terminam de executar, elas retornam ao repositório de licenças. Esta forma de licenciamento tem se mostrado mais flexível e adaptada às necessidades das empresas que os mecanismos tradicionais de licenciamento.

3.3 Aspectos Envolvidos no Projeto de um Gerenciador de Licenças

Atualmente os desenvolvedores de software necessitam de ferramentas de licenciamento mais flexíveis para implementar suas práticas de negócio e preço. É de fundamental importância que os sistemas de licenciamento não imponham as suas práticas de negócios aos usuários. A licença de software é de fato um contrato onde é estabelecida uma relação de negócios entre fornecedores e clientes. Um sistema de licenciamento de software exerce um papel fundamental na regulamentação desse contrato. Por isso, ele deve ter bastante flexibilidade para se adaptar às práticas comerciais que as empresas desenvolvedoras de software necessitam para lidar com uma ampla gama de clientes.

Outro aspecto a ser considerado no desenvolvimento de um gerenciador de licenças é o fornecimento de um grande conjunto de possibilidades de licenciamento. Se um sistema de licenciamento oferece apenas um número limitado de tipos de licença ou poucas opções a serem aplicadas a estes tipos, estaremos limitando a forma como os fornecedores de software realizam negócios com seus clientes.

O sistema de licenciamento de software lida em muitos aspectos do relacionamento da empresa com o consumidor, incluindo por exemplo: atualizações, suporte, melhoramentos, cópias de demonstração etc. Devido a isto, temos que assegurar que os fornecedores de software, e não os desenvolvedores de gerenciadores de licenças, devam escolher que opções de licenciamento queiram usar.

Um sistema de licenciamento ideal deve ser transparente a seus usuários finais.

Por exemplo, um usuário executa uma aplicação que faz chamada a uma biblioteca de licenciamento. Então, uma função da biblioteca localiza o servidor requisitando uma licença. Assumindo que uma licença está disponível e que a pessoa está autorizada a usar esta licença, é retornada uma permissão para o uso desta aplicação, permitindo assim a execução desta. Tudo isso, de forma transparente ao usuário.

Se não houver licenças disponíveis, o gerenciador de licenças pode atuar de diversas formas: pode colocar o usuário em uma fila de espera, perguntar ao usuário que tipo de ação tomar, recomendar que o usuário tente de novo mais tarde, deixar o software rodando em um modo de demonstração (sem algumas de suas funcionalidades), etc. Estas escolhas e como elas são implementadas devem estar de acordo com a política de negócios do fornecedor do software.

Outras características importantes de um gerenciador de licenças são a sua escalabilidade, confiabilidade e segurança. A escalabilidade garantirá o correto funcionamento do gerenciador de licenças em qualquer tamanho de rede. A confiabilidade garantirá a continuação da prestação do serviço em situações adversas como particionamento da rede, não devolução de uma licença concedida após o termino de execução da aplicação, etc. A segurança garantirá a proteção do serviço de licenciamento de violações causadas por pessoas agindo de má fé. Por isso, também são necessários mecanismos de autenticação para garantir que os aplicativos clientes do serviço de licenciamento estejam lidando com usuários autorizados e vice-versa.

O gerenciador de licenças deve permitir o uso das aplicações existentes sem que seja necessária nenhuma mudança no código fonte. Isto geralmente é feito usando uma camada de software que "empacote" a aplicação existente e aumente a funcionalidade desta, trazendo assim esta aplicação para o ambiente de licenciamento.

3.4 Funcionamento de um Gerenciador de Licenças

Na maioria dos casos, o serviço de licenciamento é baseado no modelo cliente-servidor e oferece uma interface de programação que permite ao desenvolvedor de software integrá-lo aos seus produtos. Com isto, a política de licenciamento do produto irá ficar isolada de sua funcionalidade, em uma camada de software isolada e bem

definida [Bezerra97].

De uma forma simplificada, podemos ver na Figura 3.1, as operações que ocorrem em um serviço de licenciamento.

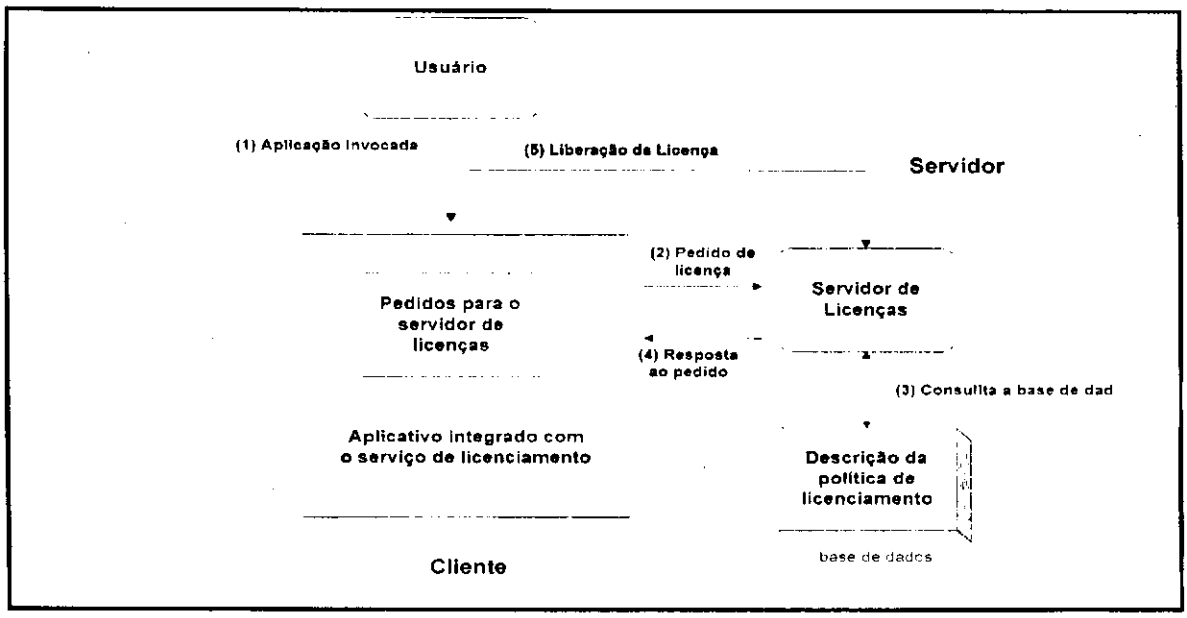

Figura 3.1 – A aplicação solicita ao Servidor de Licenças uma licença para o seu uso.

Os passos executados são os seguintes:

1) O usuário executa a aplicação.

2-3) O Servidor de Licenças verifica se é possível atender ao pedido de licença, acessando uma base de dados que descreve a política de licenciamento e contém as informações relevantes sobre as licenças em uso.

4) Se é possível atender ao pedido de licença, o servidor responde positivamente ao cliente, que avisa à aplicação para prosseguir sua execução. Caso não seja possível, a atitude tomada dependerá do desenvolvedor (finalizar a aplicação, alertar ao usuário através de uma mensagem e prosseguir a execução, etc.).

5) Quando a aplicação finaliza, esta informa ao servidor a liberação da licença.

Com o crescimento do mercado de componentes de software que agregam um funcionalidade a um produto já existente ou como módulos de um sistema, é preciso ampliar os conceitos de gerenciamento de licenças de aplicações para o de gerenciamento de licenças de módulos de software. Ou seja, o esquema de gerenciamento de licenças deverá também ser capaz de gerenciar o uso de componentes. E os componentes deverão ser escritos de uma forma que automaticamente façam seus registros nos gerenciadores de licenças.

3.5 O Gerenciamento de licenças no ambiente CORBA

O gerenciamento de licenças no ambiente CORBA está especificado como um dos Serviços de Objetos da arquitetura OMA. O *Licensing Service Specification* (Especificação do Serviço de Licenciamento) é uma das especificações de Serviços de Objetos padronizadas pela OMG, definidas em [OMG96].

O Serviço de Licenciamento CORBA especifica uma interface que o software de um determinado produtor pode utilizar para interagir com o serviço de licenciamento.

As interfaces para o serviço de licenciamento necessitam ser muito flexíveis devido ao grande número de modelos de licenciamento que podem ser utilizados por diferentes produtores de software e pelas variações possíveis destes modelos de forma a personalizar o modelo de licenciamento para um determinado usuário.

As questões relativas à administração e políticas de licenciamento não são detalhadas pelas interfaces do serviço de licenciamento. Estas questões são deixadas a cargo do implementador.

3.5.1 Interfaces para o Serviço de Licenciamento

A especificação do serviço de gerenciamento de licenças no ambiente CORBA consiste basicamente em duas interfaces (mostradas na Tabela 3.1) que fornecem todas as operações que os componentes necessitam para se protegerem contra o uso da cópia

ilegal. Os componentes usam estas interfaces para permitir que o serviço monitore quem e por quanto tempo está sendo utilizado.

Interface	Operações
LicenseServiceManager	▪ obtain_producer_specific_license_service
ProducerSpecificLicenceService	• start_use • check_use • end_use

Tabela 3.1 - Interfaces do Serviço de Licenciamento CORBA

Vamos agora descrever estas interfaces:

LicenseServiceManager: Esta interface provê um mecanismo para que o objeto a ser licenciado localize a segunda interface, a interface *ProducerSpecificLicenseService*. A interface *LicenseServiceManager* suporta uma única operação: *obtain_producer_specific_license_service*. Esta operação retorna uma referência para um objeto que implementa um serviço de licenciamento particular.

ProducerSpecificLicenseService: fornece 3 operações que permitem fazer todo o trabalho de gerência de licenças. O componente licenciado invoca *start_use* quando ele é primeiramente usado. Deve ser passado como parâmetro toda a informação necessária associada com o componente do usuário. O componente chamará *end_user* quando ele parar de ser utilizado. O componente deve invocar *check_use* periodicamente enquanto ele estiver sendo utilizado para possibilitar que o serviço saiba que a conecção com o cliente permanece ativa. O serviço deve checar o tempo de expiração da licença e retornar uma mensagem a este respeito. O componente sabe quando chamar *check_use* por uma destas formas: ou pela realização de um "*polling*" em um intervalo especificado pelo servidor, ou pelo recebimento assíncrono de uma notificação de um evento do servidor dizendo para chamar o *check_use*.

Todas estas operações entre os componentes e os serviços são protegidos por um mecanismo de autenticação chamado *challenge*.

Chamaremos de **produtores** (*producers*), os objetos que implementam esta interface. Como os produtores são específicos para cada produto ou fabricante, normalmente teremos vários deles rodando simultaneamente na rede.

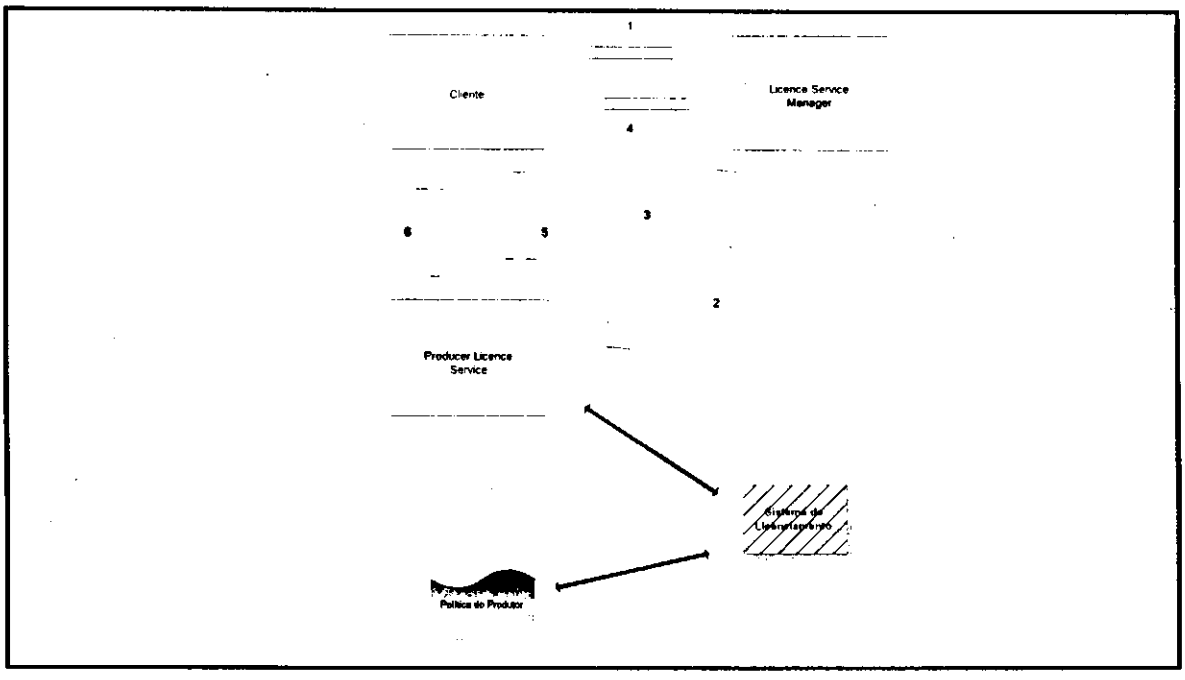

Figura 3.2 – Diagrama de Instância do Serviço de Licenciamento

Na Figura 3.2 podemos ver o Cliente (producer client) executar a operação *obtain_producer_specific_licence_service* na interface **LicenceServiceManager** (passo 1). A implementação do serviço gerenciador de licenças cria um objeto (passos 2 e 3) ou localiza uma referência para um objeto que tenha uma interface **ProducerSpecificLicenseService**. Então é retornado uma referência para o Cliente do produtor (passo 4). O Cliente do produtor usa esta referência para executar as operações *start_use*, *check_use* e *end_use* (passos 5 e 6). Em uma implementação que suporte eventos assíncronos, o objeto **ProducerSpecificLicenseService** pode assincronamente executar uma operação *push* usando como referência a interface do Cliente provido como um dos argumentos da operação *start_use* no passo anterior (passo 5).

3.5.2 Um Cenário de Licenciamento no Ambiente CORBA

A Figura 3.3 [OMG96] representa o fluxo de eventos que ocorre no Serviço de Licenciamento CORBA.

1- O Cliente executa a operação *obtain_producer_specific_license_service*. Com esta operação ele recebe uma referência para um objeto que implementa o *Producer Specific License Service.*

2- O Cliente determina que o controle de uso é necessário e executa a operação *start_use.*

3- O Cliente faz um *check_use* inicial para receber como parâmetro o tempo em que deverá ficar repetindo esta operação.

