Um Serviço de Tempo Global para Sistemas Distribuídos de Larga Escala

Joni Fraga Jean-Marie Farines Carlos Montez

Laboratório de Controle e Microinformática - LCMI
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
Caixa Postal 476 - 88040-900 - Florianópolis - SC
e-mail: {fraga, farines, montez} @lcmi.ufsc.br

Abstract

The application developments in distributed systems are made easier when a time service offering a global timebase is available. The objective of this paper is to describe a time service for large-scale distributed systems. The CORBA time service specification is adopted for building the interfaces of the proposed service. In this paper, we discussed some programming models to the service implementation in a wide area network that involves the coexistence of different communication supports. The service proposed allows the integration of different clock synchronization algorithms which are adapted to the diversity of communication supports, available in the network. In these synchronization algorithms, GPS receivers are used as source of external reference. Some examples are presented for illustrating the use of our time service.

Resumo

Serviços de tempo oferecendo referências de tempo global são reconhecidamente uma forma de simplificar o desenvolvimento de aplicações em sistemas distribuídos. O objetivo desse artigo é discutir uma proposta de serviço de tempo global para sistemas distribuídos de larga escala. As especificações do serviço de tempo do CORBA, na forma de suas abstrações e operações, são adotadas na definição das interfaces do serviço proposto. Nesse artigo são discutidos modelos para a implementação desse serviço em uma rede metropolitana envolvendo a convivência de diferentes suportes de comunicação. O serviço proposto permite a integração de diferentes algoritmos de sincronização de relógios apropriados aos diferentes suportes de comunicação existentes na rede. Os algoritmos de sincronização usados se utilizam de receptores GPS como fonte de referência externa. Com o objetivo de ilustrar a utilização do serviço de tempo, são apresentados alguns exemplos.

1. Introdução

Um serviço oferecendo uma base de tempo global é uma necessidade premente se considerarmos os sistemas distribuídos atuais. O aumento das potencialidades computacionais disponíveis e a comunicação através de redes com velocidades cada vez mais elevadas, permitem tratar aplicações com grau de complexidade crescente e exigências maiores em termos de requisitos temporais. O uso de uma referência única de tempo (tempo global) simplifica as soluções de problemas de sistemas distribuídos envolvendo necessidades de acordo, ordenação de eventos, implementações de restrições temporais, etc.

Os sistemas computacionais de larga escala se configuram hoje, em sistemas com grande distribuição espacial, com um caráter aberto, integrando quantidades significativas de recursos computacionais caracterizados pela diversidade de seus *hardwares* e *softwares*. São recentes e significativos os esforços em suportes de *middleware* que, além de lidar com a distribuição, fornecem a portabilidade de aplicações e a interoperabilidade de serviços diante da heterogeneidade dos recursos. Como exemplos bem sucedidos desses esforços, criando a idéia de sistemas abertos, estão o ambiente ANSAware [Herbert 94], o DCE/OSF [Johnson 94] e os suportes seguindo a padronização CORBA/OMG [OMG 95].

Algumas dessas proposições de *middleware* apresentam especificações ou mesmo implementações de serviços de tempo, oferecendo um conjunto de abstrações e operações envolvendo o tempo, necessárias às aplicações. Esses conceitos e operações são implementados usualmente através de acessos a relógios locais do sistema distribuído. A forma a qual esses relógios locais são mantidos sincronizados, usualmente não é tratada de forma satisfatória

nessas especificações e implementações de *middleware*. Recentemente, com a crescente disponibilidade de receptores de GPS [Dana 97], surgiram diversas novas soluções referentes à sincronização de relógios dirigidas para redes de larga escala [Veríssimo 97], [Fetzer 97] e [Schmid 97].

O objetivo desse artigo é apresentar uma proposta de serviço de tempo global para sistemas distribuídos de larga escala. As especificações do serviço de tempo CORBA (*Time Service Specification* [OMG 97]) são adotadas no serviço proposto. Como conseqüência, temos interfaces públicas e bem definidas nesse serviço, ocultando das aplicações as especificidades de cada suporte de comunicação, e a abordagem adotada para manter os relógios locais sincronizados.

O serviço de tempo proposto faz parte do projeto de uma rede metropolitana experimental em estágio de desenvolvimento. Por se tratar de um sistema de larga escala, essa rede deve envolver a integração de diferentes suportes de comunicação (síncronos, assíncronos temporizados, etc). As soluções adotadas no serviço proposto permitem a convivência integrada de diversas abordagens na sincronização de relógios locais, compatíveis com as características nos diferentes suportes de comunicação existentes em uma rede de larga escala. Em qualquer ponto do sistema, as operações e abstrações disponíveis através do serviço de tempo são as mesmas, independente das características dos serviços subjacentes de comunicação ou de sincronização dos relógios locais.

Faz parte também da apresentação do serviço proposto, mostrar o entendimento dos conceitos envolvidos nas especificações CORBA de serviço de tempo. Para ilustrar a importância de uma fonte de tempo global em um sistema distribuído e as diversas potencialidades no uso do serviço de tempo, apresentamos exemplos tirados de um experimento em desenvolvimento que integra um conjunto de aplicações previstas inicialmente no projeto da rede metropolitana considerada.

O presente texto se encontra dividido como se segue. Na seção 2, são descritos sucintamente os serviços de tempo do DCE e do CORBA. Na seção 3, o problema de sincronização de relógios é discutido, destacando em particular as soluções de sincronização baseadas no uso de GPS. Uma proposta de serviço de tempo e sua utilização numa aplicação cooperativa são apresentadas respectivamente nas seções 4 e 5.

2. Serviços de tempo em sistemas abertos

Muitas das soluções disponíveis de *middleware* para sistemas distribuídos abertos apresentam especificações ou mesmo implementações de serviços de tempo. Esses serviços de tempo oferecem um repertório de operações que permitem desde datar e ordenar eventos em um sistema distribuído até operações mais complexas, envolvendo o controle de suas ativações a partir do tempo. Nesse item descrevemos os serviços de tempo de duas propostas de ambientes para programação aberta: o DCE/OSF e o CORBA/OMG.

2.1. Serviço de tempo do DCE

O DCE (*Distributed Computing Environment*) é um ambiente de computação distribuído resultado de um trabalho coordenado pelo OSF, com o objetivo de propor soluções para sistemas abertos [Johnson 94]. O serviço de tempo do DCE (DTS - *Distributed Time Service*) é disponível através de interações cliente/servidor, utilizando o suporte de RPC do DCE. A implementação desse serviço considera a existência de *servidores de tempo* e de *clerks*. O servidor de tempo é responsável pelo fornecimento de valores de tempo, na forma de intervalos, a partir de uma fonte externa. Por sua vez, cada nó possui um processo responsável pelo relógio local, denominado de *clerk* que atende os pedidos referentes a tempo de seus clientes locais.

Os relógios locais são sincronizados a partir de pedidos submetidos pelos *clerks* aos servidores de tempo. Cada servidor de tempo (um ou mais por célula ou domínio de nomes no sistema)

expressa o tempo na forma de intervalos. Os processos *clerks* obtêm intervalos de vários servidores de tempo e calculam a interseção desses intervalos durante a sincronização de seus relógios locais. Intervalos de tempo que não possuem interseção são considerados faltosos e portanto descartados. Os relógios locais são então ajustados levando em conta os pontos médios das interseções calculadas nas interações *clerks*/servidores de tempo. A Figura 1.a esboça o cálculo do ajuste efetuado por um *clerk*. O paradigma utilizado no DCE que expressa o tempo na forma de intervalos está fundamentado em [Marzullo 84].

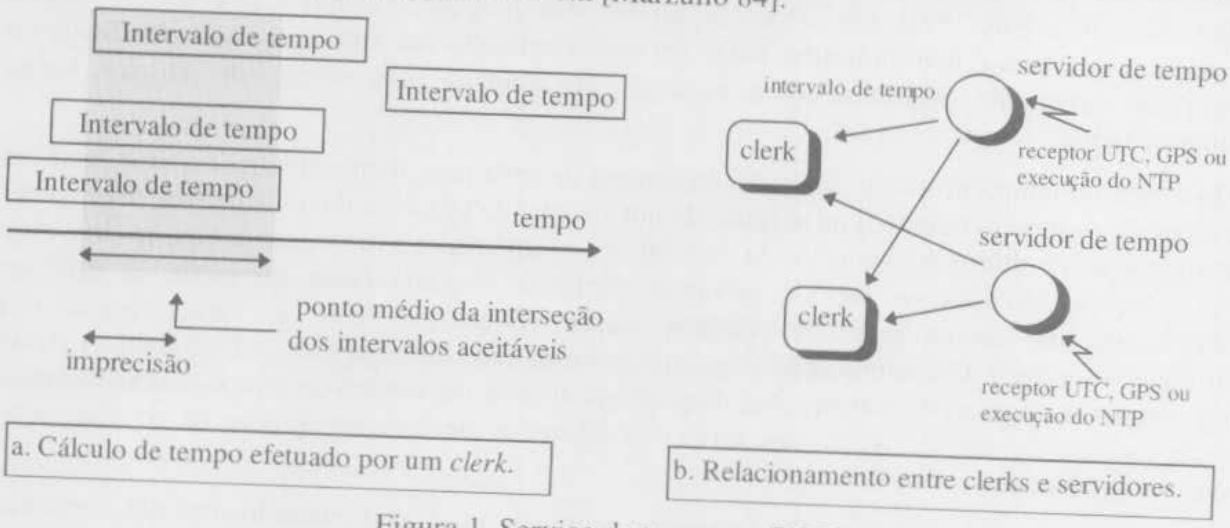

Figura 1. Serviço de tempo no DCE.

