ReMMoS - Um Modelo de Replicação em Ambientes que Suportam Mobilidade de Objetos

Débora Nice Ferrari[1], Patrícia Kayser Vargas[1], Cláudio Fernando Resin Geyer[1]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – CEP 91591-970 Porto Alegre - RS – Brasil
{nice, kayser, geyer}@inf.ufrgs.br

Abstract—
Mobility and replication are frequently used in distributed systems composed by heterogeneous network, distant and connected by different hosts of communications. However there are few systems that deal with both techniques mainly in distributed object environments. This paper presents the ReMMoS – Replication Model in Mobility Systems. The replication in distributed systems can be used to offer security and reliability and also to increase execution performance in applications. This paper presents a replication model in distributed object systems with mobility aiming to get better performance. The mobility is explicit and the replication is implicit. So, programmers do not need to worry about management and consistency of the replicated copies. The model's implementation is in progress and is briefly presented.

Keywords— **Distributed Object, Mobility Systems, Object Replication**

I. INTRODUÇÃO

O desenvolvimento de aplicações distribuídas intensificou-se com o uso em grande escala das redes de longa distância (WAN - *Wide Area Network*), cujo principal exemplo é a Internet. Esta trouxe a possibilidade de desenvolver aplicações através de redes heterogêneas, conectadas por diferentes enlaces de comunicação e distantes uma das outras [FUG 98].

Dois tópicos freqüentemente estudados nesse contexto são mobilidade e replicação [FUG 98], [KLE 96]. A mobilidade permite mover uma entidade computacional de uma máquina para outra através do sistema. Portanto, em aplicações que usam mobilidade é necessária a preocupação com a integridade das entidades envolvidas. A replicação permite que cópias de uma entidade computacional possam existir no sistema, apresentando-se assim como uma forma de obter disponibilidade. Sendo assim, recentemente surgiram vários trabalhos envolvendo mobilidade e replicação [KLE 96, ION 96 e RAT 96]. A replicação envolve a manutenção da consistência entre as múltiplas cópias, isto é, é preciso garantir que todas as cópias possuam o mesmo estado. Por isto, o desempenho do sistema pode ser diminuído devido ao tráfego incrementado pela garantia de consistência. Desta forma, é favorável utilizar políticas de posicionamento de réplicas que visam a reconfiguração do sistema com base na disponibilidade e desempenho requeridos pela aplicação.

A maioria dos trabalhos envolvendo mobilidade ou replicação não envolvem ambientes de objetos distribuídos. Além disso, até o momento, não há o conhecimento de trabalhos envolvendo mobilidade e replicação de objetos, em um único modelo, com objetivo de prover melhor desempenho ao sistema. Sendo assim, este trabalho propõe um modelo de mobilidade e replicação em ambientes de objetos distribuídos. A mobilidade fica a cargo do desenvolvedor. A replicação é feita de forma transparente, facilitando o trabalho do desenvolvedor quando este necessita deste recurso em sua aplicação. Assim, este não precisará preocupar-se com o gerenciamento e consistência das réplicas. Além disto, este modelo visa propor um desempenho satisfatório para as aplicações do usuário.

O texto está organizado da seguinte forma: na seção 2 destaca-se aspectos de mobilidade e replicação em sistemas distribuídos. Na seção 3 o modelo ReMMoS é apresentado. Finalmente, na seção 4, as conclusões do trabalho são apresentadas.

II. MOBILIDADE E REPLICAÇÃO EM SISTEMAS DISTRIBUÍDOS

A maioria das pesquisas que envolvem mobilidade referem-se a esta através do termos código móvel (*mobile code*) [FUG 98, CAR 97, CUG 96] e agentes móveis (*mobile agent*) [CHE 96, KNA 96]. Não existe ainda um consenso com relação a definição de mobilidade. Entretanto, o deslocamento das entidades computacionais envolvidas pode ser notada em todos os conceitos. Desta forma, pode-se dizer que mobilidade é a capacidade das entidades envolvidas na computação deslocarem-se através de um sistema distribuído [FER 99b].