4– A instância do *Producer Specific License Service* interpreta a política de licenciamento e interaje com o sistema de licenciamento se necessário.

5– Se eventos assíncronos são suportados, o *Producer Specific License Service* pede notificação de eventos a um determinado cliente, em um intervalo determinado pela política de licenciamento.

6- O Serviço de eventos entrega os eventos ao cliente.

7- O Cliente responde ao evento executando uma operação *check_use.*

Os passos 4,5,6,7 são repetidos até que esta instância do cliente indique que o controle de uso não é mais necessário.

8 –O cliente executa a operação *end_use* quando o controle do uso não for mais necessário

Se eventos assíncronos não forem suportados, a implementação do cliente necessitará fazer um "polling" no *Producer Specific License Service* em um intervalo que será fornecido como um dos argumentos de retorno da operação *check_use.*

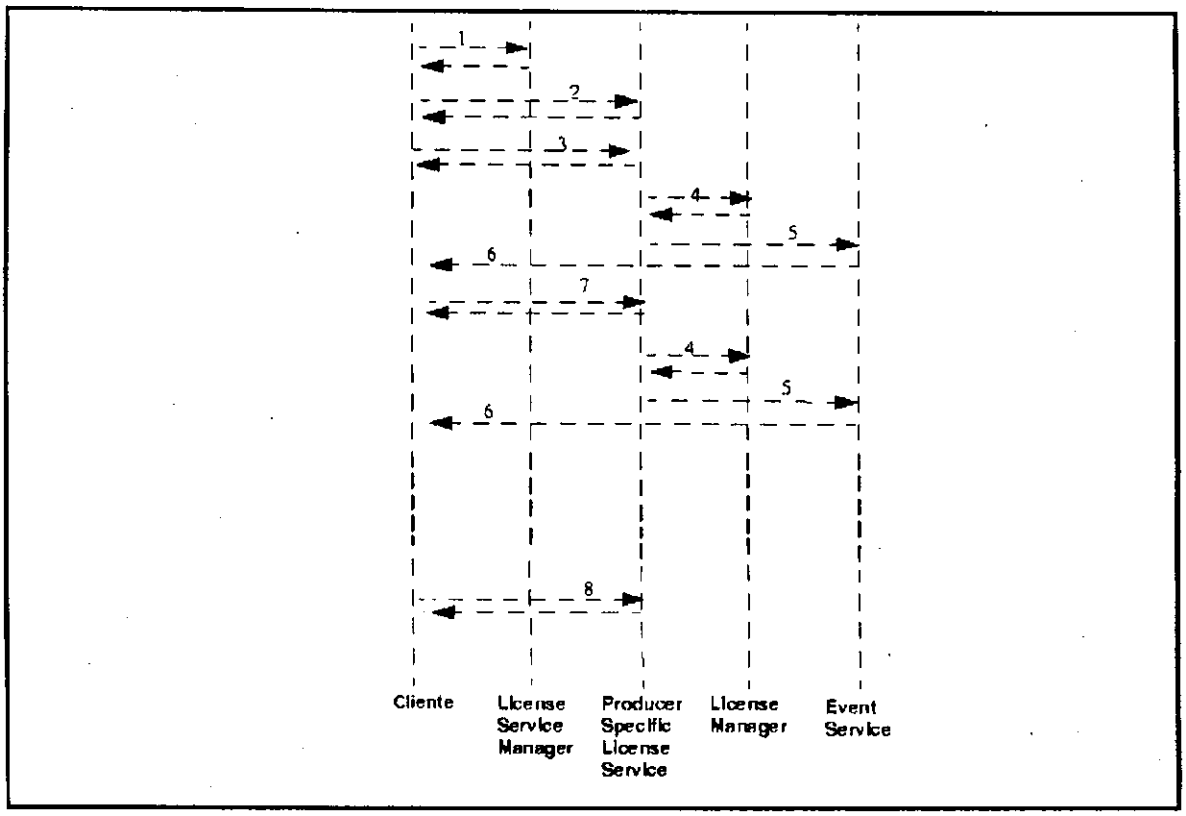

Figura 3.3 - Diagrama de Eventos do Serviço de Licenciamento

3.6 O CosLicensingModule

O arquivo *CosLicensingModule* (mostrado no Apêndice A) é a especificação em IDL das interfaces do Serviço de Licenciamento CORBA.

Este módulo usa algumas definições do módulo *CosEventComm* (que define o serviço de Eventos) e do *CosPropertyService* (que define o serviço de propriedades). O uso destes serviços é opcional na implementação do Serviço de Licenciamento.

A inclusão do Serviço de Propriedades no Serviço de Licenciamento torna possível maior facilidade na passagem das propriedades de contexto (ex: hora local, endereço IP da máquina, hora em que o objeto foi ativado, etc ...) entre os objetos.

Já o Serviço de Eventos torna possível a comunicação de eventos assíncronos aos objetos. Isso possibilita que o objeto seja comunicado caso haja algum evento. Um exemplo de evento é a ocorrência de algum erro no sistema. O objeto pode então ser

notificado da ocorrência deste evento. No nosso projeto, que será mostrado adiante, não utilizamos o Serviço de Eventos.

3.7 Exemplo do uso de um Gerenciador de Licenças no Ambiente CORBA

Para tornar mais claro o uso do serviço de licenciamento CORBA, vamos mostrar uma aplicação interagindo com este serviço:

```
1    Ref_para_produtor=X.obtain_producer_specific_license_service(parâmetros);
2    Licensa = Ref_para_o_produtor.start_use(parâmetros);
3    Ref_para_o_produtor.check_use(parâmetros);
4    Ref_para_o_produtor.end_use(parâmetros);
```

Figura 3.4 – Exemplo de uma Aplicação Interagindo com o Serviço de Licensiamento

- Inicialmente a aplicação pede ao ORB uma referência para um objeto que implemente o Serviço de Licenciamento. O ORB retorna uma referência para um objeto que implementa a interface *LicenseServiceManager*. Vamos supor que a referência retornada tenha sido X.

- De posse desta referência a aplicação pode agora chamar um método neste objeto (linha 1). A aplicação envia como parâmetros desta chamada: seu nome (para que este objeto possa retornar uma referência para um objeto apropriado que implemente a interface *ProducerSpecificLicenseService)* e uma estrutura chamada *Challenge* para efeitos de autenticação (esta estrutura normalmente usa o algoritmo de criptografia MD5 e será utilizada em todas as chamadas aos métodos do Serviço de Licenciamento).

- Agora a aplicação já possui a referência para o objeto que será responsável pelo seu controle (está armazenado na varável Ref_para_produtor). Será chamado agora o método *start_use* (linha 2). A aplicação envia como parâmetors uma série de

informações sobre si, tais como: nome, versão, nó da rede etc. Estas informações serão utilizadas para a concessão ou não da licença. O Serviço de licenciamento faz uso do Serviço de Propriedades CORBA para o envio de parâmetros adicionais, possibilitando assim, que seja enviado um grande número de informações de contexto que não foram especificados na interface do Serviço de Licenciamento. É possível enviar somente uma variável do serviço de propriedades com um grande número de informações. O retorno deste chamado será um *token* representando a licença.

- Agora será chamado o método *check_use* (linha 3). Entre os parâmetros enviados ao produtor temos: O *token* da licença recebida na operação anterior, informações de contexto que são enviadas usando o Serviço de Propriedades (endereço IP da máquina, hora local, etc.). Como parâmetros retornados temos principalmente uma estrutura indicando se a aplicação deve continuar ou não, e o tempo em que o cliente deverá ficar repetindo esta operação. Se a implementação não utilizar eventos assíncronos, o cliente deve ficar repetindo esta operação neste intervalo de tempo especificado.

- Quando o cliente for acabar sua execução este deve liberá sua licença, invocando o método *end_use* (linha 4). Os parâmetros enviados nesta operação são basicamente o *token* da licença e informações de contexto.

3.8 Sumário

Quando um software é comprado de um vendedor legal, existe um contrato entre o vendedor e o comprador. A este contrato chamamos de licença. A pirataria de software pode ser definida como o uso ou cópia de software de forma a violar essa licença.

O gerenciador de licenças é um sistema distribuído que controla a utilização de produtos de software segundo os termos contidos na licença.

Dentre as várias especificações dos Serviços de Objetos da arquitetura OMA,

temos o Serviço de Licenciamento. A especificação do Serviço de Licenciamento define complemetamente as interfaces deste serviço. Estas interfaces são a *LicenseServiceManager* e a *ProducerSpecificLicenseService*. A primeira é usada para indicar qual objeto irá ser o responsável pelo controle do cliente e a segunda especifica as operações do objeto que irão controlar o cliente.

No próximo capítulo iremos descrever um projeto de um Serviço de Licenciamento, onde serão ressaltadas as características de tolerância a falhas.

Capítulo 4 - Projeto de um Serviço de Licenciamento CORBA

4.1 Introdução

Nos capítulos anteriores vimos como é estruturado o ambiente CORBA e como é a especificação de um gerenciador de licenças neste ambiente. Serão abordados neste capítulo os aspectos inerentes ao projeto de um gerenciador de licenças no ambiente CORBA. Neste projeto, o enfoque dado foi a continuação do funcionamento do sistema, mesmo que aconteçam comportamentos indesejáveis em alguns do seus componentes.

Estes comportamento indesejáveis são chamados de **falhas**. As falhas em um sistema distribuído podem se apresentar no hardware (componentes do sistema ou da rede de comunicação), no software ou terem causas externas ao sistema (ex: queda de energia, inundação, etc.). Se um sistema distribuído não está preparado para tratar a ocorrência falhas, podem acontecer sérios prejuízos aos usuários deste sistema.

Um componente do sistema é dito correto se para uma dada entrada produz uma saída que está de acordo com a sua especificação. Ou seja, uma saída é considerada correta se está dentro das expectativas do usuário e se esta foi entregue dentro de um limite de tempo pre-definido.

O sistema deve ser capaz de lidar com a ocorrência de falhas. No projeto de um sistema distribuído é importante que sejam previstos os elementos do sistema que possam apresentar problemas, quais os problemas e possíveis soluções.

Deve ser notado que a tolerância a falhas tem um custo: financeiro, degradação do desempenho, etc. Por isto é de fundamental importância que seja estabelecido qual o custo que se deseja pagar para aumentar a robustez do sistema.

Neste capítulo vamos apresentar um projeto de um Serviço de Licenciamento que provê características de tolerância a falhas.

4.2 Inclusão de Técnicas de Tolerância a Falhas no Servico de Licenciamento

A forma mais usual de incluirmos algum grau de tolerância a falhas no sistema é através de replicação. Porém a replicação tem um alto custo relativo a sua implementação, principalmente na manutenção da consistência no estado das réplicas.

A especificação da interface do serviço de licenciamento CORBA possibilita que seja utilizado na sua implementação mecanismos de tolerância a falhas. Estes mecanismos são basicamente estruturas que permitem verificar se os outros objetos continuam ativos.

Um exemplo é a operação *check_use* da interface *ProducerSpecificLicenseService*. Esta operação é invocada periodicamente pelo cliente no objeto que implementa a interface *ProducerSpecificLicenseService*. Esta operação possibilita que apenas uma ação do cliente possa detectar as seguintes condições inesperadas:

- Término inesperado do cliente ou do produtor.

- Falha ou particionamento da rede de comunicação entre o cliente e o produtor.

A forma como a operação *check_use* vai ser implementada é deixado a cargo do implementador.

4.3 Diagrama Esquemático do Serviço de Licenciamento

Vamos propor agora um Serviço de Licenciamento que continue seu funcionamento mesmo ocorrendo falhas em seus clientes e produtores. O Gerenciador de Licenças irá conseguir se recuperar basicamente de falhas por *crash* nestes

elementos. Analisaremos os problemas que podem ocorrer e como o sistema proposto irá lidar com estas falhas.

O nosso projeto utiliza os seguintes elementos:

LSM ⇒ License Service Manager – Responsável pela indicação e controle de todos os objetos que implementam a interface *ProducerSpecificLicenseService*, chamados de Producers (Pn)

Cn ⇒ Cliente do Serviço de Licenciamento – Aplicação protegida pelo serviço de Licenciamento

Pn ⇒ Producer Specific License Service – Elemento responsável pelo controle dos Clientes de mesmo índice n.

NS ⇒ CORBA Naming Service

PS ⇒ CORBA Property Service

F ⇒ Factory. Reponsável pela criação do objeto que implementa a interface *ProducerSpecificLicenseService* na máquina especificada pelo LSM.

Informações sobre Produtores ⇒ Armazena as informações sobre quem são os produtores responsáveis pelo clientes e quais destes produtores estão ativos e sua localização. Na tabela 4.2 mostramos um exemplo.

Produtor	Nº IP da Máquina onde está rodando o Produtor	Referência para o Produtor
	150.165.85.2	**(ref. para produtor 1)**
	150.165.85.4	**(ref. para produtor 2)**

Tabela 4.1 - Exemplo da Tabela Informações sobre Produtores

O **LSM** e o **Pn** são as implementações das interfaces do Serviço de Licenciamento CORBA mostrado no capítulo anterior. O **LSM** é a implementação da interface *LicenseServiceManager* e o **Pn** do *ProducerSpecificLicenseService*. O **LSM**

indicará qual **Pn** será responsável pelo controle do cliente. É bom lembrar que poderemos ter vários **Pn** executando na rede de forma simultânea, pois cada **Pn** representa o controlador de um determinado software. Em nossa notação por exemplo **P1** será o controlador de todos os clientes do tipo **C1**. Quando qualquer cliente do tipo **C1** requisitar do **LSM** um controlador, este informará a referência de um produtor do tipo **P1**.

O **NS** representa o serviço de nomes CORBA. Este serviço armazena uma estrutura do tipo (nome, referência para objeto). Isso possibilita uma recuperação mais fácil das referências para os objetos. Por exemplo, o **LSM** pode armazenar no serviço de nomes a seguinte estrutura (LSM, *ref. do objeto que implementa o LSM*). Assim quando qualquer objeto da rede necessitar a referência do **LSM** (para invocar um método no LSM por exemplo), ele chamará o serviço de nomes e enviará o nome "LSM" e o serviço de nomes retornará a referência para o objeto que implementa o Serviço de Nomes.

O **PS** representa o Serviço de Propriedades, o uso deste serviço é opcional na implementação do Serviço de Licenciamento CORBA. Este serviço possibilita o envio de propriedades entre objetos. Por exemplo, é possível enviar o ítem licence_use_context entre um objeto e outro. Neste ítem é possível enviar informações como número IP da máquina, hora local, nome do objeto etc. Ou seja, com o envio de apenas um parâmetro, é possível enviar a quantidade de informações de propriedades do objeto que seja necessária.