Os *servidores de tempo* do DCE podem encapsular receptores de rádio UTC ou GPS. Ou ainda podem se utilizar da estrutura hierárquica do NTP [Mills 91] para se manterem sincronizados externamente. A Figura 1.b ilustra as relações *clerks/servidores de tempo*.

O DTS/DCE oferece mais de 30 operações relacionadas com o tempo. Esses serviços permitem às tarefas obter o valor de tempo atual, comparar valores de tempo, executar operações aritméticas sobre os valores de tempo, etc. É importante observar que devido às imprecisões envolvidas, nem sempre é possível estabelecer uma ordem entre dois eventos. Quando ocorre uma interseção (envelope de erro) entre dois intervalos de tempo associados a dois eventos, as imprecisões (intervalos de tempo) dos valores de tempo dos dois eventos impedem que se estabeleça uma ordem entre os mesmos.

2.2. Serviço de tempo CORBA

As especificações CORBA [OMG 95] formam propostas de padrões para suportes de *middleware* que permitam a programação aberta. O serviço de tempo, adotado recentemente nas especificações CORBA pela OMG [OMG 97], é visivelmente inspirado no DTS/DCE, fornecendo meios aos usuários para obtenção do tempo atual juntamente com uma estimativa de imprecisão associada.

Adicionalmente, a interface do serviço de tempo oferece também facilidades para lidar com intervalos de tempo e a determinação de ordens entre eventos. Outras facilidades opcionais oferecidas, são a geração de eventos programada por temporizadores. Da mesma forma que ocorre em grande parte dos serviços especificados pela arquitetura OMA, é também possível se estender o serviço de tempo CORBA através da herança. Com isso, aplicações poderiam especializar esses serviços segundo necessidades no tratamento de suas restrições temporais.

As especificações CORBA do serviço de tempo não determinam a forma como os relógios devam ser mantidos sincronizados para suportar a noção de tempo global no sistema distribuído, apenas sugerem soluções possíveis implementadas a partir de serviços subjacentes. Dessa forma, o serviço de tempo com as imprecisões associadas é factível de ser implementado usando, por exemplo, um algoritmo de sincronização probabilista de relógios como o de [Cristian 89], ou o estatístico NTP [Mills 91], ou ainda através do DTS/DCE [Johnson 94]. Nas

especificações, as únicas suposições feitas sobre o serviço subjacente são: (i) o mesmo deve ser capaz de retornar o tempo atual juntamente com um fator de erro associado; (ii) deve ser disponível e confiável; e (iii) o tempo fornecido deve ser monotonicamente crescente. Algumas soluções possíveis na sincronização de relógios em sistemas de larga escala, serão examinadas na seção 3.

2.2.1. Estrutura do serviço de tempo CORBA

No CORBA, *o serviço de tempo básico* (*Basic Time Service* [OMG 97]) é definido como possuindo uma interface de serviço de tempo que permite criar e gerenciar dois tipos de objetos (Figura 2): *Universal Time Object* (UTO) e *Time Interval Object* (TIO). Um objeto UTO traz na sua representação uma dupla (*UTO.time, UTO.inaccuracy*) que expressa um valor de tempo (*UTO.time*) com a imprecisão associada (*UTO.inaccuracy*). Os objetos TIO são introduzidos para representar instâncias de intervalos de tempo, na forma [limite_inferior, limite_superior].

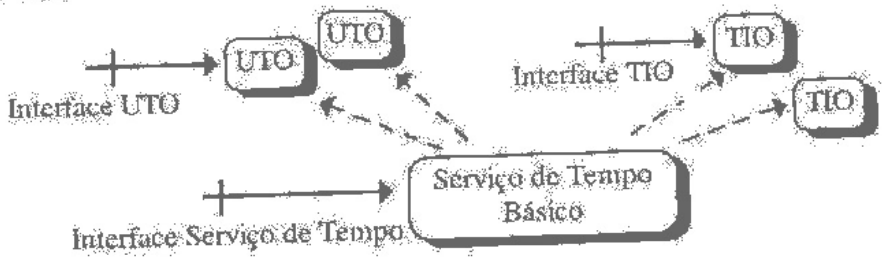

Figura 2. O modelo objeto para o serviço de tempo básico.

Dois objetos UTO podem ser comparados no sentido de estabelecer ordens. Existem dois tipos de comparação: (i) considerando apenas os atributos *UTO.time*; e (ii) considerando também as imprecisões (*UTO.inaccuracy*) dos objetos UTO envolvidos. A comparação (i) retorna um valor indicando se o atributo *UTO.time* do primeiro objeto é igual (*TCEqualTo*), maior (*TCGreaterThan*) ou menor (*TCLessThan*) que o do outro. Enquanto a comparação (ii) pode retornar, além das constantes indicadas acima, também um valor "indeterminado" (*TCIndeterminate*) no caso de ocorrência de um *envelope de erro* devido a sobreposição das imprecisões consideradas. Um objeto TIO pode ser comparado com outro objeto TIO. As comparações de intervalos em objetos TIO retornam valores de tipo enumerado que representam a forma de sobreposição dos intervalos comparados (Figura 3).

Intervalo A	⟷	⟷	⟷	⟷
Intervalo B	⟷	⟷	⟷	⟷
Enumerado	OTContainer	OTContained	OTOverlap	OTNoOverlap
Significado	Contém	Contido	Entrelaçado	Não entrelaçado

Figura 3. Tipo de sobreposição de intervalos.

As definições de tipos de estruturas de dados e constantes utilizadas nas definições das interfaces do serviço de tempo são definidas em um módulo denominado *TimeBase*, enquanto as interfaces são definidas no módulo *CosTime*.

2.2.2. Operações no serviço de tempo CORBA

Interface Serviço de Tempo

A principal operação disponível na interface Serviço de Tempo é a *universal_time()* que retorna um objeto UTO representando o tempo atual com uma estimativa de imprecisão associada. Essa interface possui também operações que permitem criar objetos UTO (*new_universal_time()*) e objetos TIO (*new_interval()*) a partir de valores de tempo especificados em parâmetros pelo cliente. Diferente da primeira operação que tem como fonte de tempo o relógio local, essas

últimas operações são as únicas formas de se criar objetos UTO e TIO com valores arbitrários supridos pelo cliente.

A interface de serviço de tempo possui também operações de conversão. É o caso da operação *uto_from_utc()* que converte uma estrutura do tipo *utc* (compatível com o formato do DTS) em um objeto UTO. Essa operação é usada para converter valores UTC recebidos pela rede na representação de um objeto UTO.

Interface UTO

Cada objeto UTO possui como atributos, um tempo e uma imprecisão associada. Esses atributos são *apenas de leitura*, não havendo na interface UTO métodos que alterem esses valores. Entretanto, as operações existentes nessa interface permitem a manipulação desses atributos. Por exemplo, a operação *interval()* retorna um objeto TIO correspondente aos atributos do UTO. A operação *compare_time()* permite a comparação de dois objetos UTO, levando em consideração o tempo no próprio objeto com o tempo em outro objeto passado como parâmetro.

Os valores de tempo são representados internamente nos objetos UTO em atributos de 64 *bits*[1], com a unidade de tempo equivalendo a 100ns. Quando o objeto UTO representa o tempo atual, considera-se que o valor armazenado é o tempo transcorrido desde o tempo base de 15 de outubro de 1582 00:00:00 até o instante atual. Entretanto, um objeto UTO pode representar qualquer valor de tempo, pois a interpretação dada a esse valor é dependente da semântica da própria aplicação. A interface UTO possui uma operação, *absolute_time()*, que interpreta o valor armazenado em um objeto UTO, como um tempo relativo (um deslocamento). O resultado dessa operação retorna um outro objeto UTO correspondendo ao valor dado pela soma: *tempo corrente + tempo relativo*. Esse novo objeto representa um valor de tempo futuro.

Interface TIO

A interface TIO possui também estruturas *apenas de leitura*. A operação *time()* existente nessa interface retorna um objeto UTO com a imprecisão (*UTO.inaccuracy*) igual à metade do intervalo de tempo no TIO e o valor de tempo (*UTO.time*) dado pelo ponto médio desse intervalo. Uma outra operação, *overlap()*, permite comparar dois objetos TIO, retornando um dos valores de tipo enumerado indicados anteriormente na Figura 3.

3. Sincronização de relógios

A necessidade de manter próximos os valores dos relógios locais, determina a necessidade de algoritmos de *sincronização interna* de relógios[2] que periodicamente corrigem esses relógios, mantendo-os próximos entre si. A sincronização interna envolve trocas de mensagens entre os nós possuidores desses relógios locais.