O texto [FUG 98] destaca duas formas principais de mobilidade: **mobilidade forte** e **mobilidade fraca**. Mobilidade forte permite que tanto o código quanto o estado da computação possam ser movidos para diferentes ambientes computacionais. Mobilidade fraca permite transferir código de um ambiente computacional para outro.

O código pode conter alguns dados inicializados, mas o estado da computação não é movido. Por exemplo, em Java é possível mover somente o código de um Applet [SUN 00]. Já em Java Aglet [LAN 97], tanto o código quanto o estado são movidos. Outros trabalhos relacionados podem ser encontrados em [CHE 96, CAS 98, FUG 98, GLA 98, OBJ 00 e ROY 97].

A replicação é uma estratégia utilizada para distribuição de dados compartilhados, permitindo que várias cópias da entidade computacional residam em diferentes memórias locais. É utilizada principalmente para habilitar acessos simultâneos de diferentes nodos ao mesmo dado [JAL 94]. Um problema fundamental neste caso é a necessidade de manter a consistência da informação. Duas estratégias básicas são usadas para isto: **invalidação** e **atualização** [LI 86].

A replicação de um objeto somente é útil se este é freqüentemente consultado por um nodo remoto. Em casos que na aplicação ocorrem tanto leituras quanto escritas, é necessário uma análise do comportamento da aplicação, em tempo de compilação e execução. Em geral, pode-se distinguir entre três estratégias para replicação: não replicar, replicação total e replicação parcial [BAL 92]. Estas estratégias utilizam modelos e algoritmos que permitem gerenciamento da consistência do objeto replicado (relacionados também à estudos sobre DSM (*Distributed Shared Memory*)).

Com relação aos algoritmos, os principais referem-se à protocolos como SRSW (*Single Reader / Single Writer*), MRSW (*Multiple Readers / Single Writer*), MRMW (*Multiple Readers / Multiple Writers*) [LI 89], [PRO 98], [TAN 95]. Dos principais modelos existentes, pode-se destacar modelos utilizados para implementar consistência de réplicas tais como *Sequential Consistency*, *Weak Consistency*, *Release Consistency*, *Lazy Release Consistency* e *Entry Consistency* [LAM 79], [ADV 90], [TAN 95]. Além disso, técnicas como Primário-Backup, Réplicas Ativas, Votação e Combinação [JAL 94] podem ser destacadas. Este trabalho utiliza a técnica Primário-*backup*, implementando um algoritmo de consistência de réplicas do tipo MRSW.

Em ambientes que suportam mobilidade de objetos, não é comum a replicação em nível de entidades computacionais. Sendo a migração e a replicação um tipo de mobilidade forte, então pode-se destacar vários trabalhos que envolvem estes dois assuntos, geralmente relacionados a gerenciamento de arquivos, tais como [HUR 96, HAC 89, KUR 88 e SHE 86]. A maioria dos trabalhos que tratam de replicação, a utilizam para prover tolerância a falhas. Neste caso, o desempenho não deve ser fator determinante. Outros trabalhos abordam apenas mobilidade.

III. O MODELO REMMOS - REPLICATION MODEL IN MOBILITY SYSTEMS

A mobilidade tem como principal vantagem manter a localidade dos objetos que trocam muitas mensagens, diminuindo assim o tráfego na rede. A replicação tem como vantagem permitir que várias cópias de uma mesma entidade computacional residam em diferentes máquinas do sistema, podendo haver acesso simultâneo de diferentes nodos à mesma entidade computacional. Os problemas que este modelo tenta solucionar correspondem a como permitir replicação em um ambiente de objetos distribuídos que suporta mobilidade e como não interferir de forma significativa no desempenho da aplicação. Através da redução do custo de comunicação e a adoção de uma política de gerenciamento de réplicas simples e eficiente, parece haver um desempenho concreto a ser atingido. A figura 1 destaca a organização do modelo ReMMos.