No esquema proposto, **F** representa uma estrutura chamada *Factory*. Esta estrutura é utilizada devido a impossibilidade de um objeto criar outro objeto em uma máquina diferente daquela da qual este objeto está sendo executando. Por isso, quando o **LSM**, por exemplo, necessita criar um objeto em outra máquina, ele não cria diretamente. O **LSM** chama um método da *Factory* que se localiza na máquina em que é necessária a criação do objeto e esta *Factory* cria o objeto. Em seguida a *Factory* retorna a referência do objeto criado ao objeto que a invocou.

O **LSM** armazenará a estrutura **Informações sobre Produtores.** Esta estrutura controlará onde estarão localizados os produtores e suas referências.

4.3.1 Esquema Completo do Funcionamento do Serviço de Licenciamento

Nesta seção iremos introduzir o projeto do Serviço de Licenciamento. Inicialmente iremos apresentar o esquema completo de funcionamento do Serviço de Licenciamento. Este diagrama pode parecer complexo, à primeira vista. Ele será apresentado nesta seção apenas para que seja possível a visualização completa do sistema. Em seguida iremos apresentar um esquema simplificado onde estarão indicados somente as partes do sistema que serão necessárias para o entendimento do nosso trabalho.

A Figura 4.1 mostra um exemplo de um sistema onde será analisada a continuidade de funcionamento após a ocorrência de falhas. Nesta figura estão representados uma série de computadores ligados em rede. Estão também apresentados os elementos de um Serviço de Licenciamento executando nestas máquinas.

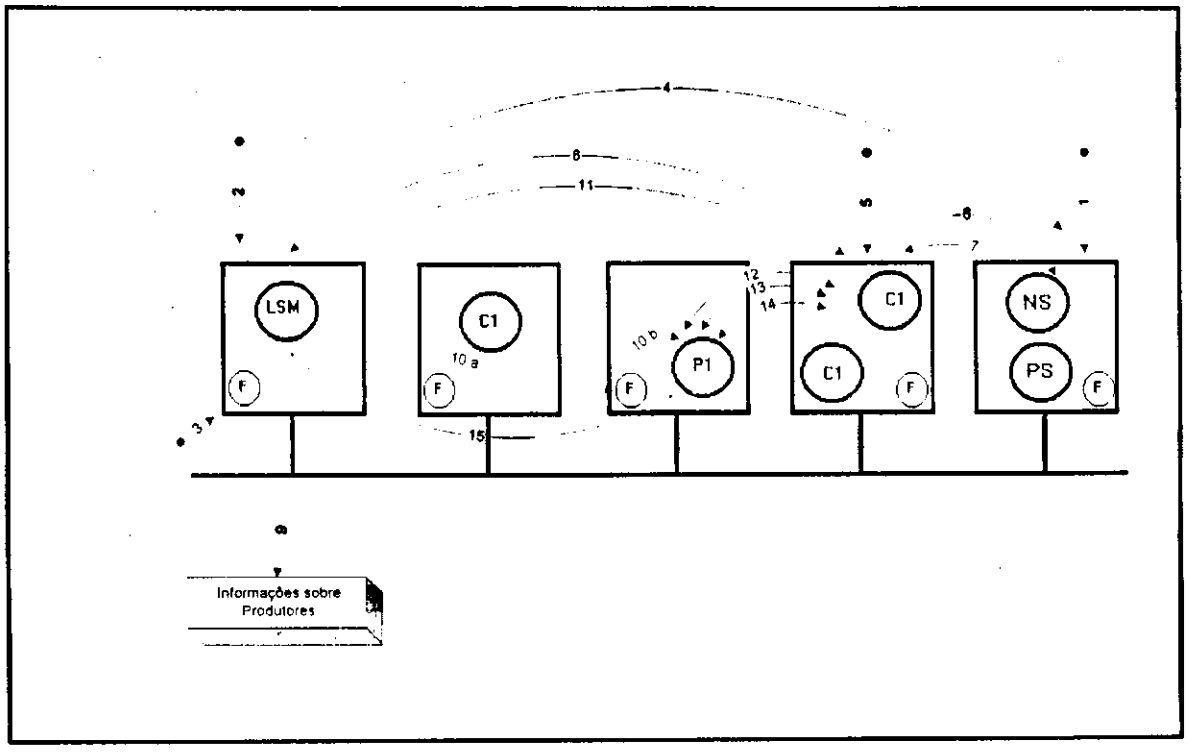

Figura 4.1 – Um esquema proposto para o Serviço de Licenciamento

Etapas da Figura 4.1:

1. Ativação do Naming Service e do Property Service .

2. Ativação do Programa Server.

3. Ativação da Factory em cada máquina da rede onde será executado o Serviço de Licenciamento (no esquema acima está mostrada apenas a ativação da Factory em uma máquina apenas para efeito de simplificação)

4. O Programa Server cria uma instancia do LSM e grava a sua localização no Naming Service.

5. Ativação do cliente.

6. Cliente vai ao Naming Service saber a referência para o LSM.

7. Naming Service resolve o nome enviado e retorna a referência ao cliente.

8. Cliente vai ao LSM e diz quem é, qual a máquina em que está sendo executado e pergunta ao LSM quem o vai controlar.

9. LSM lê uma tabela (Informações sobre produtores) e verifica se existe um produtor para esse cliente rodando na rede:

   - Caso não exista: é criado um produtor na máquina em que o cliente que solicitou a licença está rodando. Para que isto seja possível, o LSM chama um método na Factory localizada na máquina do cliente (passo 10a) e esta Factory cria o produtor (passo 10b). A referência deste objeto criado é armazenada na tabela.

   - Caso exista: A referência do objeto que implementa o produtor para este cliente é localizada na tabela.

11. A referência para esse produtor é retornada ao cliente.

12. Cliente vai agora ao seu controlador (chamaremos produtor) e chama a primitiva start_use.

13. Cliente faz o check_use inicial e recebe como um dos parâmetros o período em que irá ficar repetindo essa operação. Em seguida o cliente fica executando *check_use* neste intervalo de tempo especificado.

14. Quando o cliente for acabar sua execução este chama end_use para liberar esta licença.

15. Em um intervalo periódico de tempo, o produtor em questão vai ao LSM e verifica quem é o controlador dos seus clientes. Caso o LSM retorne uma referência diferente da referência atual deste produtor, o produtor deve abortar seu funcionamento.

4.3.2 Esquema Simplificado

Na Figura 4.2, vemos o esquema simplificado que usaremos para continuar a exposição das próximas etapas. Nele temos apenas a funcionalidade básica do Serviço de Licenciamento. Isto é tudo que precisamos para continuar a nossa exposição sobre as características de tolerância a falhas. Os passos que omitimos nesta figura são:

- Passos de 1 a 4 são realizados apenas uma vez, na inicialização do Serviço de Licenciamento. Uma vez executados o LSM fica executando à espera de algum chamado.
- Passos de 5 a 7 que são parte da ativação do cliente e busca da referência para o servidor.
- Passos 10a e 10b que são partes da criação dos produtores.

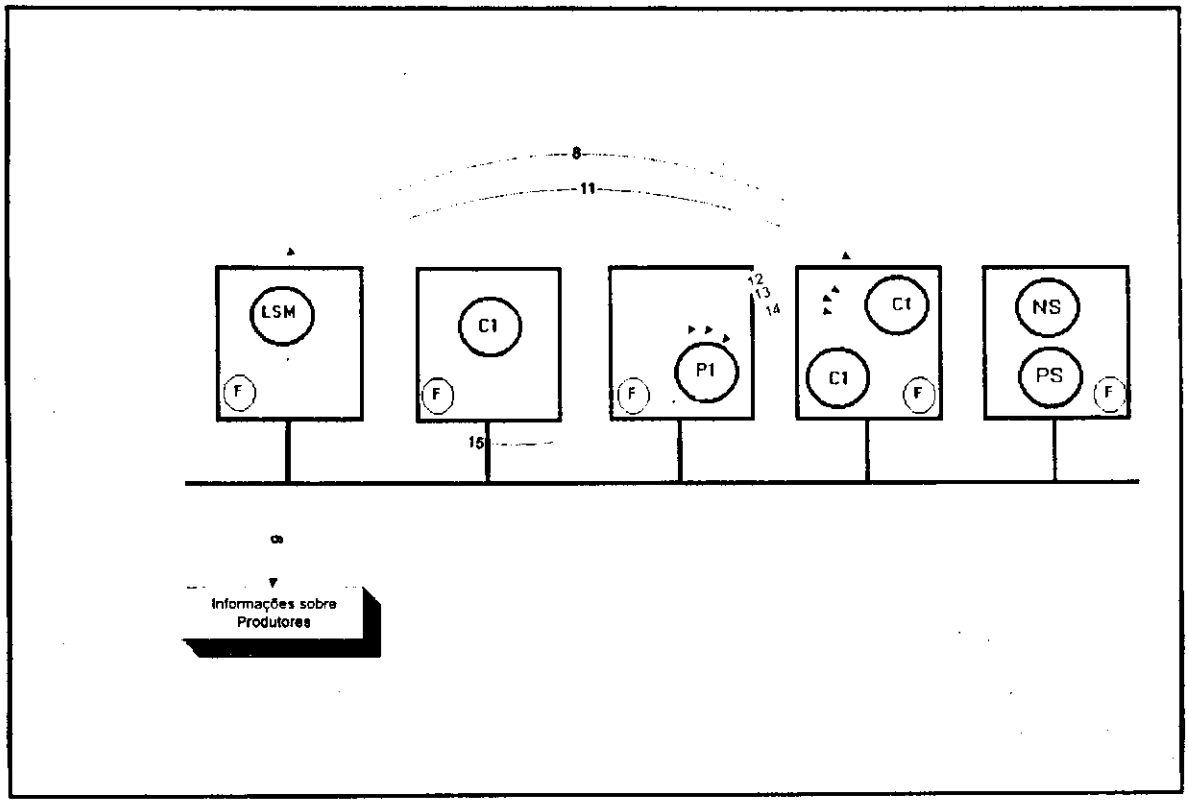

Figura 4.2 – Um Esquema Simplificado do Modelo de Gerenciamento

Para as nossas avaliações faremos as seguintes suposições iniciais:

1) O Naming Service, o Property Service, e o LMS estão situados em máquinas confiáveis e protegidas e não haverá falhas nestas aplicações.

Em seguida iremos apresentar um diagrama de seqüência (usando a notação UML – *Unified Modeling Language* [Booch97] [Booch99] [Fowler97]) que mostra as operações realizadas em um cenário normal de atividade. Este cenário mostra o comportamento dos principais componentes do sistema em um cenário livre de falhas.

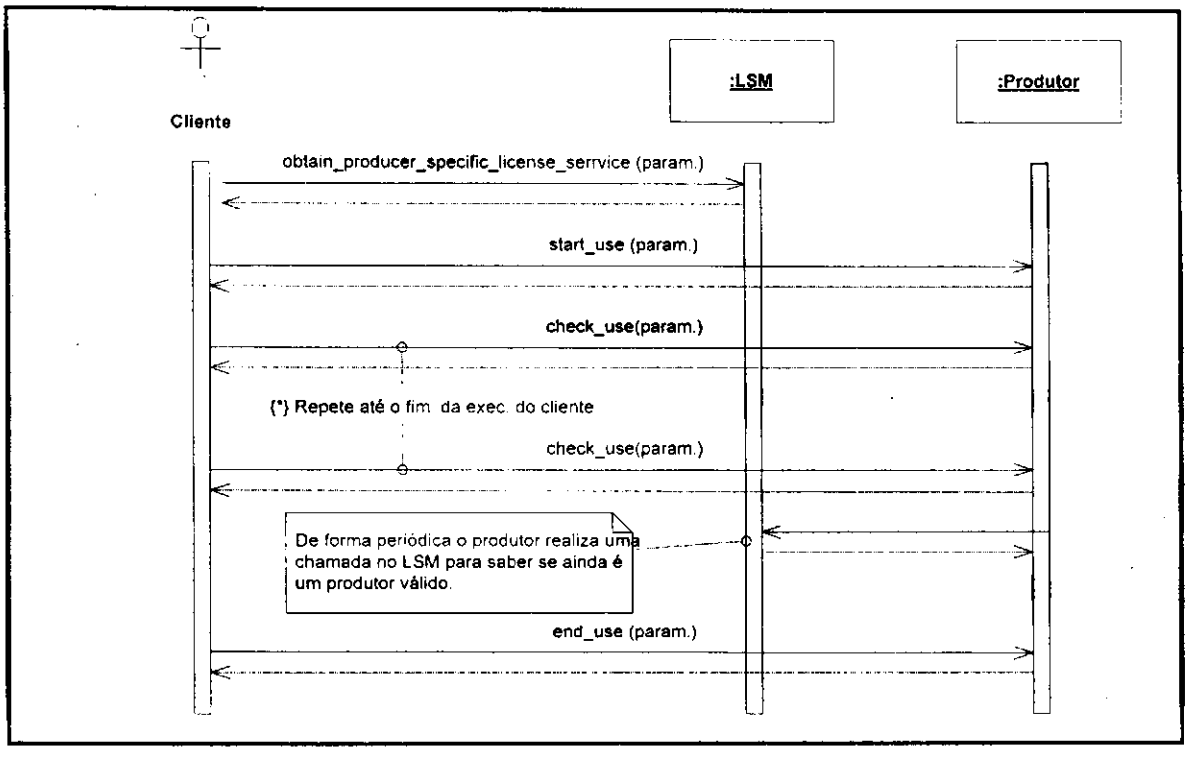

Figura 4.3 - Cenário sem Falhas

Neste cenário, mostrado na Figura 4.3, mostramos apenas os principais componentes do sistema: o Cliente do serviço de licenciamento, o LSM (objeto que implementa a interface *LicenseServiceManager*) e o produtor (objeto que implementa a interface *ProducerSpecificLicenseService*).

O Cliente chama a o operação *obtain_producer_specific_license_service* no objeto que implementa o produtor. O LSM localiza ou cria um produtor. Em seguida é retornado ao cliente a referência do objeto que implementa esta interface.

O Cliente de posse da referência do produtor pode agora executar a operação

start_use, onde este obtém um *token* de uma licença. O cliente agora chama a operação *check_use*. Esta operação retorna, entre diversos parâmetros, o tempo em que a operação *check_use* deve ser repetida. Esta operação é repetida até que o cliente necessite finalizar a aplicação. Neste momento é chamada a operação *end_use* (avisando ao produtor da liberação da licença).