Além disso, em algumas aplicações é necessário que os valores dos relógios se mantenham próximos de valores originados da fonte de tempo real (fonte externa) que pode ser, por exemplo o padrão UTC (*Coordinated Universal Time*) ou o TAI (*Temps Atomique International*), ambos disseminados via satélites ou transmissões de rádio. Alguns nós, especialmente preparados no sistema distribuído, podem captar esses valores de tempo padrão, e atuarem no sentido de manter o conjunto de nós da rede com seus relógios próximos em relação a essa fonte externa. Esse problema é denominado *sincronização externa*. É importante observar que ao se fazer a sincronização externa de relógios, automaticamente se faz a sincronização interna; apesar do inverso não ser verdadeiro [Kopetz 89].

[1] A representação do tempo no CORBA é a mesma do DTS/DCE: uma estrutura de 64 *bits* capaz de armazenar valores de tempo UTC.

[2] Nesse trabalho, uma vez que o estudo é dirigido para sistemas de larga escala, estamos limitados a soluções por *software* na sincronização de relógios que são mais adequadas a esses sistemas.

Os valores de precisão interna (*precision*) e externa (*accuracy*)[3] são indicadores da qualidade dos algoritmos usados em ambas sincronizações. Os erros de medidas, falhas de nós e os atrasos no suporte de comunicação, são fatores determinantes na qualidade dessas precisões.

3.1. Abordagens na sincronização de relógios

Os algoritmos de sincronização de relógio, podem ser classificados em deterministas, probabilistas e estatísticos.

Sincronização determinista de relógios

Diversos algoritmos clássicos de sincronização de relógios, assumem a existência de uma delimitação nos atrasos máximos na transmissão de mensagens, isto é, assumem suportes de comunicação síncronos. Esses algoritmos são deterministas e funcionam basicamente com todos os nós envolvidos em trocas de mensagens. Podem envolver um mestre central; ou evoluírem totalmente distribuídos, constituídos ou de um ciclo (*round*) de trocas de mensagens seguido de cálculos de ajuste através de uma função de convergência; ou ainda distribuídos e baseados em protocolos de acordo [Kopetz 89].

Em sistemas de larga escala, como por exemplo os sistemas baseados na *Internet* (chamados em [Stankovic 96] de *sistemas globalmente distribuídos*), devido às dimensões, ao número de nós e às características de carga, soluções envolvendo muitas trocas de mensagem e exigindo delimitação nos atrasos, ou se tornam impraticáveis ou possuem um desempenho sofrível.

Sincronização probabilista de relógios

A abordagem probabilista assegura apenas que os relógios serão sincronizados com uma certa probabilidade. Algoritmos que seguem essa abordagem são adequados a sistemas assíncronos temporizados (*timed asynchronous distributed systems* [Cristian 96]), caracterizados pela ausência de uma delimitação máxima nos atrasos de mensagens no suporte de comunicação. A abordagem é baseada na existência de uma curva de distribuição de atrasos (Figura 4).

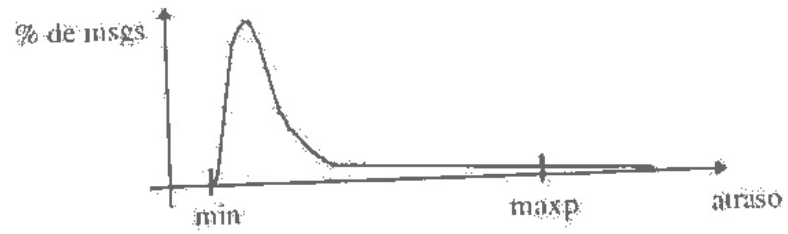

Figura 4. Distribuição de atrasos característica em suportes assíncronos temporizados.

Em síntese, os probabilistas são projetados para redes de comunicação onde seja possível obter atrasos mínimos (*min*, na Figura 4) e curvas de distribuição de atrasos. Na prática, a distribuição das mensagens nessas redes apresenta a sua situação de pico próxima do valor de atraso mínimo.

A sincronização de relógios nessas redes é conseguida, assumindo um valor de *timeout* (*maxp*, Figura 4) como forma de detectar falhas de comunicação e delimitar o tempo máximo de espera de mensagens. Os valores de *maxp* são escolhidos a partir de um estudo sobre parâmetros e características físicas da rede em tempo de projeto. Devido a descartes de leituras de relógios remotos cujas mensagens ultrapassem *maxp*, o algoritmo normalmente não garante uma

[3] Nós de um sistema distribuído possuem relógios locais que apresentam diferentes desvios entre si e em relação a uma fonte externa (fornecedora do *tempo real*). As taxas de desvio são dependentes das características físicas dos relógios. Os algoritmos de sincronização devem manter limitados os desvios máximos. Nesse sentido, a diferença máxima permitida pela sincronização interna entre valores de diferentes relógios locais é denominada de *precisão interna* (*precision*). Na sincronização externa, a diferença máxima entre o tempo obtido em relógio local de um nó e o tempo externo correspondente (tempo real) é denominada de *precisão externa* (*accuracy*).

sincronização determinista. Existe uma negociação entre a precisão interna desejada e a probabilidade de sincronização. Quanto melhor a precisão escolhida (valores de *maxp* próximos de *min*), menor a probabilidade de sucesso na sincronização. O inverso também é verdadeiro: quanto pior a precisão desejada, maior a probabilidade de sincronização.

Em [Cristian 89] é apresentado um algoritmo probabilista de sincronização externa de relógios. O algoritmo supõe em cada nó uma tarefa *servidor de tempo*, responsável pelo fornecimento e correção de um serviço de tempo a partir do relógio local. A idéia básica é permitir que cada servidor de tempo obtenha a informação do relógio de um servidor remoto com acesso a referência externa de tempo (*servidor de referência*), através de interações mestre-escravo. Esse método é estendido também para a sincronização interna de relógios em redes assíncronas temporizadas em [Cristian 95]. Nessa extensão, um servidor de tempo faz leituras probabilistas dos valores dos relógios remotos e usa os mesmos em funções de convergências para o cálculo de ajustes.

A necessidade do conhecimento em tempo de projeto da curva de distribuição de atrasos do suporte de comunicação, inviabiliza o uso de algoritmos probabilistas em sistemas baseados na *Internet*. Essa restrição é reconhecida em [Cristian 89] que propõe como possível extensão, um valor de *timeout* adaptando-se continuamente às condições dinâmicas da rede.

Sincronização estatística de relógios

Algoritmos de sincronização estatística[4] não impõem quaisquer restrições à rede de comunicação. São construídos tendo como base a determinação de seus parâmetros dinamicamente, ajustando-os à estatística de variações dos atrasos nos suportes de comunicação.

O algoritmo NTP (*Network Time Protocol* [Mills 91]) apresenta características de sincronização estatística de relógios. Relógios locais são sincronizados, usando diferentes valores de relógios remotos, disponíveis a partir de *servidores de tempo*. Estampilhas de tempo incluídas nas mensagens trocadas entre um nó e servidores de tempo (pedidos e respostas) são usadas para determinar as defasagens entre os relógios. O nó pode calcular (estimar) os atrasos das mensagens trocadas e o desvio de seu relógio em relação ao do servidor remoto.

A precisão do NTP não é delimitada como nas outras abordagens mas é função das condições dinâmicas da rede em um dado instante. A sincronização estatística é adequada para redes do tipo como a *Internet* que se caracterizam pela instabilidade de seus suportes de comunicação e pela carga dinâmica, não permitindo, por exemplo, um levantamento para essas redes de uma curva confiável da distribuição de seus atrasos ou ainda, o conhecimento prévio de atrasos máximos e mínimos. A princípio, o algoritmo estatístico NTP pode ser usado em qualquer tipo de rede, pois funciona com mecanismos que compensam variações estatísticas nos atrasos. Entretanto, o mesmo não é adequado para aplicações críticas de tempo real.

O NTP tem como características adicionais: a organização hierárquica dos servidores de tempo; a sincronização externa baseada em receptores do padrão UTC; tolerância a faltas e auto-reconfiguração automática; precisão interna (estatística) da ordem de 1 ms em redes locais e da ordem de dezenas de milisegundos para sistemas de larga escala.

3.2. Soluções para sistemas de larga escala

A grande maioria dos algoritmos de sincronização não são adequados aos sistemas de comunicação envolvidos em sistemas de larga escala. O NTP se apresenta hoje como a solução mais usada para sistemas de larga escala baseados na *Internet*, embora as limitações citadas acima. Esse quadro tende a se modificar com a crescente utilização do GPS. Devido ao baixo

[4] A sincronização NTP foi denominada estatística em [Cristian 95], para diferenciar de seu algoritmo de sincronização probabilista.

custo dos receptores de GPS, soluções mais adequadas começam a surgir. Muitas dessas soluções combinam propriedades das abordagens citadas anteriormente.

3.2.1. GPS - Global Positioning System

O NavStar GPS (*Global Positioning System*), ou simplesmente GPS, é um sistema baseado em 24 satélites na órbita terrestre, que emitem continuamente sinais de navegação, e fornecem uma cobertura de toda superfície terrestre [Dana 97]. Apesar de ter sido concebido originalmente como uma ferramenta de auxílio à navegação, o GPS permite a obtenção precisa de valores de tempo. O GPS foi colocado completamente operacional a partir de julho de 1995. Alguns receptores de sinais GPS operam na forma de placas que podem ser adicionadas ao barramento de computadores pessoais. Muitos receptores GPS distribuem sinais de tempo sobre interfaces RS-232 ou IEEE-488, em uma variedade de formatos.