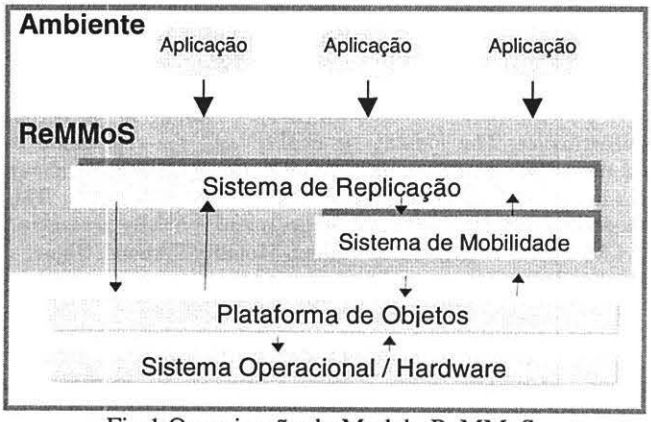

Fig.1 Organização do Modelo ReMMoS

O sistema de replicação é responsável por toda a gerência dos objetos replicados. Desta forma, deve controlar o acesso aos objetos potencialmente replicáveis. O sistema de mobilidade é o responsável por controlar a mobilidade dos objetos não-replicados e replicados. A mobilidade de objetos replicados deve ser feita em conjunto com o sistema de replicação.

A. *Decisões quanto à concepção do modelo*

A identificação de operações de consulta ou atualização ocorre somente em objetos que podem ser serializados. Para isto, é necessário que o código da aplicação seja modificado, de forma que seja identificado se um método tem função de consulta ou de atualização. Se o método tem alguma função de atualização ele é considerado um **método de escrita**. Neste caso, deve ser incluído no código uma chamada ao sistema de replicação que controla requisições de atualização. Se o método tem função de consulta, ele é considerado um **método de leitura**. Assim sendo, deve ser incluído no código uma chamada ao sistema de replicação

que controla requisições de consulta. Embora este tópico seja um assunto importante para este trabalho, o ReMMoS apenas destaca quais informações são necessárias ao modelo. Estas informações devem ser fornecidas por um analisador de código. Uma requisição de leitura a um objeto replicado pode ser tratada de duas formas:

- O cliente **não possui** uma cópia local do objeto replicado. O acesso para leitura deve ser feito remotamente, diretamente no objeto replicado. O sistema de replicação é encarregado de monitorar as requisições ao objeto replicado, pois pode haver necessidade de criar uma cópia na máquina cliente;
- O cliente **possui** uma cópia local do objeto. Assim, a leitura ocorre localmente. O sistema de replicação é encarregado de encaminhar o processamento da consulta para a réplica local. Novas consultas serão realizadas diretamente na réplica local até que uma exceção ocorra e o cliente tenha que reportar-se remotamente ao objeto replicado.

Quando uma réplica do objeto replicado recebe uma requisição de escrita, esta requisição deve ser tratada de forma especial, de modo a manter a consistência do objeto replicado. Neste caso, a atualização ocorre da seguinte forma: quando uma réplica recebe um pedido de atualização, a mesma deve ser bloqueada e a requisição de atualização encaminhada à réplica primária. A partir deste momento o protocolo de atualização segue da seguinte forma:

1. ao receber a mensagem, a réplica primária do objeto replicado envia para as outras réplicas uma mensagem contendo uma ordem de bloqueio das mesmas e os parâmetros de atualização;
2. terminada a operação de atualização, as réplicas mandam uma mensagem de retorno à réplica primária, indicando que seus estados já encontram-se consistentes;
3. a réplica primária manda uma mensagem de desbloqueio para as demais réplicas assim que receber a mensagem de retorno de todas elas;
4. ao ser desbloqueada, a réplica que recebeu a solicitação de atualização por parte do cliente, responde ao mesmo o seu pedido.

B. *Controle Dinâmico do Número de Réplicas*

O modelo de replicação apresentado neste trabalho é adaptativo ao tipo de aplicação, visto que não restringe a característica da aplicação. Para isto, conforme o comportamento da aplicação, um objeto replicado pode ter suas réplicas criadas ou descartadas. Desta forma, quando a aplicação se comporta de modo a predominar consultas, as réplicas são criadas. Se predominar atualizações, as réplicas ociosas do objeto vão sendo descartadas, para diminuir o custo de atualização do objeto replicado, uma vez que este custo cresce à medida que aumentam as réplicas [BAL 92]. Entende-se por réplica ociosa a que não está processando, por um determinado período de tempo, consultas locais. No pior caso, se predominar no sistema atualizações sobre os objetos replicados, estes tornam-se objetos não-replicados, pois não existirão mais réplicas associadas a ele.