Os parâmetros passados nesta operação podem ser vistos na especificação em IDL do Serviço de Licenciamento, que se encontra no Apêndice A desta dissertação.

4.4 Características de Tolerância a Falhas no Serviço de Licenciamento

Vamos agora analisar os possíveis problemas e soluções:

4.4.1 Falhas no Cliente

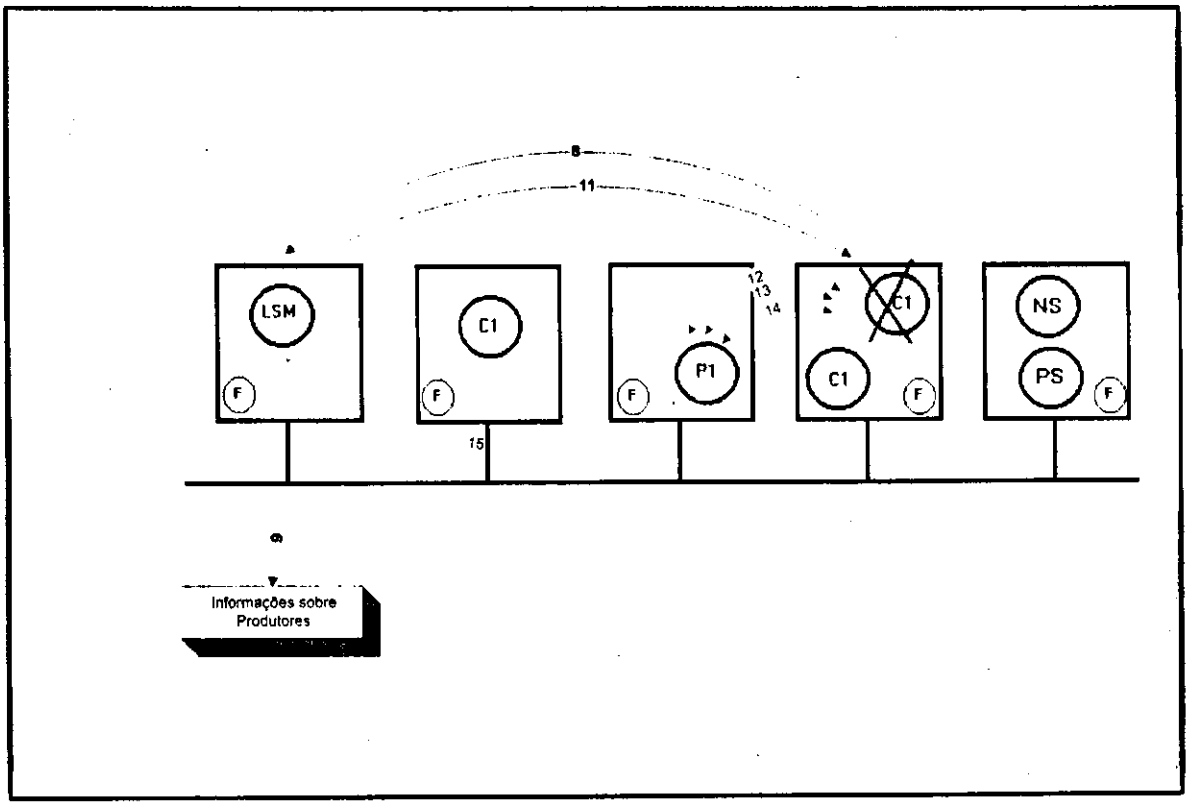

Figura 4.4 – Falhas no Cliente

- **Problema**: Licenças Pendentes.
- **Causa**: O cliente falhou e por isso não liberou sua licença.
- **Detecção do Problema**: A primitiva *check_use* retorna ao cliente o intervalo que indica o período em que essa operação deve ser executada. Se após o período de tempo especificado (poderemos aqui incluir uma margem de erro para o caso do servidor estar sobrecarregado ou outros fatores), o cliente não executar a primitiva *check_use*, o produtor considerará que este cliente não está mais funcionando.
- **Atitude a ser Tomada:** O produtor cancelará assim a licença concedida a este cliente.

Na Figura 4.5 apresentamos o cenário que ocorre quando existe a falha no cliente.

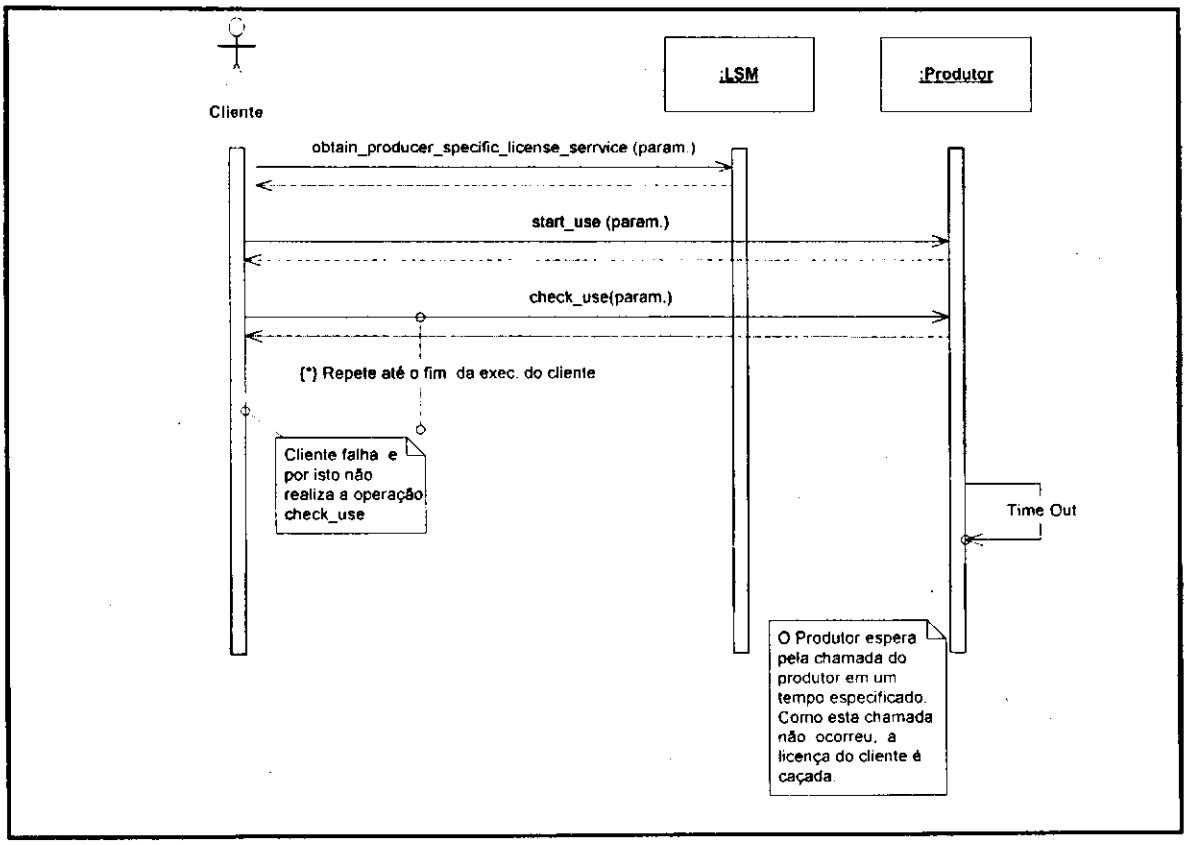

Figura 4.5 – Cenário na Presença de Falhas no Cliente

4.4.2 Falhas no Produtor

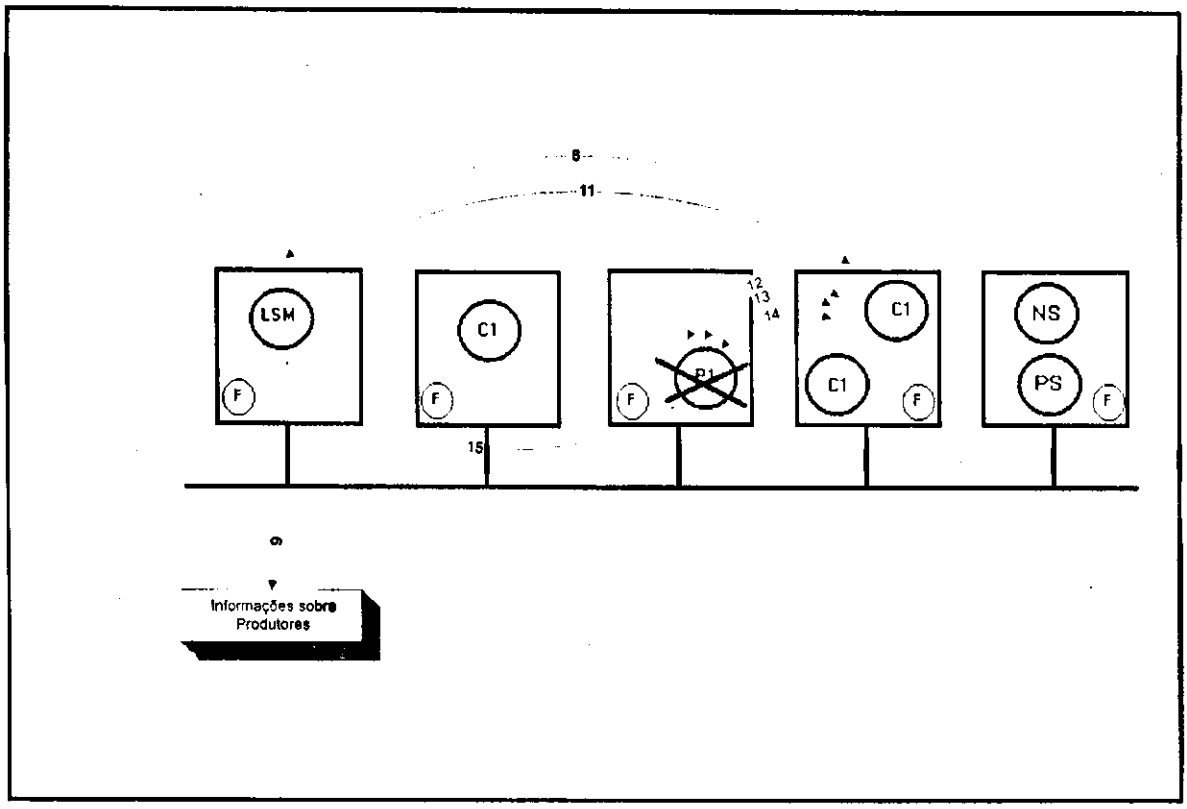

Figura 4.6 – Falhas no Produtor

- **Problema:** Clientes não tem com validar suas licenças
- **Causa:** Falha no Produtor
- **Detecção do Problema:** Quando o cliente não recebe o retorno de qualquer operação feita no produtor dentro de um certo intervalo de tempo, podemos supor que o produtor falhou.
- **Atitude a ser Tomada:** A atitude a ser tomada irá depender de:

1) Cliente Consegue se comunicar com o LSM: Cliente consegue se comunicar com o LSM: Neste caso o cliente irá ao LSM e pedirá uma nova referência para um produtor, indicando neste pedido que o produtor anterior falhou. O LSM criará um novo produtor para este

cliente e atualizará a tabela Informações sobre os Produtores. Este cenário pode ser visto na figura 4.7.

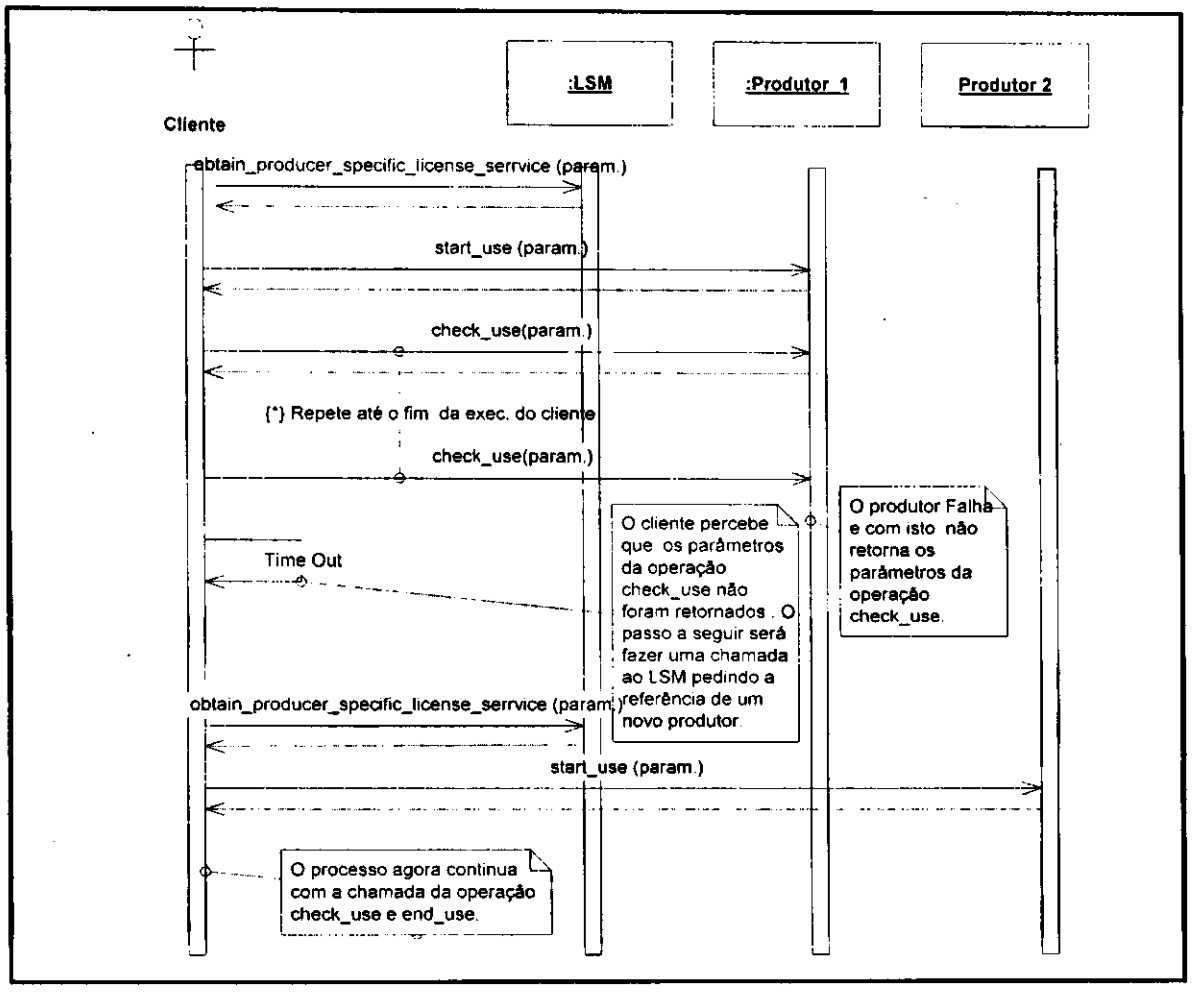

Figura 4.7 – Falhas no Produtor e Comunicação entre Cliente e LSM Possível.