O tempo propagado pelo GPS é uma medida contínua tendo como base o tempo UTC da 0 horas de 6 de janeiro de 1980. O tempo GPS como padrão TAI, não introduz o segundo de preenchimento (*leap second*), encontrando-se, portanto, alguns segundos adiante do UTC (11s em 1 de janeiro de 1996). Entretanto, as informações transmitidas pelos sinais dos satélites GPS contêm parâmetros que permitem o cálculo do valor correto do UTC.

No atual estado da tecnologia, a colocação da antena é crítica para aplicações de tempo. A antena deve ser externa, e estar disponível para a maior parte do espaço aberto, de horizonte a horizonte em todas as direções. Receptores GPS devem ser conectados às respectivas antenas através de cabos curtos para não introduzir atrasos. Além disso, interferências de rádio frequência locais podem afetar a recepção.

Em 1996, existiam mais de 50 companhias fornecendo mais de 275 modelos de receptores GPS, com soluções variando de preço desde dezenas de milhares de dólares até algumas centenas de dólares, e valores de precisão obtidos desde centenas de nanosegundos até centenas de microsegundos. A faixa de preço atual de soluções completas (antena, cabos e receptor) para microcomputadores/estações de trabalho se situa na média entre US$1000 a US$3000, para precisões obtidas da ordem de poucos microsegundos.

3.2.2. CesiumSpray

Em [Veríssimo 97] é apresentado o *CesiumSpray*, um esquema de sincronização de relógios para ser usado no oferecimento de um serviço de tempo em sistemas de larga escala, representados como uma WAN de LANS. As redes locais são assumidas com atrasos máximos delimitados e possuindo um nó especializado com um receptor GPS (ou mais de um, no sentido da tolerância a falhas).

O *CesiumSpray* é hierárquico, onde o conjunto de satélites que formam o NavStar GPS formam a raiz da hierarquia e espalham sua referências de tempo sobre um conjunto de nós que possuem receptores GPS. Os nós GPS formam o primeiro nível da hierarquia. O segundo nível é formado pelos nós de cada rede local do sistema. Esse algoritmo é um esquema híbrido de sincronização, onde somente poucos nós executam a sincronização externa (nós receptores de GPS) que, por sua vez, cooperam na sincronização interna no sentido de disseminar o tempo externo em todo o sistema.

O *CesiumSpray* utiliza um algoritmo de sincronização interna inspirado no algoritmo de *acordo a posteriori* [Veríssimo 92] para "espalhar" o tempo externo entre os relógios de cada rede local. Esse algoritmo se utiliza da possibilidade de difusão em redes locais, no sentido de reduzir o impacto das variações dos atrasos de mensagens sobre a precisão interna. As precisões interna e externa obtidas no algoritmo são excelentes e independentes do tamanho da WAN e de sua posição geográfica.

3.2.3. Algoritmo de Fetzer e Cristian

Fetzer e Cristian [Fetzer 97] propuseram também uma solução integrada de sincronização externa e interna de relógios locais para sistemas de larga escala. Embora se adapte ao modelo probabilista, esse algoritmo, por usar os receptores GPS como referência externa, pode também ser utilizado em redes do tipo WAN de LANs como o *CesiumSpray*. O esquema proposto se baseia em um conjunto de *servidores de tempo* disponíveis nos diferentes nós de uma rede local e de *servidores de referência* situados em nós com receptores GPS na mesma rede. O modelo de falhas adotado assume que os servidores podem sofrer falhas arbitrárias.

A cada ciclo de sincronização do algoritmo, o ajuste interno (na sincronização interna) é calculado a partir dos valores dos relógios de todos os servidores de tempo corretos, no ciclo considerado, usando uma função de convergência (*midpoint fault-tolerant function* [Schneider 86]). O ajuste externo leva em consideração o desvio entre o relógio local considerado e o valor médio obtido dos relógios dos servidores de referência. A integração das sincronizações externa e interna é feita pelo uso do ajuste externo na alteração do ajuste interno até no máximo um valor limite de ajuste (a constante *D* introduzida em [Fetzer 97]). O fato de ter um limite nos ajustes possibilita que os relógios sejam reajustados suavemente quando do retorno de um estado de somente sincronização interna (perda da referência externa no sistema) ao estado de sincronização externa/interna; este aspecto é salientado pelos autores como uma vantagem sobre o *CesiumSpray*.

4. Um serviço de tempo em sistemas abertos de larga escala

4.1. Sistema alvo

As aplicações distribuídas de larga escala vêm gerando um tráfego de rede com características diferentes para as diversas mídias (áudio, vídeo, dados) e têm demandado um aumento crescente de largura de banda passante. As redes baseadas no modo de transferência assíncrona (ATM) vêm sendo cada vez mais adotadas como solução para a comunicação, por apresentar vantagens técnicas que permitem atender a esses requisitos. Alta banda passante, incorporação da qualidade de serviço (QoS) fim-a-fim, suporte para múltiplas conexões virtuais e para classes de tráfego diferentes, flexibilidade da topologia, interoperabilidade em redes locais e de longa distância são as principais qualidades que permitem considerar a rede ATM como uma solução adequada e duradoura para aplicações distribuídas, em particular quando estas envolvem várias mídias.

Por estas razões, uma rede metropolitana de alta velocidade com quatro comutadores ATM interconectando nós ATM e redes locais em diversas instituições de ensino e pesquisa e empresas da cidade de Florianópolis está sendo instalada. Essa rede, inicialmente, servirá de campo de prova para testar soluções de suportes e de aplicações. Entre os experimentos previstos destacam-se aplicações de teleconferência, sistemas de trabalho cooperativo, controle e monitoramento de processos industriais, entre outras.

O nosso objetivo nesse trabalho é propor um serviço de tempo para essa rede metropolitana, mas que também pode ser visto como uma solução geral para sistemas abertos de larga escala. A arquitetura básica adotada neste trabalho é a de uma rede ATM composta de vários comutadores (*switch*) ATM, de placas adaptadores ATM em algumas estações de trabalho e com conectividade com redes locais do tipo *Ethernet*, *Token-Ring* (Figura 5). Essa rede nos permitirá analisar diversas situações — desde aplicações rodando apenas em nós ATM até aplicações distribuídas em nós ATM e nós de redes locais.

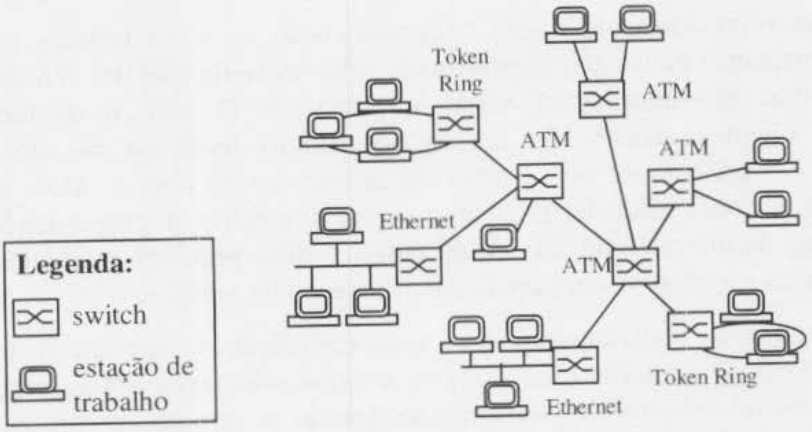

Figura 5. Rede metropolitana de Florianópolis

4.2. Implementação do serviço de tempo CORBA

A solução de *middleware* adotada para o serviço de tempo é fundamentada nas especificações CORBA. Uma das vantagens de se utilizar CORBA, é o fato desse padrão emergente facilitar a portabilidade de aplicações e a interoperabilidade de serviços, ainda que na presença de heterogeneidade de arquiteturas, de sistemas operacionais e de linguagens. Muitos dos experimentos de multimídia em desenvolvimento para a rede estão sendo definidos fazendo uso de suporte CORBA. Um serviço de tempo com as mesmas interfaces especificadas pela OMG permitirá uma integração mais fácil de aplicações CORBA sobre a rede metropolitana.

Apesar de possuir as mesmas operações que a interface do serviço de tempo básico CORBA, o serviço de tempo proposto está sendo desenvolvido no sentido de permitir a extensão fácil através de herança. Uma aplicação poderá então especificar interfaces mais adequadas às suas necessidades.

As especificações CORBA identificam duas possibilidades de implementação do serviço de tempo. Esse serviço pode ser implementado através de um servidor de tempo centralizado que atenderia as requisições de clientes remotos (Figura 6.a). Neste caso é necessária a presença de um *proxy* de servidor de tempo em cada nó cliente. O *proxy* é o representante local do serviço remoto, sendo responsável, entre outras coisas, pela adição dos atrasos no suporte de comunicação aos valores de imprecisão (*UTO.inaccuracy*, item 2) obtidos em cada leitura de relógio e que são repassados aos clientes.