O método utilizado neste trabalho para determinar se uma réplica será descartada inspira-se no modelo LRU (*Least Recently Used*), utilizado para gerência de páginas de memória [TAN 92]. As réplicas de um objeto replicado são descartadas à medida que sucessivas operações de atualização incidem sobre o objeto replicado. Desta forma, inspirando-se no método LRU, a réplica que está ociosa por mais tempo é descartada. Cada réplica do objeto replicado possui associada a si um **contador de acesso local**. Este contador é incrementado a cada solicitação de **consulta local** à réplica do objeto replicado. Entendendo ser vantagem uma réplica existir somente se está sendo acessada localmente, o uso do contador visa controlar se a réplica está ociosa ou não. Assim, a cada operação de atualização, o contador de acesso local é verificado. Após um determinado período de tempo, a réplica que apresenta o contador com o número de acessos de leitura constante é descartada. Desta forma, permanecem no sistema somente réplicas que estejam localmente ativas, evitando a atualização desnecessária de réplicas que não estão sendo localmente usadas. Caso todas as réplicas do objeto replicado sejam descartadas, este objeto será considerado não-replicado. Desta forma, requisições para este objeto serão remotas, e não mais locais. Esta parte do trabalho está sendo implementada. Testes estão sendo feitos para determinar qual a melhor heurística para descartar as réplicas ociosas.

C. *Modelo de Mobilidade*

O modelo de mobilidade tem como objetivo manter a localidade de objetos que trocam muitas mensagens entre si. Neste trabalho, estes objetos são chamados de **objetos não-replicados**. Desta forma, o modelo diferencia objetos **não-replicados** e **objetos replicados**. Como a mobilidade é **explícita**, objetos não-replicados podem ser movidos conforme decisão do programador e o modelo de replicação não interfere na mobilidade do objeto se este não é replicado. O modelo de mobilidade utilizado no sistema Voyager [OBJ 00] foi escolhido para a mobilidade de objetos não-replicados. O modelo de Voyager é simples e eficiente, integrando vários conceitos consolidados quando se trata de mobilidade em sistemas distribuídos, tais como o uso de *proxies* e a possibilidade de desenvolvimento usando mobilidade forte e fraca, por exemplo. Objetos replicados recebem um tratamento especial quanto a mobilidade, pois sendo a replicação implícita, o programador pode mover um objeto que já está replicado.

D. Mobilidade de Objetos Replicados

A mobilidade de um objeto replicado requer uma atenção especial. Como a mobilidade é explícita e a replicação implícita, o desenvolvedor pode mover um objeto que está replicado. Quando um objeto replicado é movido, sua referência deve ser alterada para a nova localização. Desta forma, procura-se manter a transparência quanto à replicação. Em caso de mobilidade de objetos replicados, duas possibilidades podem ocorrer:

1 - **Existe uma réplica do objeto no nodo destino**: neste caso, a réplica no endereço destino torna-se a nova referência para o objeto replicado (as demais réplicas são informadas na próxima requisição de atualização no objeto replicado). O objeto é destruído no endereço antigo e um *proxy* para o objeto é criado neste endereço. Este *proxy* é necessário para encaminhar as mensagens que o objeto recebe no endereço antigo para o novo endereço;

2 - **Não existe uma réplica do objeto no nodo destino**: neste caso, o objeto replicado move-se para o nodo destino utilizando um protocolo semelhante ao usado em objetos não-replicados. A réplica no endereço destino torna-se a nova referência para o objeto replicado. O objeto é destruído no endereço antigo e um *proxy* para o objeto é criado neste endereço. Este *proxy* é necessário para encaminhar as mensagens que o objeto recebe no endereço antigo para o novo endereço.

Como o objeto replicado pode alterar seu endereço por motivos de mobilidade, é necessário que as réplicas do objeto replicado sejam informadas da nova localização. Assim, o protocolo de atualização é responsável por informar também a nova localização do objeto. Sempre que o objeto replicado mudar seu endereço por motivo de mobilidade, a próxima alteração neste objeto atualiza seu endereço nas suas réplicas. Se o pedido de atualização incidir sobre uma réplica do objeto replicado antes da mesma ter sido informada sobre a nova localização do objeto, o sistema de replicação, através do *proxy* do objeto, é responsável por encaminhar a requisição até a nova localização do objeto.