2) Cliente não consegue se comunicar com o LSM: O Cliente deve abortar o seu funcionamento. Este cenário pode ser visto na Figura 4.8.

Caso o produtor não tenha falhado, ele descobre depois de algum tempo que alguém "suspeitou" indevidamente de sua falha e portanto ele deve abortar. O produtor obtém esta informação através do passo 15 do esquema mostrado

na Figura 4.1, onde ele periodicamente vai ao LSM, indagando qual é produtor que controla seu tipo de cliente. Se o LSM retornar a referência deste produtor, ele sabe que ele continua sendo o controlador. Se o LSM retornar outra referência, ele deve abortar.

Uma vez que se trata de um sistema assíncrono, não podemos ter certeza se o produtor realmente falhou ou não. No caso temos apenas uma "suspeita".

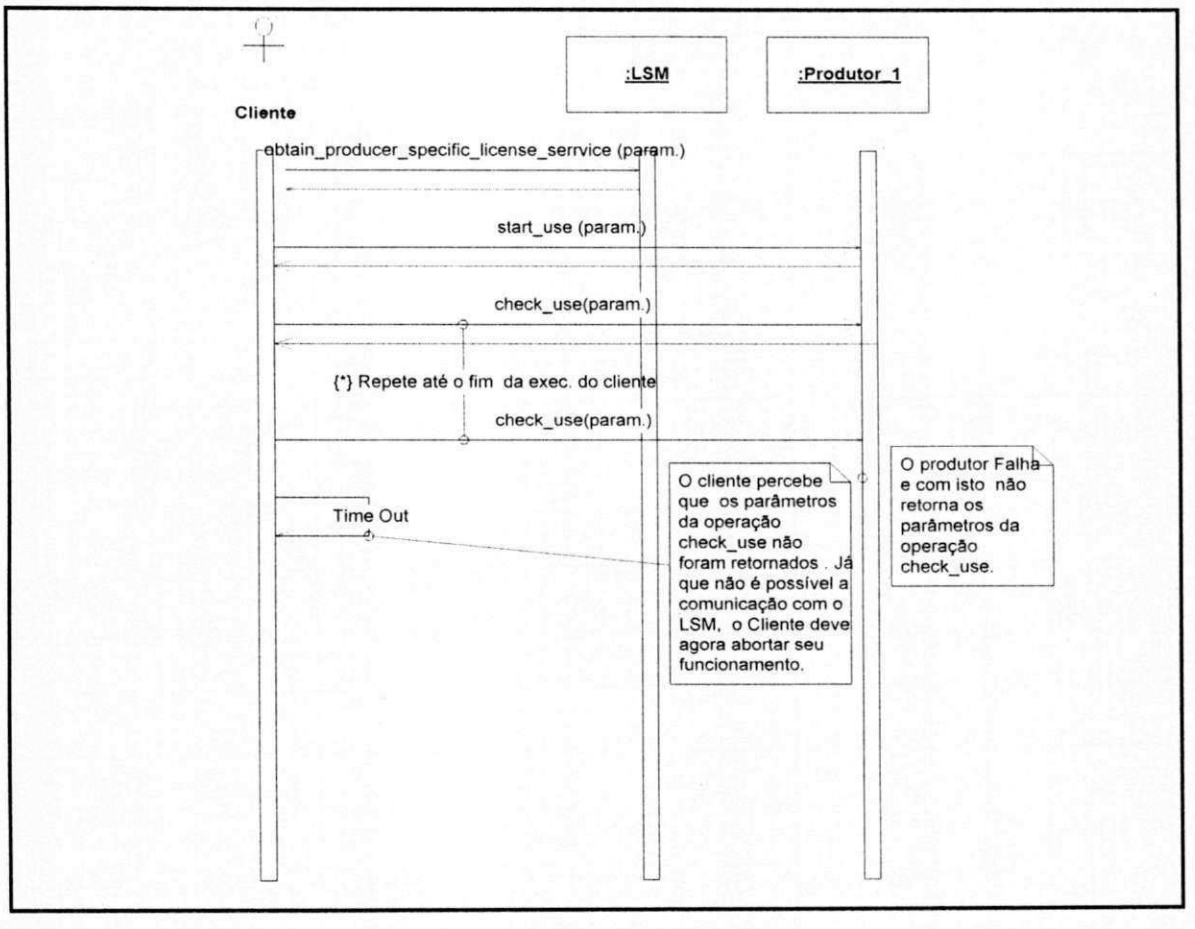

Figura 4.8 – Falhas no Produtor e a Comunicação entre Cliente e Produtor não é Possível

4.4.3 Particionamento Entre o Cliente e o Produtor

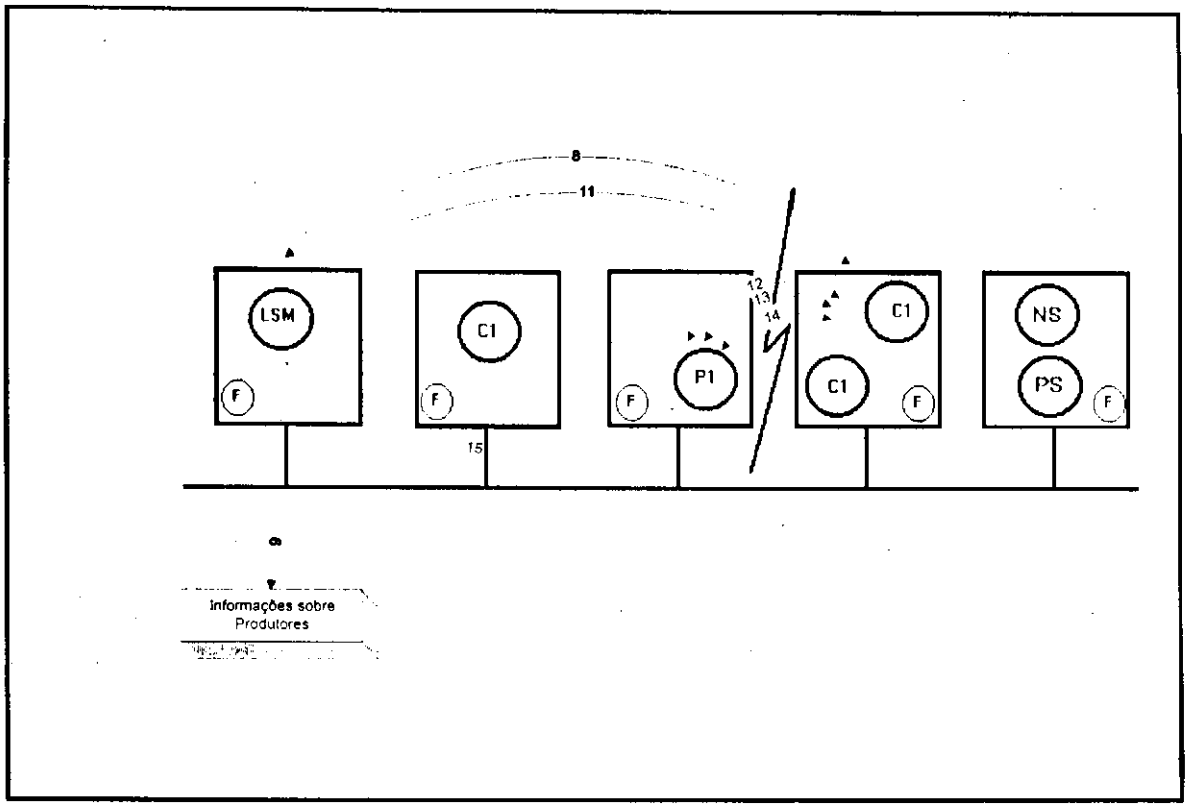

Figura 4.9 – Particionamento da Rede

- **Problema:** Cliente irá suspeitar que o seu produtor falhou. Produtor irá suspeitar que cliente falhou.
- **Causa:** Problemas na rede de comunicação.
- **Detecção do Problema:** O sistema não sabe que houve um particionamento. Do ponto de vista do cliente, o produtor falhou. Do ponto de vista do produtor, o cliente falhou.
- **Atitude a ser Tomada:** Devido ao exposto no ítem anterior , a atitude a ser tomada é análoga à das seções 4.4.1 e 4.4.2.

4.4.4 Particionamento entre o Produtor e o LSM

Na Figura 4.10 é mostrado um particionamento entre o produtor e o LSM.

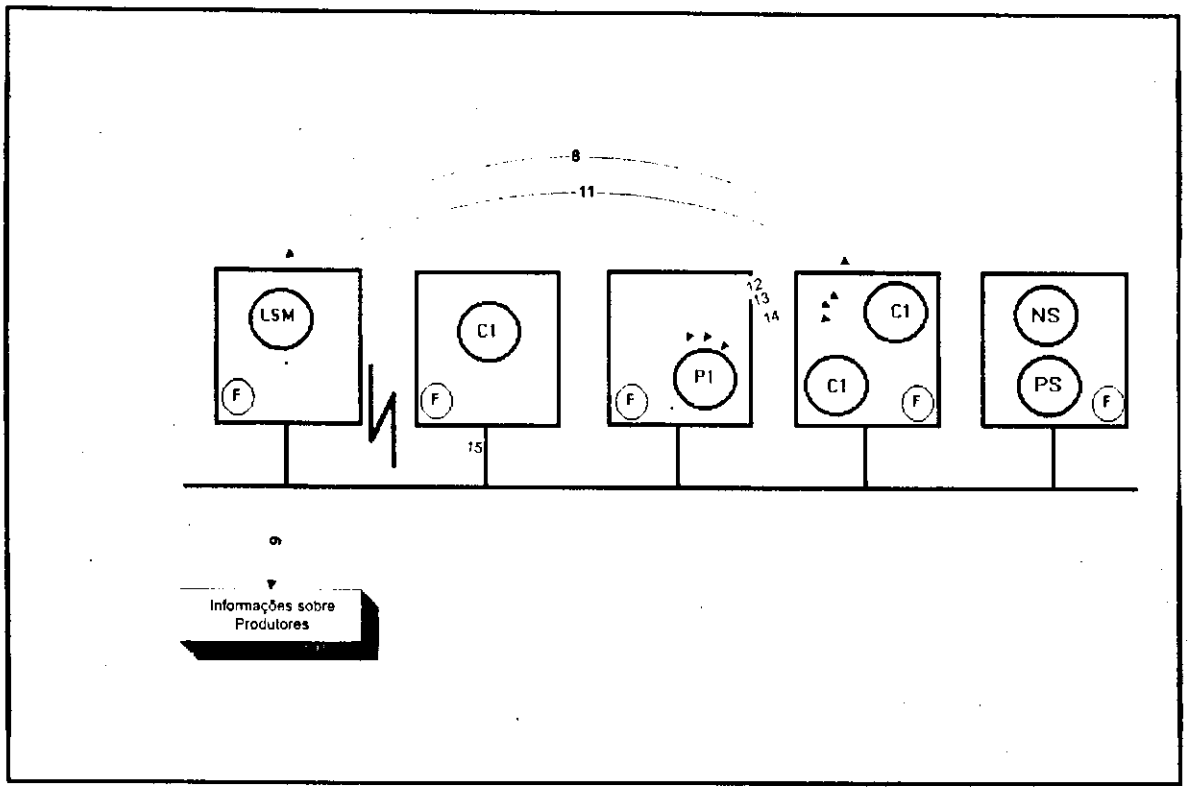

Figura 4.10 – Particionamento entre o Produtor e o LSM.

- **Problema**: Produtor não tem como saber se ele ainda é um produtor válido para o LSM.
- **Causa**: Problemas na rede de comunicação.
- **Detecção do Problema**: O LSM não consegue retorno da operação que visa identificar se este é ainda um produtor válido (passo 15) da Figura 4.1.
- **Atitude a ser Tomada:** Neste caso, a atitude a ser tomada vai depender da política de licenciamento.
    - Em uma política de licenciamento que privilegie a continuidade do serviço, o produtor deverá ignorar que não conseguiu se comunicar com o LSM e continuar tentando chamar o LSM no intervalo especificado. Quando acabar o particionamento, o produtor irá contatar o LSM e caso tenha sido criado outro produtor no outro lado do particionamento

da rede, este deve abortar. O lado negativo desta solução é permitir temporariamente a existência de dois produtores na rede (um em cada segmento da rede). Isto temporariamente aumentará o número de licenças que podem ser concedidas.

- Se a política de licenciamento enfatizar a segurança, quando o produtor não conseguir se comunicar com o LSM, este deve abortar. Assegurando assim, que não haja mais de um produtor ativo na rede. O lado negativo desta solução é impedir a continuação da execução dos clientes que ficaram isolados do LSM.
- Estes cenários podem ser vistos na Figura 4.11.