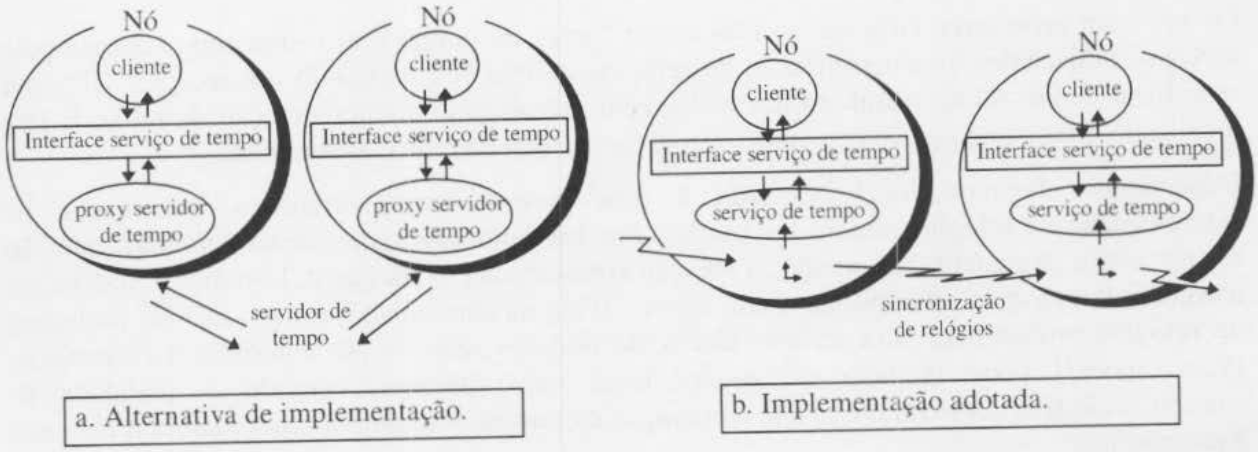

Figura 6. Alternativas de implementação do serviço de tempo CORBA.

Entretanto, a alternativa mais natural e envolvendo menores imprecisões é a que considera o serviço de tempo CORBA se localizando no mesmo nó (no mesmo espaço de endereçamento) de cada cliente (Figura 6.b). Nesse caso, as questões de suporte de comunicação (se a rede é

síncrona ou assíncrona, atrasos máximos de comunicação, etc.) são deixadas para serem tratadas em serviços subjacentes de tempo, responsáveis pela sincronização dos relógios locais dos nós, utilizando algumas das abordagens vistas na seção 3. O serviço de tempo atenderia as requisições de clientes através do acesso ao relógio local do nó. Essa alternativa de implementação é a adotada na nossa proposta de serviço de tempo. Além de aspectos como melhor precisão e confiabilidade, essa abordagem simplifica a programação das aplicações clientes; o código da interface do serviço de tempo é mais portável, pois questões dependentes de rede são tratadas no nível do serviço de sincronização de relógios.

Existem dois pontos de conformidade para a especificação do serviço de tempo [OMG 97]: *serviço de tempo básico* (*basic time service*) e *serviço de evento temporizado* (*timer event service*). Inicialmente estamos implementando apenas o serviço de tempo básico, cujas as interfaces foram descritas no item 2.2 e apresentadas na Figura 2. Estas interfaces são implementadas utilizando dois módulos CORBA denominados *TimeBase* e *CosTime* (item 2.2.1). O serviço de evento temporizado — que é descrito como um ponto de conformidade opcional na especificação — será implementado posteriormente como um segundo passo. A grande vantagem de implementar esse serviço, é que ele estende o *serviço de eventos* do CORBA no sentido de permitir a programação de tempos para ocorrência de determinados eventos, isto é, permite a construção de ativações de eventos *time-trigger*.

As funcionalidades do serviço de tempo básico CORBA estão sendo projetadas para serem implementadas na forma de uma biblioteca C++ que será ligada junto com as aplicações (clientes). Para o desenvolvimento das primeiras aplicações, que serão usadas nos testes citados do item anterior, o ORB utilizado é o ORBIX™ [IONA 96].

4.3. Sincronização de Relógios

Considerando a topologia da rede adotada (Figura 5), a implementação do serviço de tempo está baseada na existência de alguns nós especialmente preparados possuindo receptores GPS. A premissa da qual partimos é de pelo menos a existência de um receptor GPS em cada rede local ou em nó ATM, entretanto, foi descartada a possibilidade de todos os nós possuírem receptores GPS. A utilização de um receptor GPS em cada estação de trabalho praticamente elimina a necessidade de se utilizar algoritmos de sincronização de relógios. Porém, essa solução ainda não é viável atualmente em redes com um número significativo de nós. O motivo principal é devido ao fato da colocação da antena ser crítica e não poder estar situada em ambientes fechados. Também os custos do conjunto antena/cabos/receptor GPS não podem ainda ser negligenciados.

Os nós com receptores GPS são usados como fontes de tempo real (referência externa) pelo serviços subjacentes na sincronização de relógios e, uma vez definindo um receptor GPS por rede local ou nó ATM, o tráfego das mensagens envolvendo a sincronização de relógios fica localizado, praticamente não ocupando o tronco principal da rede metropolitana.

Dependendo da rede local acoplada à rede metropolitana, diferentes abordagens de sincronização de relógios podem ser usadas. Por exemplo, nas redes locais *Ethernet*, que são assíncronas e largamente difundidas, a solução apresentada em [Fetzer 97] (item 3.2.3) deve ser usada em um modelo probabilista. Cada nó envolvido na sincronização usa o método de leitura de relógios probabilista para ler os valores de relógios remotos. O algoritmo *CesiumSpray* [Veríssimo 97] pode também ser usado numa rede *Ethernet*, contudo a qualidade de sincronização tem como premissa que a operação de transmissão *time-bounded* sobre o *Ethernet* é probabilista.

Algumas poucas redes locais síncronas (redes com MACs do tipo *Token-ring*, *Token-bus*, e CSMA/DCR [LeLan 93], entre outros), usadas em controle de processo também serão acopladas à rede metropolitana. Nessas redes, porque são síncronas (comunicação delimitada), a abordagem descrita em [Veríssimo 97] deve ser usada na sincronização dos relógios locais. Para nós ligados diretamente a comutadores ATM, as abordagens hierárquicas ou mestre-escravo são

mais adequadas. Dessa forma, a abordagem de [Cristian 89] será utilizada nesses casos. Devido às facilidades de especificação de QoS das redes ATM, a utilização da abordagem probabilista de [Cristian 89] pode alcançar resultados deterministas (probabilidade 1). Basta nesse caso estipular um valor de *timeout* superior ao atraso máximo estipulado nessa rede.

A Figura 7 apresenta um exemplo de diferentes abordagens de sincronização de relógio que devem ser adotadas no serviço de tempo.

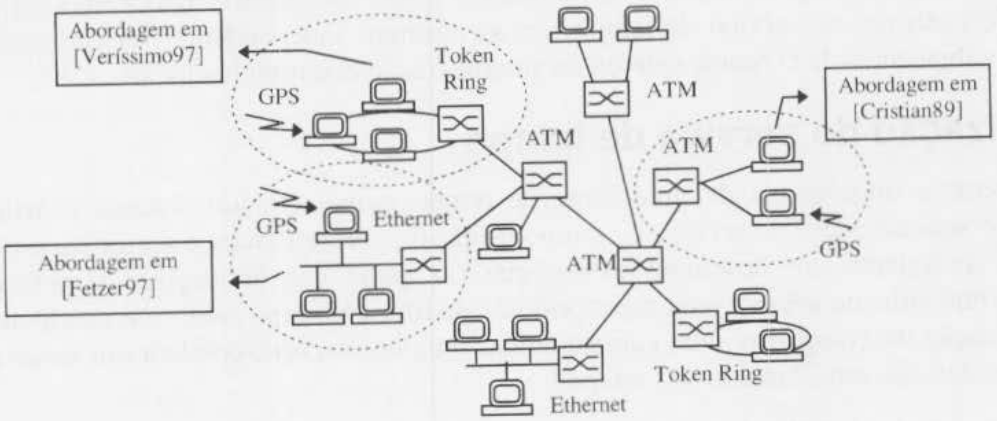

Figura 7. Sincronização de relógios na rede considerada.

A implementação nas redes locais desses algoritmos de sincronização de relógios deve usar protocolos de comunicação de baixo nível, em serviços sem conexão, evitando os custos de protocolos de mais alto nível ou mesmo a comunicação através de ORBs do CORBA.

Devido a possibilidade de relógios não conseguirem se sincronizar em abordagens probabilistas, de falhas de nós GPS, etc., o serviço de tempo pode funcionar de forma degradada em nós ou em algumas redes locais que compõem a rede de larga escala considerada. Nesse sentido, o serviço de tempo é definido supondo três modos operacionais: Global, Interno ou Local. O serviço está no modo Global quando a sincronização externa/interna é mantida. Esse é o funcionamento correto do sistema. O serviço está no modo Interno, quando apenas a sincronização interna de relógios é mantida. Esse é o caso, por exemplo, de todos os nós GPS de uma mesma rede local falharem. O serviço está no modo Local, quando mesmo a sincronização interna é perdida.