E. Implementação do Modelo

O ReMMoS encontra-se em fase de prototipação. A linguagem usada para implementar o modelo ReMMoS é a linguagem Java [SUN 00]. Além da sua portabilidade e segurança, ela oferece a possibilidade de implementar tanto mobilidade, quanto replicação de objetos. A mobilidade é implementada usando o sistema Voyager [OBJ 00]. A escolha deste como sistema base para a mobilidade deu-se por várias razões:

- é um sistema projetado para utilizar a linguagem Java;
- várias aplicações distribuídas que suportam mobilidade estão sendo desenvolvidas com Voyager;
- possibilidade de obter uma versão gratuita (*freeware*) do sistema;
- suporte técnico facilitado;
- não possui replicação de objetos.

Comprovando a portabilidade de Java, tem-se usado tanto sistema operacional Conectiva Linux 5.0 quanto Solaris 5.7.

IV. Conclusões

O objetivo principal deste trabalho consistiu em apresentar um modelo de mobilidade forte e replicação em ambientes de objetos distribuídos. As principais características são:

- permitir o desenvolvimento de sistemas distribuídos orientados a objetos com recursos de mobilidade e replicação;
- a mobilidade é explícita. Sendo assim, o programador é responsável por especificar qual objeto vai ser movido e qual sua origem;
- a replicação e sua gerência são transparentes para o usuário;
- preocupa-se com um desempenho satisfatório das aplicações com mobilidade. Assim, além de replicar o objeto, o modelo busca a melhor forma de replicar este objeto, visando desempenho.

Vários estudos ainda são necessários para aprimorar o modelo, tais como o uso de um analisador de código para inferência de informações ao modelo, análise do comportamento do modelo na presença de aplicações com características diversas e o estudo do comportamento do modelo com outros tipos de sistemas que permitem mobilidade (além de Voyager).

V. Referências

[ADV 90] ADVE, Sarita V.; HILL, M. D. Weak Ordering – A New Definition. In ANUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, ISCA, 1990. Proceedings.

[BAL 92] BAL, Henri E. et al. Orca: A Language for Parallel Programming of Distributed Systems. *IEEE Transactions on Software Engineering*, New York, v. 18, n. 3, Mar 1992, p. 190-205.

[BAL 98] BAL, Henri E. et al. Performance Evaluation of the Orca Shared Object System. *ACM Transactions on Computer Systems*, New York, v. 16, n. 1, Feb. 1998.

[CAS 98] CASTILLO, A.; KAWAGUCHI, M. et al. Concordia as Enabling technology for cooperative Information Gathering. Mitsubishi Eletric ITA, USA, 1998.

[CAR 97] CARZANIGA, A.; PICCO, G.; VIGNA, G. Designing distributed applications with a mobile code paradigm. 19th INTERNATIONAL CONFERENCE ON

SOFTWARE ENGINEERING. *Proceedings...* Boston, may 1997.

[CUG 96] CUGOLA, G.; GHEZZI, C.; PICCO, G. P.;VIGNA, G. Analyzing Mobile Code Languagens. SECOND INTERNATIONAL WORKSHOP MOBILE OBJECT SYSTEMS, MOS'96. *Proceedings...* Linz, Austria, july 1996. Lecture Notes.

[CHE 96] CHESS, D.; HARRISON, C.; KERSHENBAUM, A. Mobile agents: are they a good idea? SECOND INTERNATIONA WORKSHOP MOBILE OBJECT SYSTEMS, MOS'96. *Proceedings...* Linz, Austria, july 1996. Lecture Notes.

[FER 99b] FERRARI, Débora N. Um estudo sobre mobilidade em sistemas distribuídos. CPGC - II - janeiro, 1999. Trabalho Individual

[FUG 98] FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding Code Mobily. *IEEE Transactions on Software Engineering*. Vol.24, num. 5, May 1998.