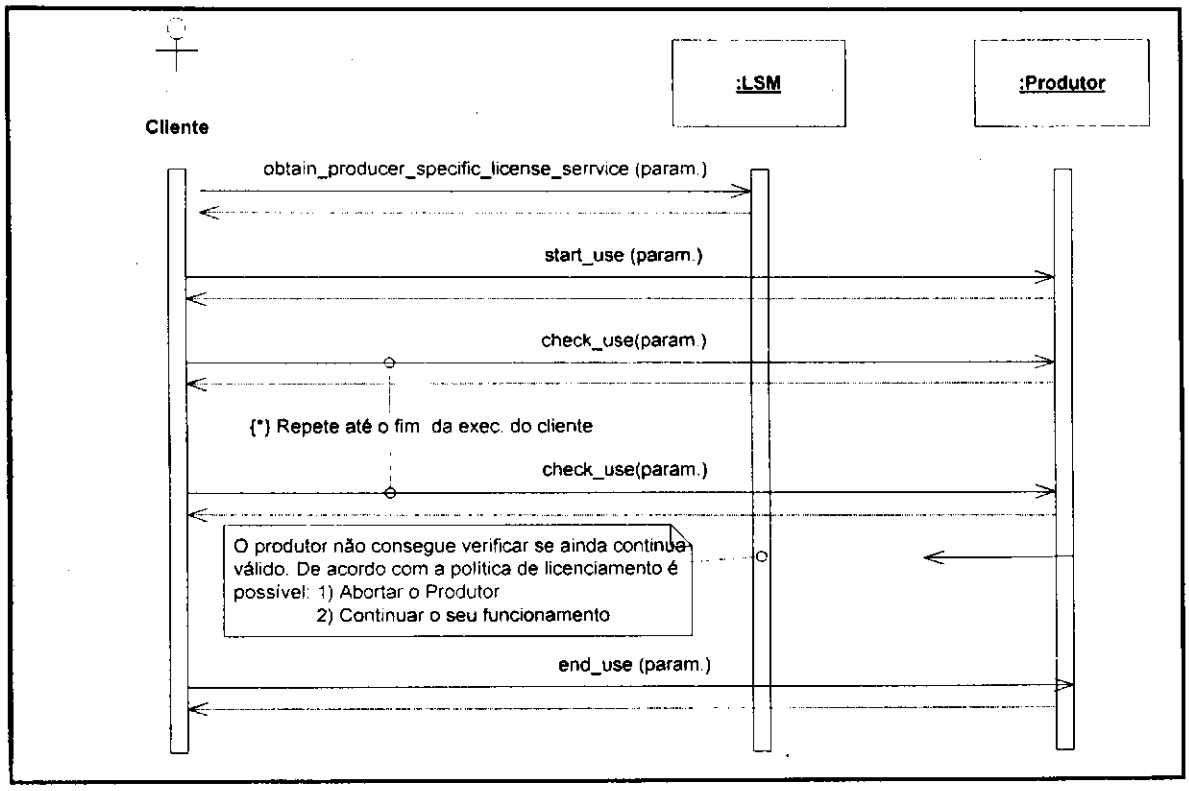

Figura 4.11 – Cenário no Particionamento entre o Produtor e o LSM.

4.4.5 Particionamento entre o Cliente e o LSM

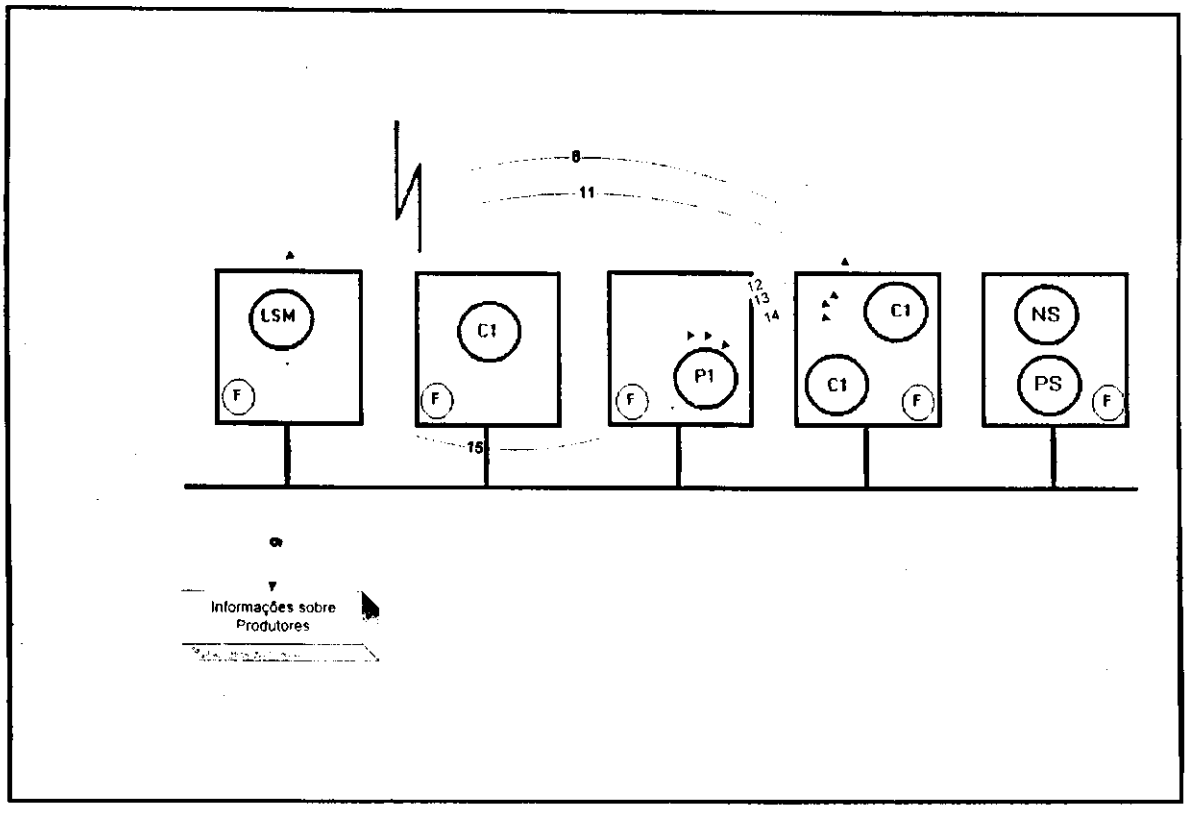

Figura 4.12 – Particionamento entre o cliente e o LSM

- **Problema**: Não serão fornecidas novas licenças.
- **Causa**: Problemas na rede de comunicação.
- **Detecção do Problema**: O cliente não obterá retorno da operação *obtain_producer_specific_license_service.*
- **Atitude a ser Tomada:** Caso o cliente já tenha obtido a sua licença, ele continuará funcionando normalmente. Caso contrário, ele não poderá obter uma licença .

4.4.6 Um Cliente, LSM ou Produtor Falso

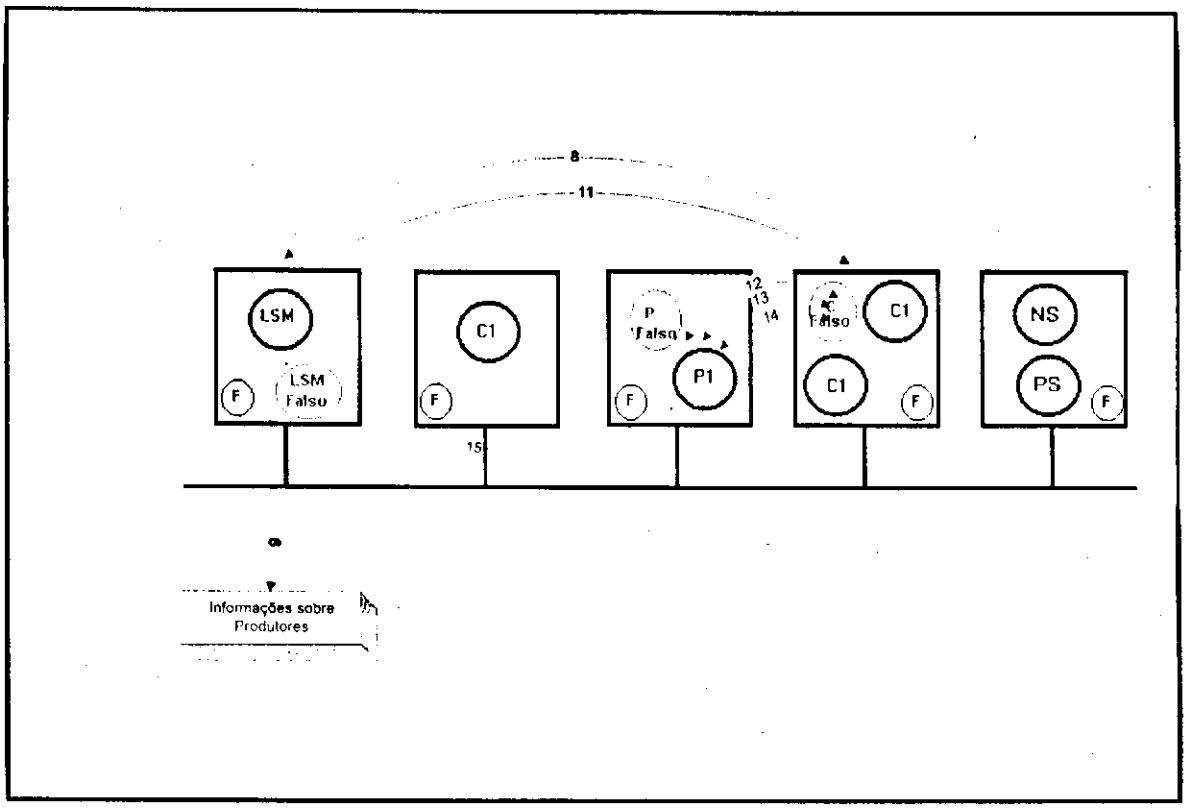

Figura 4.13 – Elementos Falsos tentam ingressar no sistema

- **Problema**: Neste caso teremos vários problemas, entre eles: concessão de licenças indevidas, perturbação no sistema devido a comunicação indevidas, etc ...
- **Causa**: Elementos falsos ingressando no sistema.
- **Como Evitar:** Todas as mensagens trocadas entre os elementos do sistema usam o mecanismo de autenticação *Challenge*, especificado no Serviço de Licenciamento CORBA. Outra alternativa é a utilização de um Serviço de Objetos CORBA. No caso o *Security Service.*

Sumário

Neste capítulo foi visto o projeto de um Sistema de Licenciamento no ambiente CORBA com características de Tolerância a Falhas.

Foram analizadas várias possibilidades de ocorrências de problemas no sistema e como o sistema irá lidar com estes, continuando assim normalmente o seu funcionamento. O sistema proposto irá garantir a continuidade de funcionamento do sistema, mesmo em face da ocorrência de problemas nos seus clientes e nos controladores destes (os produtores).

Capítulo 5 – Aspectos da Implementação de um Serviço de Licenciamento CORBA

5.1 Introdução

Neste capítulo abordaremos os aspectos relativos à implementação do Serviço de Licenciamento descrito no capítulo anterior. Descreveremos os passos dados, as escolhas feitas no decorrer desta implementação e as características inerentes à implementação do sistema.

Além do Serviço de Licenciamento propriamente dito, foi implementado um protótipo de um "produtor" que implementa a política do *Controle de Uso Concorrente de Software*. Como explicado anteriormente no capítulo 3, essa política de licenciamento irá controlar o número máximo de ativações simultâneas de um determinado programa em uma rede de computadores.

5.2 O Ambiente Utilizado

5.2.1 O ORB

O ORB usado em nossa implementação e testes foi o **ORBacus for Java** (antigo OmniBroker) da OOC (*Object Oriented Concepts*) [OOC99a]. A versão utilizada foi a 3.1.2. Dentre as suas características mais importantes temos:

- Licença de uso gratuita para fins não comerciais.
- Completo código fonte disponível.
- Suporte total a CORBA IDL.
- Suporte completo à especificação do mapeamento CORBA *IDL-to-Java*..

- Implementação dos Serviços de Objetos: Nomes, Eventos e Propriedades.
- Completa compatibilidade com a especificação CORBA 2.0.
- Suporte a vários protocolos de comunicação inter-ORBs; utilizando o IIOP como padrão.
- Suporte à *Multi-Threading.*
- Completo suporte a programação dinâmica.
- Suporte de mecanismos de *timeouts.*

As características acima, principalmente o fato deste produto ter o código fonte disponível (possibilitando com isso a compilação para plataformas como Linux e Windows NT, por exemplo) e de incluir os serviços de Nomes, Propriedades e Eventos, justificaram a sua escolha.

O Serviço de Nomes é bastante útil para a grande maioria das aplicações CORBA. Os Serviços de Propriedades e Eventos estão incluídos na especificação do Serviço de Licenciamento CORBA, sendo utilizados a critério do implementador para aumentar a sua funcionalidade.

Outro fato que motivou a escolha deste produto foi o suporte gratuito fornecido pela empresa desenvolvedora através de lista de discussão.

5.2.2 Linguagem de Programação

Foi utilizada a linguagem de programação Java. Esta linguagem é considerada uma das melhores para se implementar aplicações CORBA. Dentre as características principais desta linguagem temos:

- Portabilidade entre plataformas
- Programação para Internet
- Linguagem orientada a objeto
- Modelo de componentes

Vamos analisar mais detalhadamente estas características:

5.2.2.1 Portabilidade entre Plataformas

Os programas em Java são altamente portáteis devido ao seu compilador gerar uma representação padronizada e independente de máquina chamada de *byte-code.* Atualmente já existe um grande número de ferramentas disponíveis (que interpretam *byte-code)* para várias plataformas de hardware e sistemas operacionais.

Isto é uma vantagem significativa em relação às outras linguagens de programação, em particular para o desenvolvimentos de clientes, uma vez que o código-fonte ou o *byte-code* podem ser utilizados em quaisquer plataformas sem que seja necessário nenhuma modificação. Consequentemente, os custos de manutenção podem ser significativamente reduzidos [Vogel98].

5.2.2.2 Programação para Internet

A linguagem de programação Java permite a implementação de clientes CORBA como *applets.* Isto permite o acesso aos objetos CORBA através de *Web Browsers.* Estes *Browsers* estão se tornando um padrão para clientes que usam GUI (*Graphical User Interface).* Isto possibilita, por exemplo, a padronização de todos os clientes dentro de uma corporação, facilitando assim as atualizações e modificações de softwares, que neste caso, só precisam serem efetuadas no servidor.

5.2.2.3 Linguagem Orientada a Objeto

Java possibilitada que seja utilizada todas as vantagens da orientação a objeto de maneira mais simples que C++. O programador tem menos responsabilidades no gerenciamento de memória, não tem ponteiros para manipular, tem uma sintaxe mais simples, etc.

Além disso, Java apresenta características adicionais como o "Coletor de Lixo" automático, tratamento de exceções e suporte nativo a *threads.*

5.2.2.4 Modelo de Componentes

O modelo de componentes permite ao programador combinar a funcionalidade de um grande número de classes Java em um único componente. Estes componentes são chamados de *Java Beans*. Estes componentes podem ser facilmente colocados juntos, mesmo por pessoas sem experiência em programação, para adquirirem uma nova funcionalidade.

5.2.3 Compilador e Sistema Operacional

Usamos o Compilador JDK (*Java Development Kit*) da Sun Microsystems, Inc. [Sun99]. A versão utilizada foi a 1.1.7 para o Sistema Operacional Linux. Este compilador foi portado para Linux pela organização Blackdown [Black99].

O Sistema Operacional Utilizado foi o Linux Conectiva Guarani (versão do Linux RedHat), com o kernel 2.0.36 [Conectiva99].