Algumas aplicações críticas com relação ao tempo, necessitam obter a informação quando o serviço de tempo não está funcionando no modo global, isto é, quando a informação de tempo atual obtida no serviço de tempo, possivelmente está se desviando do tempo real. Para sinalizar às aplicações, estamos considerando duas possibilidades. Na primeira, a invocação da operação *universal_time()* retorna uma exceção quando o serviço não se encontra no modo Global. Nesse caso, o chamador dessa operação, apesar de ser informado da qualidade da sincronização, não consegue obter o tempo atual. A segunda possibilidade, mais flexível, é o serviço subjacente de sincronização de relógios gerar sinalizações de exceção pré-definidas, através do serviço de eventos CORBA, quando da mudança do estado de sincronização. As aplicações que desejarem, podem receber esses eventos de mudança do estado de funcionamento do serviço de tempo, usando o serviço citado, para se manterem a par da "qualidade" dos tempos obtidos.

Conforme descrito em [Veríssimo 97], na sincronização de relógios em sistemas de larga escala, além da precisão interna (*precision*) e precisão externa (*accuracy*), uma outra métrica de qualidade obtida é a precisão global (*global precision*). A precisão global diz respeito à maior diferença de valores de relógios obtida entre dois nós quaisquer da rede. Uma solução homogênea, como as apresentadas em [Veríssimo 97] e em [Fetzer 97], usando um único algoritmo de sincronização de relógios nas diversas redes locais do sistema, conduz a um valor de precisão global sempre maior ou igual ao dobro da precisão externa obtida pelo algoritmo. Na nossa proposta, onde coexistem diferentes algoritmos de sincronização (solução

heterogênea), o valor de precisão global é sempre igual a duas vezes a maior precisão externa obtida entre os algoritmos envolvidos.

O serviço de tempo proposto, com base nas especificações CORBA, tem como vantagem o fato de esconder aspectos da sincronização de relógios. Na nossa abordagem, devido às diferenças entre as redes locais acopladas à rede metropolitana, o serviço de tempo além de tornar transparente toda a heterogeneidade de soluções na sincronização de relógios, integra as mesmas em interfaces padronizadas. Em qualquer ponto do sistema, as operações e abstrações disponíveis através do serviço de tempo são as mesmas, independente das características dos serviços subjacentes de comunicação ou da sincronização dos relógios locais.

5. Utilização do serviço de tempo

Para ilustrar a importância de uma fonte de tempo global em um sistema distribuído e as diversas potencialidades do serviço de tempo proposto, apresentamos a seguir exemplos tirados de um experimento em desenvolvimento que faz parte das aplicações de teleconferência previstas inicialmente sobre a rede metropolitana de Florianópolis, conforme citado no item 4.1. Essa aplicação de *groupware* com características temporais, corresponde a um leilão eletrônico similar ao descrito em [Panzieri 97] no qual:

- os participantes espalhados geograficamente devem poder participar do leilão, entrando e saindo a qualquer instante;
- os participantes devem poder submeter lances ao leiloeiro, mesmo simultâneos, como num leilão tradicional em prazo fixado pelo leiloeiro (prazo de submissão de propostas);
- os participantes devem ter a mesma visualização do objeto leiloado;
- os lances devem poder ser vistos pelos participantes com uma razoável qualidade e ouvidos de forma inteligível pelos participantes e segundo a ordem de submissão.

Na nossa aplicação, o leilão tem o leiloeiro como moderador da teleconferência, implicando numa coordenação centralizada. O leilão de um objeto é dividido em *ciclos de leilão* que iniciam com o leiloeiro propondo o *valor atual* do objeto. São estipulados dois prazos vinculados a cada ciclo de leilão : o *prazo de submissão* D_s que limita no tempo a submissão de *propostas de lance* por parte dos participantes no ciclo de leilão e, o *prazo de decisão* D_d limitando, por sua vez, a decisão do leiloeiro pelo *valor atual* final do objeto no ciclo. O prazo de decisão encerra o ciclo e, para o correto funcionamento do leilão, é necessário que $D_s \leq D_d$. O valor atual decidido no fim do ciclo com base nas propostas submetidas será o valor atual do objeto no início do próximo ciclo. O leilão termina em um ciclo onde não ocorra propostas de lance e o valor alcançado pelo objeto é o valor atual decidido no ciclo anterior, tendo como comprador o participante responsável pelo lance correspondente. A seguir apresentamos três exemplos do uso de tempo global no projeto de implementação dessa aplicação.

Primeiro Exemplo

O objetivo nesse item é destacar sobretudo a expressão das restrições temporais impostas à aplicação usando o serviço de tempo, inicialmente, nos deteremos apenas no algoritmo do leiloeiro que concentra os aspectos de coordenação da aplicação. A Figura 8, ilustra esse algoritmo. O procedimento que implementa o leiloeiro tem um parâmetro de entrada correspondente ao valor mínimo do objeto para começar o leilão (linha 1). Cada ciclo de leilão inicia com o leiloeiro enviando em mensagem aos participantes o valor atual do objeto (*ValorLance*), o dono do lance correspondente (*IdParticipante*), e um *timestamp* (T_s em tempo absoluto) que é usado como o tempo de inicio do ciclo (linha 9). A função *clock* (linhas 8 e 10) retorna o tempo atual, e *send_to_all* (linhas 9 e 21) corresponde ao envio de mensagem a todos os participantes. Na recepção de cada lance de um comprador (*PropostaLance*, linha 11), é verificado se a mesma está dentro do prazo de submissão (linha 12) e se o valor do lance é superior ao valor atual do objeto (linha 13). Verificando-se essas condições, o valor do lance e

seu proponente são armazenados como valor atual e comprador do objeto, respectivamente (linhas 14-15). Esgotado o prazo de decisão (linha 10), o leiloeiro inicia um novo ciclo (linhas 7-19). Caso não tenha recebido nenhuma proposta durante o ciclo, o leiloeiro difunde a todos os participantes a proposta vencedora, o comprador e tempo em que a proposta foi feita (linhas 20-21). Do lado dos compradores, a recepção da informação difundida pelo leiloeiro (linha 9) lhes permite verificar o interesse em fazer novo lance e as possibilidades de tempo em enviá-lo.

```
1: procedure leiloeiro ( in ValorLance : tipo_lance )
2:     const Ds : prazo de submissão;  Dd : prazo de decisão;
3:     var IdParticipante, IdVencedor, Id : tipo_id;
4:         PropostaLance, ValorVencedor : tipo_lance;
5:         Ti, Ts: tipo_tempo;  RecebeuProposta : boolean;
6: begin
     ...
7:     repeat
8:         RecebeuProposta ← false; Ti ← clock;
9:         send_to_all( ValorLance, IdParticipante, Ti);
10:        do while clock < Ti + Dd
11:            receive( PropostaLance, Id, Ts)
12:            if Ts < Ti + Ds ∧
13:               PropostaLance > ValorLance then
14:                   ValorLance ← PropostaLance;
15:                   IdParticipante ← Id;
16:                   RecebeuProposta ← true;
17:            fi
18:        od
19:    until not recebeu_proposta;
20:    ValorVencedor ← ValorLance;  IdVencedor ← IdParticipante;
21:    send_to_all( ValorVencedor, IdVencedor, clock );
       ...
22: end;
```

Figura 8. Algoritmo do leiloeiro.

Na Figura 9, é apresentada o código C++ que implementa esse algoritmo, ilustrando o uso de algumas operações das interfaces do serviço de tempo, para expressar e controlar restrições temporais da aplicação considerada. Aspectos tidos como secundários para o entendimento do problema, tais como manipulação de exceção não são apresentados no código.

A função *clock()* é usada na obtenção do valor de tempo atual (linhas 1-5). Essa função utiliza a operação *universal_time()* do módulo *Cos_Time* na atualização do objeto UTO *Agora* (linha 3). O valor retornado é do tipo *TimeT* (linha 2) e corresponde ao atributo *time* do UTO *Agora* (linha 4). Alguns objetos UTO são declarados na Figura 9 com o objetivo de representar os *timestamps Ti* e *Ts* do algoritmo representado (linha 9). As interfaces UTO, tais como todas as interfaces existentes no serviço de tempo básico, são implementadas dentro do módulo *CosTime*, daí a necessidade da declaração *CosTime::*. Da mesma forma, os prazos *Ds* e *Dd* são declarados como variáveis do tipo *TimeT* (linha 10). Esse tipo de 64 bits, implementado no módulo *TimeBase*, é usado para armazenar os valores de tempo dentro de objetos UTO, e pode ser obtido através da operação *time()*[5] (linhas 4, 14 e 16).

As comparações de tempo do algoritmo, são executadas diretamente extraindo os valores de tempo dos objetos UTO: $clock() < Ti.time() + D_d$ (linha 14) e $Ts.time() < Ti.time + D_s$ (linha 16). Nessas comparações, os valores de imprecisão (*UTO.inaccuracy*) são desconsiderados. Uma outra maneira de executar comparações de tempo em objetos UTO é através da utilização da operação *compare_time()* que será ilustrada no exemplo seguinte.