[GLA 98] GLASS, Graham. ObjectSpace Voyager - The Agente ORB for Java. WORLDWIDE COMPUTING AND ITS APPLICATIONS (WWCA'98). Second International Conference. *Proceedings...* Tsukuba, Japan, march, 1998.

[HAC 89] HAC, A. A Distributed Algorithm for Performance Improvement Through File Replication, File Migration and process Migration. *IEEE Trans. Software Engineering*, vol. 15, num 11, págs 1459-1470. Nov, 1989.

[HUR 96] HURLEY, R. T. File Migration and File Replication: A Symbiotic Relationship. *IEEE Transaction on Parallel and Distributed Systems*. Vol. 7, nº 6, págs. 578-586. june, 1996.

[ION 96] IONITOIU, Cristian et al. Replicated Objects with Lazy Consistency Second International Workshop, ECOOP'96. *Proceedings...* Politechnica Univeristy of Timisoara.

[JAL 94] JALOTE, P. *Fault Tolerance in Distributed Systems*. Ecolo Pelytechnique Fédérale de Lausanne, 1996.

[KLE 96] KLEINÖDER, J.; GOLM, M. *Transparent and Adaptable Object Replication Using a Reflexive Java*. Computer Science Department. University of Erlangen-Nürnberg. Erlangen, Germany. September, 1996. (technical report).

[KNA 96] KNABE, F.; Na Overview of mobile agente programming. FIFTH LOMAPS WORKSHOP ON ANALYSIS AND VERIFICATION OF MULTIPLE AGENT LANGUAGES. *Proceedings...* Stockholm, Sweden, june 1996. Lecture Notes.

[KUR 88] KURE, Q. *Optimization of File Migration in Distributed Systems*. PhD Thesis, University of California. Berkeley, 1988.

[LAN 97] LANGE, Danny, et al. Aglets: programming Mobile Agents in Java. WORLDWIDE COMPUTING AND ITS APPLICATIONS (WWCA'97). INTERNATIONAL CONFERENCE. *Proceedings...* Tsukuba, Japan, March, 1997.

[LAM 79] LAMPORT, L. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Transaction on Computers*, v.28, n. 9, p.690-691, sep. 1979.

[LI 89] LI, KAI; HUDAK, Paul. Memory Coherence in Shared Virtual Memory Systems .*ACM Transaction on Computer Systems*. New York, V.7, n. 4, p.321-359, nov.1989.

[LI 86] LI, KAI. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. New Haven, Conn.: Dept. of Computer Science, Yale University, 1986. Ph.D. dissertation.

[OBJ 00] ObjectSpace Voyager Core Tecnology 3.0 Disponível em http://www.objectspace.com

[ORF 96] R. Orfali, D. Hankey, J. Edwards, *The Essential Distributeds Objects Survival Guide*, John Wiley & Sons, Inc.1996.

[OSH 98] OSHIMA, M. ; LANGE, D. *Mobile Agentes with Java: The Aglet API*.

[PRO 98] PROTIC, Jelica et. Al. Distributed Shared Memory: Concepts and Systems. Los Alamitos, California: *IEEE Computer Society Press*, 1998

[RAT 96] RATNER, D.; POPEK, G. J; REIHER, P. *The Ward Model: A Scalable Replication Architecture for Mobility*. (technical report)

[ROY 97] ROY, Peter et al. Mobile Objects in Distributed Oz. *ACM Transactions on Programming Languages and Systems*, v.19, n.5, p.804-851, September 1997.

[RUM 96] RUMBAUGH, James; et al.; *Modelagem e Projetos baseados em Objetos*. Editora Campus, Rio de Janeiro, 1996.

[SHE 86] SHENG, Lui O. R. *Models for Dynamic File Migration in Distributed Computer Systems*. PhD Thesis. University of Rochester, 1986.

[SUN 00] Sun Microsystems. *The Source for Java tecnology*. Disponível em http://java.sun.com/

[TAN 95] TANENBAUM, Andrew S. Distributed Operation Systems. Prentice Hall, New Jersey, 1995.

[TAN 92] TANENBAUM, Andrew S. et al. Parallel Programming using Shared Objects and Broadcasting. *IEEE Computer*, New York, v. 25, n. 8, p. 10-19, Aug. 1992