5.3 Aspectos Relativos ao Desenvolvimento

5.3.1 Especificação em IDL do Serviço de Gerenciamento

O desenvolvimento do sistema iniciou-se com a IDL do serviço de licenciamento (módulo CosLicensingManager.idl mostrado no Apêndice A). Este módulo é a interface padronizada do Serviço de Licenciamento no ambiente CORBA. Ele está disponível no *Web Site* da OMG [OMG99].

A este módulo foi adicionado para a nossa implementação a interface Factory (podemos ver esta etapa no arquivo CosLicensingManager2.idl mostrado no Apêndice B). Este arquivo adiciona uma interface ao Serviço de Licenciamento. Essa funcionalidade será útil em nossa aplicação para que seja possível a criação de objetos em máquinas remotas, como veremos adiante. É importante observar que o acréscimo

de funcionalidade à IDL não fere a especificação do padrão.

5.3.2 A Compilação da IDL e a Geração dos *Stubs*

O próximo passo foi a compilação da IDL e a geração dos *stubs* em linguagem de programação Java. Isto é realizado pelo programa *jidl*.

O programa *jidl* gerou uma série de códigos-fonte em Java que foram a base da nossa implementação. Esses programas gerados irão tornar transparentes para o programador os aspectos inerentes à comunicação entre os objetos.

5.3.3 Implementação

Agora que a base para o sistema está pronta, foi iniciada a implementação das interfaces:

5.3.3.1 Programa Server

Este programa é responsável pela ativação do LicenseServiceManager e armazenamento da referência para este no Serviço de Nomes. Isto é feito ativando a classe que implementa o LicenseServiceManager (LicenseServiceManager_impl).

5.3.3.2 Implementação da Interface LicenseServiceManager

Esta interface indica ao cliente quem será o produtor responsável pelo seu controle. Em nossa implementação, o arquivo que contém código-fonte desta interface é o LicenseServiceManager_impl.java. Esta interface possui apenas um método o *obtain_producer_specific_license_service()*. Este método recebe um tipo String e retorna uma referência para um objeto que implementa a interface *ProducerSpecificLicenseService*.

Este arquivo herda as definições do _LicenseServiceManagerImplBase.java um dos *stubs* criados para esta interface.

As atividades principais deste programa são as seguintes:

1. Inicialmente são separados os parâmetros recebidos da String fornecida pelo Cliente. Os parâmetros são: nome do cliente, nome da máquina, endereço IP e uma indicação de erro (caso tenha acontecido erro com o produtor anterior).

2. Em seguida é criada a matriz Informações sobre Produtores (caso ela ainda não tenha sido criada) que contém as informações sobre produtores e a referência para estes.

3. É verificado na tabela se já existe um produtor para este cliente na rede. Caso haja, esta referência é retornada ao cliente. Caso não haja, é chamada um método na Factory da máquina do cliente, para que um produtor seja criado nesta máquina. A Factory cria um objeto na máquina do cliente (ativando assim um objeto que implementa a interface *ProducerSpecificLicenseService*) e retorna esta referência ao LSM. O LSM retorna agora esta referência ao cliente.

Em seguida, na Figura 5.1, mostraremos (usando português estruturado) o algoritmo utilizado na implementação desta interface. Neste algoritmo estarão presentes apenas os principais passos e interações realizados por esse módulo.

- Inicio
- Recebe ⇐ LSM
  Operação: obtain_producer_specific_licence_service
  Parâmetros: Uma cadeia de caracteres com o formato:
  (nome do cliente/nome do host/numero IP)
- Separa esta cadeia isolando seus termos.
- Ler arquivo de configuração (onde estão armazenadas os números Ips das máquinas da rede autorizadas a utilizarem o Serviço de Licenciamento)
- Verifica se existe a TabeladeReferências (armazena informação sobre quais produtores estão rodando em quais máquinas da rede).
  Caso não exista: inicialize esta tabela.
- Procura pelo produtor associado a este cliente.
- Procura na TabeladeReferencias pelo nome do produtor.
- Verifica se já existe alguma referência de objeto associado a este produtor.

  Caso já exista – Esta referência é retornada ao cliente
  Caso não exista – É necessário criar um produtor e armazenar sua referência na tabela.

  Criação do Produtor: - Localiza no Serviço de Nomes a Ref. para a Factory localiza na máquina do cliente.
  Envia ⇒ Factory
  Operação: createProducer(nomedoprodutor)
  Recebe: producer (Ref. para este produtor criado pela factory)
  Retorna referência ao cliente.

- Fim

Figura 5.1 – Algoritmo do LSM

5.3.3.4 Implementação da Interface *ProducerSpecificLicenseService*

São vários os programas que implementam a interface *ProducerSpecificLicenseService*. Como poderemos ter vários produtores com características diferentes, cada produtor será uma implementação desta interface.

Como exemplo de implementação desta interface temos a classe Producer1_impl. A principal atividade desta classe é fornecer métodos para que os clientes façam seus pedidos de licenças. No nosso protótipo, implementamos esta classe com a política de licenciamento de *controle do uso concorrente de software*. Esta

política de licenciamento controla o número de ativações concorrentes de um determinado software em uma rede. Esta classe é composta de uma série de métodos: *producer_contact_info, producer_specific_license_service_info, start_use, check_use, end_use.* Os dois primeiros métodos servem apenas para retornar ao cliente informações sobre o produtor, tais como, telefone para contato, etc. Vamos agora analisar o comportamento dos métodos principais desta interface: *start_use, check_use* e *end_use.*

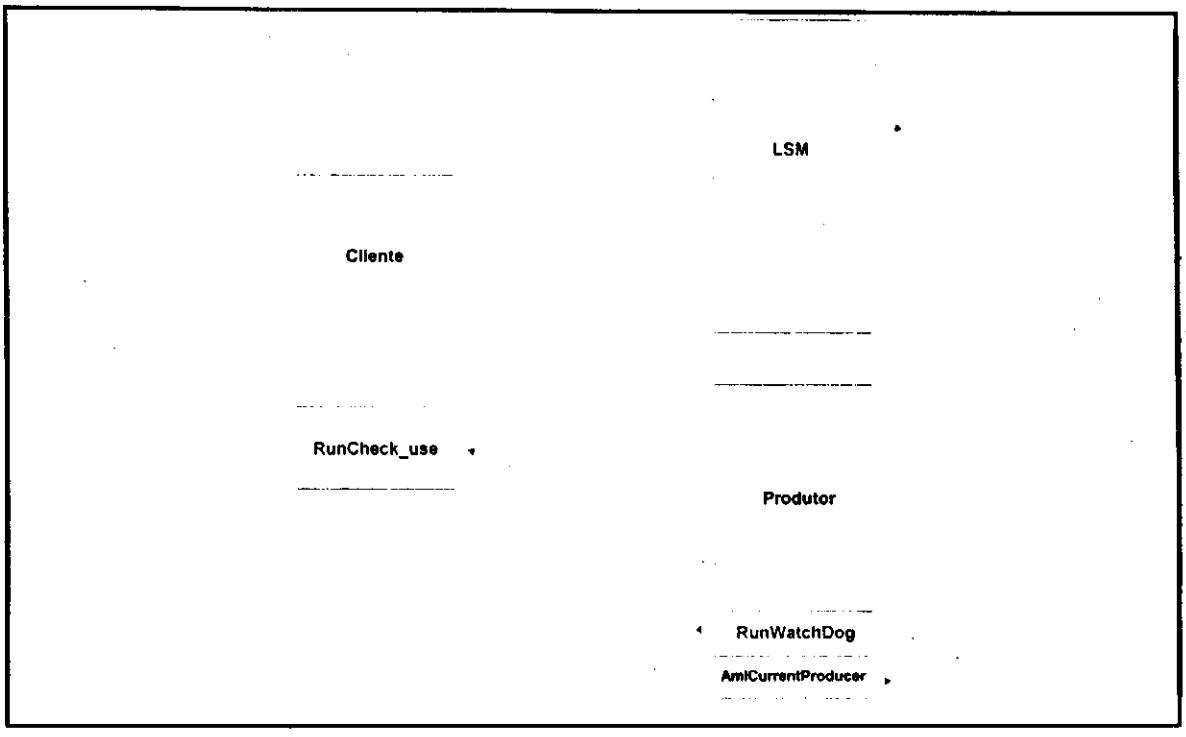

Figura 5.2 – Comunicação entre as *threads*

1. Quando o cliente chama *start_use* e envia como parâmetros: seu nome, versão, informações de contexto, e uma estrutura para autenticação. Este método verifica se o cliente é um elemento válido do serviço de licenciamento. Em caso arfirmativo é retornado um *token* (um elemento que identifica unicamente) uma licença.

2. Quando o método *check_use* é chamado, este recebe o *token* fornecido pelo cliente na operação *start_use*, e verifica se existem licenças disponíveis. Caso não haja licenças disponíveis, a operação retorna um parâmetro indicando que o cliente deve terminar (o cliente então termina ou bloqueia,

dependendo da política de licenciamento). Caso haja licenças disponíveis, é retornado ao cliente a informação para continuar e o tempo que especifica o período em que o cliente deve ficar repetindo a invocação do método *check_use*. É então criado uma *thread* que funciona como um *watchdog*. Esta *thread* é chamada de RunWatchDog como podemos ver na Figura 5.2. Esta *thread* é ativada periodicamente para verificar se o cliente está chamando a operação *check_use* neste produtor, no tempo especificado. Caso esta *thread* perceba que a operação *check_use* não foi chamada, esta considera que o cliente morreu. A *thread* então avisa ao produtor para caçar a licença concedida a este cliente.

3. Quando é chamada o método *end_use*, esta irá cancelar a licença concedida ao cliente.

Neste módulo temos também uma *thread* que periodicamente vai ao *LicenseServiceManager* (LSM) e verifica se este ainda é um produtor válido para o seu tipo de cliente. Isto é feito de forma análoga ao modo como um cliente interage com o LSM, ou seja, o produtor informa que é um cliente do tipo X, e indaga qual a referência para um produtor que responde por este tipo de cliente. Como podemos ver na Figura 5.2 esta *thread AmICurrentProducer* é encarregada de implementar esta funcionalidade. Se o produtor retornar a sua própria referência, este é um produtor válido e pode continuar funcionando. Se o produtor retornar outra referência, este produtor não é mais um produtor válido e deve abortar (isto irá ocorrer, por exemplo, se este produtor tiver tido problemas na comunicação com seus clientes, e por isso o LSM criou outro produtor).

Este algoritmo pode ser visto na Figura 5.3. Lembrando que no algoritmo apresentamos apenas os elementos fundamentais (operações, parâmetros) da implementação.

- Inicia
- Inicia *Thread* AmICurrentProducer
- Espera ser chamado

Se operação *start_use* for chamada.

Recebe ⇐ Cliente

Operação: start_use
Parâmetros: recebe (nome do componente, versão)
Retorna: *token* para uma licença

Verifica se é possível conceder uma licença
Caso afirmativo: conceder *token* válido
Caso contrário: conceder *token* inválido

Se a operação *check_use* for chamada

Recebe ⇐ Cliente

Operação: check_use
Parâmetros: recebe (*token*)
Retorna: *recommended_check_interval:* tempo para continuar repetindo esta operação periodicamente .
Action: Estrutura indicando se a continuação da execução do cliente é permitida Ou não.

Se *token* for válido:

Se for a primeira vez que o Cliente chama esta operação
então iniciar *Thread* RunWatchDog

Envia ⇒ RunWatchDog (*token*,tempo de repetição)

Verifica se licença continua válida
Caso afirmativo: valida a licença
Caso negativo: nega a licença

Se *token* não for válido: retornar informação negando a licença

Se operação *end_use* for chamada.

Recebe ⇐ Cliente

Operação: end_use
Parâmetros: recebe (*token*)

Registra a informação que a licença do cliente finalizou sua execução
Encerra *Thread* RunWatchDog

Se a Thread *RunWatchDog* for chamada:

Recebe ⇒ RunWatchDog (*token*,tempo de repetição)
No tempo de repetição recebido verifica se cliente chamou a operação *check_use*

Se a *Thread AmICurrentProducer* for chamada:
Em um intervalo de tempo periódico verificar se o produtor continua válido.

- Fim

Figura 5.3 – Algoritmo do Produtor

5.3.3.5 Implementação da Interface *Factory*

A implementação desta interface (Factory_impl) basicamente disponibiliza para o LSM o método *create_producer* para que seja criado um objeto que implementa a interface *ProducerSpecificLicenseService* na máquina do cliente. Em seguida a referência para o objeto criado é retornada ao LSM

```
- Inicia
- Recebe ⇐ LSM
      Operação: createProducer
      Recebe: nome_do_produtor
      Retorna: Referência para este produtor criado

      Se createProducer for chamada
            Crie Produtor
            Retorne referência do produtor criado ao LSM
```

Figura 5.4 – Algoritmo da Factory

5.3.3.6 Implementação do Cliente

Agora vamos mostrar a implementação de um cliente do Serviço de Licenciamento. As atividades desenvolvidas por este cliente basicamente são:

1) cliente busca no serviço de nomes a referência para o objeto que implementa o LSM.
2) cliente chama o método *obtain_producer_specific_license_service* no LSM enviando como parâmetro em uma *String* as informações: nome, máquina, endereço IP, ocorrência de erro (caso o produtor anterior tenha falhado).
3) Com a referência recebida do LSM, o cliente chama o método *start_use* no produtor.
4) Em seguida chama o método *check_use e* recebe como retorno a informação se deve continuar ou não e qual o intervalo de tempo que precisará ficar repetindo esta operação no produtor. Se receber uma informação para terminar este terminará ou ficará bloqueado, dependendo da política de licenciamento. Caso receba a informação para continuar este criará uma *thread* para ficar repetindo esta

operação periodicamente no produtor e continuará o seu funcionamento normalmente. Esta é a *thread RunCheck_use*, mostrada na Figura 5.2.

5) Quando o uso do cliente for encerrado, este chamará o método *end_use* no produtor.

- Inicialize o ORB
- Descobrir Referência para o Objeto que implementa o Serviço de Nomes
- Descobrir Referência para o Objeto que implementa o LSM
- Obter numero IP desta máquina (Este número será enviado ao LSM)

Enviar ⇒ LSM

Operação: obtain_producer_specific_licence_service
Parâmetros: Enviar:uma cadeia de caracteres com o seguinte formato (nome do cliente/nome da máquina/número ip da máquina).
Receber: Referência para um Produtor.