[5] O acesso a cada atributo *time* da interface UTO é feita através da função membro *time()* gerada automaticamente durante o processo de compilação da IDL e mapeamento para C++, conforme é feito no ORBIX™.

```
01: // funcao clock: retorna o tempo atual
02: TimeBase::TimeT  clock()
03: {   static CosTime::UTO Agora = CosTime::universal_time();
04:     return Agora.time();
05: }

06: // Procedure do leiloeiro
07: void leiloeiro( tipo_lance ValorLance )
08: {
09:     CosTime :: UTO    Ti, Ts;
10:     TimeBase::TimeT   Ds, Dd;
        . . .
11:     do
12:     {  RecebeuProposta = FALSE;   Ti = CosTime::universal_time();
13:        send_to_all( ValorLance, IdParticipante, Ti );
14:        while( clock() < Ti.time() + Dd )
15:        {   if( receive( PropostaLance, Id, Ts ) &&
16:                Ts.time() < Ti.time() + Ds        &&
17:                PropostaLance > ValorLance )
18:            {       ValorLance       = PropostaLance;
19:                    IdParticipante   = Id;
20:                    RecebeuProposta = TRUE;
21:            }
22:        }
23:     }while( RecebeuProposta );
24:     send_to_all( ValorLance, IdParticipante, CosTime :: universal_time() );
        . . .
24: }
```

Figura 9. Implementação do algoritmo do leiloeiro.

Segundo Exemplo

Um segundo exemplo, referente ao mesmo experimento, diz respeito a manter uma mesma percepção por parte de todos os participantes da evolução do leilão. Esta situação pressupõe a difusão das mensagens/lances entre os participantes, e também a existência de critérios de ordenação sobre as mensagens emitidas. Esses procedimentos visam criar a sensação de uma sala clássica de leilão onde cada comprador está ciente dos lances que ocorrem. Para tanto, é necessário manter as relações de ordem entre as mensagens, definidas segundo a semântica da aplicação. Por exemplo, nenhum participante pode receber a mensagem de inicio de ciclo enviada pelo leiloeiro depois de um lance referente ao mesmo ciclo de leilão. As ordens na liberação de mensagens (*message delivery*) para a aplicação, em cada estação, deve levar em conta também a dependência entre lances, ou seja, propostas são feitas após o conhecimento de lances anteriores submetidos no mesmo ciclo, sendo importante os participantes terem noção da evolução nos valores de lance no sentido de manterem uma estratégia de participação.

As ordens de origem semântica necessárias na aplicação são obedecidas, a partir do uso de um protocolo que implementa ordenações causal de mensagens levando em conta prazo de validade para as mesmas. Nesse sentido, o protocolo de *Δ-causalidade* introduzido em [Yavatkar 92] e descrito em [Baldoni 96] é utilizado em nossas aplicações de teleconferência. O prazo de validade das mensagens (o tempo relativo Δ) é controlado no protocolo a partir de etiquetas de um tempo global anexadas as mensagens quando de seus envios.

A Figura 10 descreve o algoritmo de [Baldoni 96]. Através da *Δ-causalidade*, uma mensagem m de O_j recebida em O_i (linha 9), ou é descartada porque ultrapassou o prazo Δ (linha 10) ou pode ser atrasada até sua condição de liberação (DC_m) se verificar (linhas 11-15). A condição DC_m (linhas 11 e 12) é verificada na situação em que todas as mensagens predecessoras diretas de m na ordem causal ou foram liberadas à aplicação ou perderam seus *deadlines* (ultrapassaram o prazo Δ). DC_m é testada na recepção da mensagem m usando uma informação de controle dessa mensagem, chamada barreira causal CB_m, constituída pelo conjunto de pares ordenados (n,t)

formados a partir das identificações e dos tempos de envio de todas as mensagens predecessoras de m (declarada na linha 4). Na liberação de m, a CB_i do objeto receptor O_i é atualizada retirando todos as etiquetas dos predecessores diretos de m e adicionando o identificador e a data de envio da mensagem liberada m; é desta forma que se produz o controle das predecessoras diretas das mensagens futuras emitidas por O_i (linha 14). A etiqueta de tempo de m é memorizada como sendo a data de envio da ultima mensagem transmitida de O_j para O_i (linha 13).

```
01: procedure recepcao()
02:     const Δ   : tempo de validade;
03:           np : número de participantes;
04:     type tipo_barreira_causal = ( tipo_id, tipo_tempo );
05:     var CBi[1 : np], CBm[1 : np] : tipo_barreira_causal;
06:         Tm, Del[ 1 : np] : tipo_tempo;
07:         m : tipo_lance;  id : tipo_id;
08: begin
    . . .
09:     when receive (m, id, Tm, CBm)
10:         if clock ≤ Tm + Δ  then
11:             wait  ∀(n,t) ∈ CBm: (t ≤ Del [n]) ∨
12:                                 (clock > t + Δ);
13:             Del[id] ← Tm;
14:             CBi ← CBi - CBm ∪ {id, Tm};
15:             libera(m);
16:         fi
17:     end
    . . .
18: end;
```

Figura 10. Protocolo de ordenação causal.

A Figura 11 ilustra, em um trecho de implementação C++ do algoritmo, novas manipulações das abstrações de interface do serviço de tempo. Alguns objetos UTO são declarados no sentido de auxiliar a manipulação de valores de tempo (linha 2). O objeto (UTO) *clock* é usado para armazenar valores de tempo atual, sendo atualizado pela operação *universal_time()* (linhas 5 e 12). O UTO *Tm* corresponde ao *timestamp* do instante de envio da mensagem cujo valor é obtido na recepção da mesma (linha 4). O objeto UTO *tmp* é usado nas manipulações necessárias em atributos de outros objetos quando do uso de operações de comparação. Esse UTO temporário é atualizado através da operação *new_universal_time()*, com os valores *Tm+Delta* (linha 6) e *t+Delta* (linha 10) passados como parâmetros. As comparações de tempo, são executadas fazendo uso da operação *compare_time()* da interface UTO (linhas 7, 13, 14). A constante *IntervalC* passada como parâmetro nas ativações de *compare_time()*, indica que nas comparações, os valores de imprecisão são considerados.

```
01: void recepcao()
02: {  CosTime :: UTO  clock, tmp, Tm;
03:    TimeBase :: TimeT Delta;
       . . .
04:    if( receive(m, id, Tm, CBm) )
05:    {   clock = CosTime::universal_time();
06:        tmp  = CosTime::new_universal_time(Tm.time() + Delta, Tm.inaccuracy(), . . .
07:        if( clock.compare_time(IntervalC, tmp) != TCGreatherThan )
08:        {   for( i = 0 ; i < n_elementos(CBm) ; i++)
09:            {  n = CBm[i].n;  t = CBm[i].t;
10:               tmp = CosTime::new_universal_time(t.time() + Delta, t.inaccuracy(), . . .
11:               do
12:                  clock = CosTime::universal_time();
13:                  condicao = ( t.compare_time(IntervalC, Del[n]) != TCGreatherThan) ||
14:                              clock.compare_time(IntervalC, tmp) == TCGreatherThan);
15:               while(condicao);
16:            }
17:            Del[id] = Tm;
18:            atualizaCB(CBm, id, Tm);
19:            libera(m);
20         }
21:    }
       . . .
```

Figura 11. Implementação do algoritmo de ordenação causal.

A exemplo de [Cristian 89], assumimos nesta aplicação o prazo de validade Δ das mensagens como sendo o atraso máximo que uma mensagem pode ter na sua transmissão no suporte de comunicação. Porém, na rede metropolitana, porque muitas das redes locais acopladas ao tronco principal não possuem delimitação máxima determinista nos atrasos, assumimos Δ como um limite máximo de atraso probabilista. Associada ao Δ escolhido, haverá sempre uma probabilidade de que mensagens cheguem em seus destinos após esse prazo de validade e sejam descartadas. Os prazos de submissão D_s e de decisão D_d da aplicação do leilão (primeiro exemplo) devem ter seus valores definidos de modo a permitir aos usuários do sistema tempo suficiente para acompanhar e participar em cada ciclo de leilão. Definindo esses valores de modo que $\Delta << D_s < D_d$, permite que a aplicação não trate em um ciclo de leilão com mensagens referentes a ciclos anteriores (mensagens antigas).

O protocolo de *Δ-causalidade* não garante que todas as mensagens estarão presentes em todos os participantes e muito menos uma ordem total sobre as mesmas. Mesmo assim, devido as características dos ciclos de leilão, a perda eventual de mensagens não invalida a participação dos compradores. Soluções algorítmicas *timed asynchronous*, mais estritas em termos de ordem total e de acordo na recepção de mensagens, podem ser implementadas com custos maiores em termos de desempenho.

A proposta de serviço de tempo apresentada pode ser utilizada com grande utilidade também em outros tipos de algoritmos distribuídos assíncronos temporizados. Os protocolos de engajamento atômico temporizado (*all-or-nothing timed commit protocols*) ou por maioria (*majority timed commit protocols*) [Raynal 97], ou ainda algoritmos assíncronos temporizados de *membership*, são exemplos de algoritmos que poderiam ser implementados usando o serviço de tempo.

Terceiro Exemplo

O terceiro exemplo também referente ao experimento do leilão eletrônico diz respeito a qualidade da informação de áudio e vídeo trocadas entre todos os participantes (leiloeiro e compradores). Neste experimento, como em outras aplicações cooperativas multimídias, é necessário garantir uma qualidade suficiente das mídias para permitir uma boa interatividade. Para tanto, as amostras (ou quadros) devem ser liberadas na mesma ordem de envio e dentro de limites de tempo bem definidos.

A noção de prazo de validade para as amostras permite a utilização, por exemplo, do protocolo de *Δ-causalidade* no controle da liberação das amostras, com os eventuais descartes das que tiverem os seus prazos de validade ultrapassados. Pode-se ainda definir para uma mesma mídia, um intervalo máximo entre amostras (sincronização intra-mídia). O protocolo de *Δ-causalidade* pode também condicionar a liberação das amostras à verificação desse intervalo. Existe ainda a necessidade de uma sincronização entre diferentes mídias (sincronização inter-mídia). No caso presente, a sincronização entre quadros de vídeo e de áudio (*lip synch*) é uma necessidade para os participantes, em particular o leiloeiro, poderem acompanhar melhor o leilão.

No experimento considerado, assumimos na sincronização de lábios que o quadro de áudio possa anteceder o quadro de vídeo por no máximo 150 milisegundos. Os procedimentos de recepção de mídia utilizam o protocolo de *Δ-causalidade* nos controles de liberação das amostras, verificando os prazos de validade e a sincronização intra-mídia. Para a verificação da sincronização inter-mídia, é introduzido nesses procedimentos um teste suplementar. Por exemplo, no procedimento da mídia vídeo, o teste incluído verifica a condição $T_v < T_a + 150$, onde T_v e T_a representam respectivamente os tempos de início de recepção dos quadros de vídeo e áudio associados. Uma codificação C++ desse teste de sincronização inter-mídia é apresentada na Figura 12. Objetos TIO são usados para representar a duração dos quadro de áudio e de vídeo associados (*DeltaTa* e *DeltaTv*, na linha 2). Em seguida, é verificado se o intervalo de tempo correspondente à T_v - T_a é menor que um valor máximo especificado, no caso de 150 ms (linha 4). A comparação é feita considerando os inícios dos intervalos *DeltaTv* e *DeltaTa*, através do uso da operação *time_interval().lower_bound* para obter os limites inferiores dos objetos TIO considerados.

```
01: #define ms    (* 10000)    // 1 milisegundo = 100ns * 10000
    . . .
02: void lip_sync(CosTime:: TIO DeltaTa, CosTime :: TIO DeltaTv)
03: {
    . . .
04:     if( (DeltaTv.time_interval().lower_bound < DeltaTa.time_interval().lower_bound) + 150 ms )
05:         entrega_quadros();
    . . .
```

Figura 12. Código C++ da sincronização inter-mídia.

6. Conclusões

Em sistemas distribuídos de larga escala não existe a garantia que os seus componentes sejam construídos utilizando uma mesma tecnologia (de *hardware*, de sistemas operacionais, de suporte de comunicação, etc.). Propostas de soluções para esses sistemas, para serem viáveis, devem levar em consideração essa grande heterogeneidade.

As aplicações que convivem em sistemas de larga escala necessitam em diferentes níveis, para poder garantir suas correções temporais, de um serviço de tempo global que leve em conta a natureza distribuída e a existência de relógios locais nesses sistemas. Muitas dessas aplicações requerem um tempo global que esteja relacionado com o padrão UTC principalmente pelas dimensões da distribuição geográfica envolvida.

Nesse artigo foi apresentada uma proposta de serviço de tempo global para sistemas distribuídos de larga escala. As especificações CORBA de serviço de tempo são adotadas como interfaces do serviço proposto permitindo ao mesmo interfaces bem definidas. A solução padronizada das interfaces do serviço de tempo permite uma grande flexibilidade, adaptando-se facilmente às evoluções tecnológicas. Acrescenta-se ainda que a utilização da padronização CORBA traz diversas vantagens adicionais, tais como a possibilidade de que as interfaces especificadas sejam herdadas pelas aplicações, possibilitando-as acrescentar novas operações adequadas à cada utilização específica; além da vantagem da linguagem de descrição de interfaces (IDL) de

CORBA permitir que as operações das interfaces sejam invocadas por diferentes linguagens de programação.

Por outro lado, com a crescente disponibilidade de receptores GPS surgiram recentemente novos algoritmos de sincronização de relógios apropriados para sistemas de larga escala, tais como os apresentados em [Veríssimo 97] e [Fetzer 97]. A questão básica que guia esses novos algoritmos é a integração de técnicas de sincronização externa e interna em uma só estrutura algoritmica. Considerando as características de cada rede local acoplada à rede metropolitana, diferentes algoritmos são adotados na nossa abordagem para a sincronização dos relógios locais. No serviço proposto, várias soluções de sincronização de relógios podem co-existir, dependendo das características das redes locais acopladas à rede metropolitana. Essas soluções de sincronização, por se utilizarem de receptores GPS na sincronização externa, definem trocas de mensagens localizadas na topologia da rede.

O serviço de tempo proposto, tomando como base as abstrações definidas nas especificações CORBA, tem como vantagem o fato de esconder aspectos da sincronização de relógios. O serviço de tempo além de tornar transparente toda a heterogeneidade de soluções na sincronização de relógios, integra os diferentes algoritmos em interfaces padronizadas. Ocultar das aplicações os detalhes e as diferenças nesses mecanismos de sincronismo de relógios é requisito central para facilitar o desenvolvimento de aplicações em sistemas de distribuídos larga escala.

O serviço de tempo global discutido nesse texto faz parte do projeto de uma rede metropolitana experimental em estágio de desenvolvimento.

7. Referências

[Baldoni 96] R.Baldoni, R. Prakash, M. Raynal, M. Singhal; *Broadcast with time and causality constraints for multimedia applications*, 22nd EUROMICRO Conference, Sep. 1996.

[Cristian 89] F. Cristian, *Probabilistic Clock Synchronization*, Distributed Computing, Springer-Verlag, Vol. 3, 1989, pp. 146-158.

[Cristian 95] F. Cristian, C. Fetzer, *Probabilistic Internal Clock Synchronization*, Internal Report at http://www-cse.ucsd.edu/users/flaviu, Dec. 1995.

[Cristian 96] F. Cristian, *Synchronous and Asynchronous Group Communications*, Communications of ACM, Jan. 1996.

[Dana 97] P. H. Dana, *Global Positioning System (GPS) Time Dissemination for Real-Time Applications*, Real-time Systems, 12, 1997, pp. 9-40.

[Fetzer 97] C. Fetzer, F. Cristian, *Integrating External and Internal Clock Synchronization*, Real-time Systems, 12, 1997, pp. 123-171.

[Herbert 94] A. Herbert, *An ANSA Overview*, IEEE Network, Jan./Feb. 1994, pp. 18-23.

[IONA 96] IONA Technology, *ORBIX Programmer's Guide and Reference Manual*, 1996.

[Johnson 94] B. C. Johnson, *A Distributed Computing Environment Framework: An OSF Perspective*, In Distributed Open Systems Edited by F. M. T. Brazier and D. Johansen, IEEE Computer Society Press, 1994, pp. 57-77.

[Kopetz 89] H. Kopetz, W. Schwabl, *Global Time in Distributed Real-Time Systems*, Research Report No. 15/89, Institute für Technische Informatik, Austria, Oct. 1989.

[LeLann 93] G. LeLann, N. Rivière, *Real-Time Communications Over Broadcast Networks: the CSMA-CDR and DOD-CSMA-CD Protocols*, Technical report 1863, INRIA, Mar. 1993.

[Marzullo 84] K. Marzullo, *Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service*, Ph.D Thesis, Dept. of Electrical Engineering, Stanford University, 1984.

[Mills 91] D. L. Mills, *Internet Time Synchronization: the Network Time Protocol*, IEEE Trans. Communications 39, 10, Oct. 1991, pp.1482-1493.

[OMG 95] OMG, *Common Object Request Broker: Architecture and Specification*, rev. 2.0, Jul. 1995.

[OMG 97] OMG, *Time Service Specification*, Document adopted/97-02-22, Feb. 1997.

[Panzieri 97] F. Panzieri, M. Roccetti, *Synchronization support and group-membership services for reliable distributed multimedia applications*, Multimedia Systems (ACM) 1997, vol.5, pp. 1-22, Sep. 1997.

[Raynal 97] M. Raynal, *Real-time Dependable Decisions in Timed Asynchronous distributed systems*, IEEE Workshop WORDS 97, Newport Beach, CA, Feb. 1997, pp. 283-290.

[Schmid 97] U. Schmid, K. Schossmaier, *Interval-based Clock Synchronization*, Real-time Systems, 12, 1997, pp. 173-228.

[Schneider 86] F. Schneider, *A Paradigm for Reliable Clock Synchronization*, Proc. Advanced Seminar on Real-Time Local Area Networks, pp. 85-104, INRIA, April, 1986.

[Stankovic 96] J. A. Stankovic et al., *Strategic Directions in Real-Time and Embedded Systems*, ACM Computing Survey, Vol. 28, No. 4, Dec. 1996, pp. 751-763.

[Verissimo 92] P. Veríssimo, L. Rodrigues, *A Posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks*, 22nd International Symposium on Fault-Tolerant Computing, Jul. 1992.

[Verissimo 97] P. Veríssimo, L. Rodrigues, A. Casimiro, *CesiumSpray: A Precise and Accurate Global Time Service for Large-scale Systems*, Real-time Systems, 12, 1997, pp. 243-294.

[Yavatkar 92] R. Yavatkar, *MCP: A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications*, 12th IEEE Conference on Distributed Computing Systems, pp. 606-613, May 1992.