Enviar ⇒ Produtor

Operação: start_use
Parâmetros: Enviar: Nome do Cliente e Versão
Receber: *token* para licença

Enviar ⇒ Produtor

Operação: check_use
Parâmetros: enviar: *token* para licença
Receber: continua ou não e intervalo para repetir periodicamente esta operação

Caso receba informação para interromper então:

- Abortar o funcionamento

Caso continua iniciar *Thread* RunCheckUse

Envia ⇒ RunCheckUse
Parâmetros: envia: (*token,* intervalo para repetição)

Agora o cliente de posse da licença pode executar

Quando o cliente for encerrar sua execução

Enviar ⇒ Produtor

Operação: end_use
Parâmetros: Enviar:*token*

Finalizar *Thread* RunCheckUse

Fim

Figura 5.5 – Algoritmo do Cliente

5.4 Considerações sobre a Implementação no ambiente CORBA

O aprendizado da construção de uma aplicação CORBA apresenta uma certa dificuldade inicial. Podemos observar que mesmo uma aplicação simples, um programa que passa apenas uma *String* por exemplo, entre um cliente e um servidor, possui dezenas de linhas de código.

Uma vez superada esta dificuldade inicial, o programador passa a usufruir das vantagens do ambiente. A transparência da rede na comunicação entre objetos torna a programação bastante eficiente. Outro ponto importante é a comunicação dos objetos criados com outros rodando em outros sistemas operacionais e escritos em outras linguagens de programação.

5.5 Sumário

Neste capítulo descrevemos os aspectos relativos à implementação de um Serviço de Licenciamento no ambiente CORBA. Vimos as características do ambiente de programação e descrevemos o funcionamento dos módulos do sistema.

Capítulo 6 – Considerações Finais

Como o crescimento do uso da indústria da informática e da utilização da Internet e das redes de computadores em geral, torna-se cada vez mais necessária a proteção da propriedade intelectual dos produtores de softwares. Estes produtores necessitam de formas de controle de acesso e uso dos seus produtos. Na maioria dos casos, isto é necessário para assegurar uma compensação justa pelo uso destes produtos. O método mais comum de se realizar este controle é através do licenciamento, onde a licença é fornecida através de um meio técnico (baseado em hardware ou software) ou através de formas contratuais. Embora a os meios contratuais sejam uma opção viável, eles não asseguram o mesmo nível de controle que os meios técnicos, que usam ferramentas de hardware ou de software para controlar o licenciamento. Devido a isto, os fornecedores de software continuam necessitando de métodos de licenciamento técnicos para complementarem os seus contratos legais.

O avanço da Internet e das redes de computadores facilitam a distribuição de softwares, por outro lado, facilitam a difusão da pirataria. Os serviços de licenciamento possibilitam a distribuição destes produtos de maneira fácil e prática, e continuam resguardando a propriedade intelectual do autor do software.

Com o aumento da banda passante das redes, temos também o crescimento de um novo mercado, o mercado dos componentes. Os consumidores estão relutantes em pagar por um software de prateleira no qual eles só utilizarão uma pequena percentagem. Para estes consumidores seria interessantes que fosse disponibilizado os componentes de forma *on-line* para o uso imediato. Esses componentes seriam utilizados ou baixados na rede. Neste caso seria necessário uma forma eficiente e confiável para medição da utilização destes componentes e da forma como seria feita a cobrança por essa utilização. O serviço de licenciamento é um elemento fundamental neste processo.

Para o desenvolvimento destes componentes, é necessário que seja utilizado algum padrão para o desenvolvimento das aplicações distribuídas. CORBA é um padrão aberto e proposto por um consórcio formado por mais de 700 empresas da área da informática, incluindo praticamente todas as grandes empresas do setor. Seus principais concorrentes são soluções proprietárias (como o DCOM) da Microsoft [Micro99], que atrelam o desenvolvedor às empresas produtoras destas soluções. CORBA especifica as interfaces do Serviço de Licenciamento como um dos seus Serviços de Objetos.

É necessário que este Serviço de Licenciamento seja robusto o suficiente para se recuperar de falhas no sistema. Estas falhas podem trazer grandes prejuízos aos produtores de software, na medida em que podem levar a permissão do uso indiscriminado de um produto. Para os consumidores, essas falhas podem trazer sérios transtornos, pois o sistema pode negar o uso de produto pelo qual este consumidor pagou. Outra conseqüência destas falhas pode ser a interrupção do serviço. Este trabalho de dissertação procurou lidar com as questões mostradas acima, no projeto de um Serviço de Licenciamento CORBA tolerante a falhas.

6.2 Contribuições deste trabalho

No projeto do Serviço de Licenciamento CORBA formam introduzidas maneiras simples e efetivas para lidar com falhas nos componentes e na rede de comunicação. As técnicas aqui propostas foram implementadas em um protótipo como forma de validação destas idéias.

Essas idéias mostram-se factíveis na prática, melhorando assim a confiabilidade do sistema como um todo.

Além das contribuições citadas acima, o nosso trabalho teve como contribuição o estudo e o desenvolvimento das tecnologias CORBA no DSC (Departamento de Sistemas e Computação) da UFPB. O fato de sermos os pioneiros na implementação de aplicações CORBA trouxe muitas dificuldades e muitas realizações.

As dificuldades foram a conseqüência do pioneirismo, principalmente pela falta de uma comunidade local para o intercâmbio de idéias. Como vimos no capítulo 2

(com o desenvolvimento de uma aplicação CORBA simples), o desenvolvimento em CORBA, apresenta uma certa dificuldade inicial, trazendo uma certa satisfação com o domínio da tecnologia e das facilidades trazidas para a programação em rede. CORBA apresenta-se como uma tecnologia bastante promissora para a implementação de aplicações distribuídas e a integração destas com os sistemas legados.

Outra contribuição foi a continuação da pesquisa sobre mecanismos de licenciamento. Área de pesquisa que começou neste departamento com o trabalho de [Bezerra96]. A continuação de uma pesquisa em uma determinada área leva a um maior enriquecimento do conhecimento e nos permite tratar com maior profundidade os assuntos abordados

6.3 Trabalhos Futuros

A implementação realizada é um protótipo para um Serviço de Licenciamento. A finalidade de seu desenvolvimento foi o de validar as técnicas de tolerância a faltas introduzidas no serviço . A implementação realizada não utilizou o arquivo que define as políticas de licenciamento. Em uma implementação para produção é importante a utilização deste arquivo. No protótipo (utilizado para os testes), a política de licenciamento era embutida no código fonte.

Devido a esta razão, o primeiro trabalho futuro seria a inclusão do arquivo de definição das políticas de licenciamento. Isto tornaria o serviço mais flexível.

Entre os trabalhos futuros encontram-se:

- Inclusão de novas políticas de licenciamento no sistema.

  Após a inclusão do arquivo de definição das políticas de licenciamento, seria incluída e testada novas políticas de licenciamento. Entre as políticas de licenciamento que podem ser facilmente implantadas no sistema encontram-se: quantidade de ativações acumuladas, data de expiração e licenciamento de sítio e "*gas metering*" (medição da utilização do software para controle ou cobrança de utilização). Já que o

aluguel de software apresenta-se, no presente momento, como uma tendência da computação nos próximos anos, uma atenção especial deve ser dada as políticas de licenciamento que controlam a utilização do software (como o "*gas metering*" por exemplo).

- Elaboração de uma interface gráfica para a configuração destas políticas de licenciamento.

  Com a realização da etapa anterior poderá ser criada uma interface gráfica para a configuração destas política aumentando assim a facilidade de uso do sistema.

- Inclusão de outros mecanismos de tolerância a falhas para aumento de confiabilidade. Como exemplo: a replicação dos Serviços de Objetos utilizados pelo Serviço de Licenciamento (nomes, eventos e propriedades) e do módulo LSM.

  Seria interessante o teste do sistema com a utilização de outros Serviços de Objetos CORBA replicados. Um teste interessante seria por exemplo a utilização do CosNamingFT proposto por [Lung99].

- Análise rigorosa do desempenho do sistema, para a medição da degradação do desempenho do sistema com a inclusão destes mecanismos de tolerância a falhas.

  Assim poderíamos avaliar com precisão a degradação do desempenho do sistema com a inclusão das técnicas propostas neste trabalho. Deve se levar em consideração que a degradação no desempenho do sistema pode causar comportamento errôneos no sistema, como por exemplo, a negação de fornecimento de licenças já disponíveis.

Referências Bibliográficas

[Baker97] Baker,Seán; *Corba Distributed Objects using Orbix, ACM Press 1997*

[Bezerra96] Bezerra, Tárcio ; *Bouncer – Uma Solução Distribuída para o controle de Licenças de Software.*Dissertação de Mestrado, COPIN/DSC/UFPB,Dezembro/96.

[Bezerra97] Bezerra, Tárcio; Brasileiro, Francisco;Cirne, Walfredo, *Bouncer – Um Serviço Distribuído e Tolerante a Faltas para Controle de Licenças de Software*, 1997.

[Black99] Java-Linux Home Page, http://www.blackdown.org

[Booch97] Booch, Grady et al; *The Unified Modeling Language User Guide, Addison Wesley Logman 1997*

[Booch99] Booch, Grady et al; *The Unified Modeling Language Reference Manual, Addison Wesley Logman 1999*

[BSA99] Business Software Alliance, *1998 Global Software Piracy Report, Maio 1999*

[Conectiva99] Conectiva Linux - O Linux em Português, http://www.conectiva.com.br

[Farley98] Farley, Jim; *Java Distributed Computing*, O`Reilly&Associates,Inc., Janeiro de 1998.

[Fowler97] Fowler, Martin; Scott, Kendall, *UML Distilled, Addison Wesley Longman 1997*

[Iona99] Iona Technologies PLC, *Orbix Web Programmer's Guide, Junho 1999*

[Kotopoulis98] Kotopoulis Alexander & Julia Miller; *Licensing Metering: Usage Control provides meaningful benefits for eletronic commerce*, Distributed Computing Magazine, May 1998.

[Lung99] Lung, Lau Cheuk; *CosNamingFT – Um Serviço de Nomes Tolerante a Falhas em Conformidade com o Padrão OMG*, SBRC 99 (Simpósio Brasileiro de Redes de Computadores)

[Micro99] COM: Delivering on the Promises of Component Technology, http://www.microsoft.com/com

[OMG96] Object Management Group, *CORBAservices: Common Object Request Services Specification*, MarÓo 1997

[OMA97] Object Management Group, *A Discussion of the Object Management Architecture, Janeiro 1997*

[OMG98] Object Management Group; *The Common Object Request Broker: Architecture and Specification, version 2.2, OMG Document 98-07-03, Fevereiro 1998*

[OMG99] Object Management Group Home Page, http://www.omg.org

[OMGserv99] CORBAservices Index, http://www.omg.org/library/csindx.html

[OOC99a] OOC home page , http://ooc.com

[OOC99b] *OOC – Object Oriented Concepts,* ORBacus for C++ and Java Documentation – Version 3.1.2

[SEGUE99] SegueSoftware inc, http://www.segue.com/html/s_solutions/silk/s_silkmeter .htm

[Sun99] The Source for Java Technology, http://www.java.sun.com

[Vinoski97] Vinoski, Steve; *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*, IEEE Communications Magazine, Vol. 35, No. 2, Fevereiro 1997.

[Voguel98] Voguel, Andreas; Duddy, Keith; *Java Programming with CORBA*, John Wiley&Sons, Inc., 1998

Apêndice A: IDL do Serviço de Licenciamento

Interface do Serviço de Licenciamento em IDL (Interface Definition Language) padronizada pela OMG (Object Management Group).

```
#ifndef _COS_LICENSING_MANAGER_IDL_
#define _COS_LICENSING_MANAGER_IDL_

#include <CosEventComm.idl>
#include <CosPropertyService.idl>
#include <CosEventComm.idl>

#pragma prefix "omg.org"

module CosLicensingManager {
      exception InvalidProducer{};
      exception InvalidParameter{};
      exception ComponentNotRegistered{};

      typedef Object ProducerSpecificNotification;

      enum ActionRequired { continue, terminate};

      enum Answer { yes, no };

      struct Action {
            ActionRequired action ;
            Answer notification_required ;
            Answer wait_for_user_confirmation_after_notification ;
            unsigned long notification_duration;
            ProducerSpecificNotification producer_notification;
            string notification_text;
      };

      struct ChallengeData {
            unsigned long challenge_index;
            unsigned long random_number;
            string digest;
      };


      enum ChallengeProtocol { default_protocol, producer_defined
};
      struct Challenge {
            ChallengeProtocol challenge_protocol;
            unsigned long challenge_data_size;
            any challenge_data;
      };
```

```
            typedef any LicenseHandle;

            interface ProducerSpecificLicenseService {

            readonly attribute   string producer_contact_info;
            readonly attribute   string
producer_specific_license_service_info;

            LicenseHandle start_use (
                  in CORBA::Principal aPrincipal,
                  in string component_name,
                  in string component_version,
                  in CosPropertyService::PropertySet license_use_context,
                  in CosEventComm::PushConsumer call_back,
                  inout CosLicensingManager::Challenge Challenge)

                  raises ( InvalidParameter, ComponentNotRegistered);

                  void check_use (
                        in LicenseHandle handle,
                        in CosPropertyService::PropertySet
license_use_context,
                        out unsigned long recommended_check_interval,
                        out Action action_to_be_taken,
                        inout CosLicensingManager::Challenge Challenge)

                        raises ( InvalidParameter );

                  void end_use (
                        in LicenseHandle handle,
                        in CosPropertyService::PropertySet
license_use_context,
                        inout CosLicensingManager::Challenge Challenge)

                        raises ( InvalidParameter );
            };

            interface LicenseServiceManager {
                  ProducerSpecificLicenseService
                  obtain_producer_specific_license_service (
                        in string producer_name,
                        inout CosLicensingManager::Challenge Challenge)

                        raises ( InvalidProducer, InvalidParameter ) ;
            };
      };

      #endif /* ifndef _COS_LICENSING_MANAGER_IDL_ */
```

Apêndice B: Interface Adicionada ao Serviço de Licenciamento

Interface Adicionada ao Serviço de Licenciamento:

```
// Este arquivo é uma extensão do CosLicencingModule.idl
// CosLicensingManager2.idl


#include "CosLicensingModule.idl"
#ifndef COS_LICENSING_MODULE_IDL
#define COS_LICENSING_MODULE_IDL



module CosLicensingManager {

     exception Could_not_create_producer {};


     interface Factory
     {

          ProducerSpecificLicenseService createProducer (
                                        in string nome_do_produtor,
                                        in string status
                                        )
                              raises (Could_not_create_producer);

     };




};

#endif
```