

Persistência de Componentes num Ambiente de Reuso

Marta Mattoso, Cláudia Werner, Regina Braga, Robson Pinheiro, Leonardo Murta, Victor Almeida,
Marcelo Costa, Eduardo Bezerra, Jorge Soares, Nicolaas Ruberg

COPPE/UFRJ - Engenharia de Sistemas e Computação
Caixa Postal 68511 – Rio de Janeiro, 21945-970 - Brasil
goa@dbd.cos.ufrj.br <http://www.cos.ufrj.br/~goa>

1. Introdução

Ambientes de apoio ao reuso de software, necessitam de recursos de armazenamento e manipulação de diversos tipos de componentes ligados ao Desenvolvimento Baseado em Componentes (DBC). Uma das características de um componente é a complexidade da sua representação (podendo ser objeto longo), em particular, dos relacionamentos entre eles. Aliado à necessidade de um modelo capaz de representar e armazenar componentes, está a manipulação de grandes coleções de componentes e a navegação entre instâncias de componentes. Tais requisitos praticamente inviabilizam o uso de sistemas de arquivo para a persistência, devido à falta de flexibilidade do armazenamento e manipulação além de impor uma gerência desses componentes em memória para manipular coleções grandes. Sendo assim, um SGBD (Sistema de Gerência de Bases de Dados) surge como uma solução mais indicada. Entretanto, o modelo de representação relacional não se mostra adequado, pois é deficiente quanto ao poder de representação de relacionamentos complexos e de herança, por exemplo. Tais restrições não ocorrem em SGBDs com capacidade de representação do modelo de objetos. Nesse sentido, estamos propondo o uso do servidor de gerência de objetos GOA++ [4] como uma ferramenta para oferecer persistência e manipulação de componentes típicos (ex.: classe, use case, diagrama de estado, etc.) de um DBC.

A vantagem principal em utilizar o GOA++ ao invés de um produto de mercado, seja ele orientado a objetos(OO) ou relacional objeto(RO) consiste em sua arquitetura aberta e flexível. Assim, o GOA++ pode ser configurado para prover os serviços essenciais para o DBC sem a sobrecarga do ambiente de um SGBD produto por exemplo. O GOA++ vem sendo utilizado com sucesso infra-estrutura de reuso Odyssey [1]. Outro ponto importante do GOA++ é sua aderência a padrões OO, em particular ao ODMG [7]. O padrão ODMG contempla as construções básicas e avançadas do modelo OO, tais como: uso de identificador único, herança múltipla, persistência por extensão da classe, relacionamentos inversos, entre outros, além de ser compatível com o modelo do OMG e conseqüentemente com a notação UML.

Através da OQL (*Object Query Language*) obtém-se recuperação e manipulação flexível de componentes representados segundo o modelo de objetos e navegação entre relacionamentos multivalorados através de expressões de caminho. Esse tipo de manipulação ainda é muito incipiente em sistemas RO. O GOA++ conta ainda com índices sobre coleções tanto com herança quanto para expressões de caminho [3]. Além disso, no caso do Odyssey e outros ambientes de apoio ao DBC, os componentes são agrupados por domínio e distribuídos. Portanto, a capacidade de gerência de objetos distribuídos do GOA++[2] é uma opção natural, trazendo maior eficiência para o DBC.

Apresentamos na seção 2 características gerais do GOA++, porém com ênfase em extensões que foram feitas à versão apresentada em [6] para apoiar melhor o DBC. Na seção 3 apresentamos a incorporação de mecanismos de remoção explícita de objetos, bastante útil para DBC e normalmente ausente em SGBDOOs. Na seção 4 é apresentado o novo modelo de manipulação de objetos em memória, dando ênfase à eficiência de cache para relacionamentos complexos. Apresentamos um exemplo de uso real do GOA++ junto ao Odyssey na seção 5.

2. Modelo de Classes do GOA++

A Figura 1 descreve o modelo de classes em notação UML do Gerente do Objetos GOA++. A classe Socket é responsável por receber as requisições de clientes para acesso ao servidor. A classe GoaServer interpreta as requisições solicitadas, repassando-as à classe Goa, que gerencia todo o processo de criação de esquemas e de acesso ao servidor de páginas. Para recuperar um objeto do disco pelo seu identificador único (Oid), a classe PageServer solicita à Cache a página a qual este pertence, ou a busca diretamente no disco, caso esta não esteja na Cache. A nova página carregada do disco é então colocada na Cache e o objeto desejado posto na lista de objetos em memória (classe ObjectMemoryList). O GOA++ permite a manipulação de objetos curtos (GoaInstance), objetos longos (GoaBlob) e de coleções de objetos (GoaCollection).

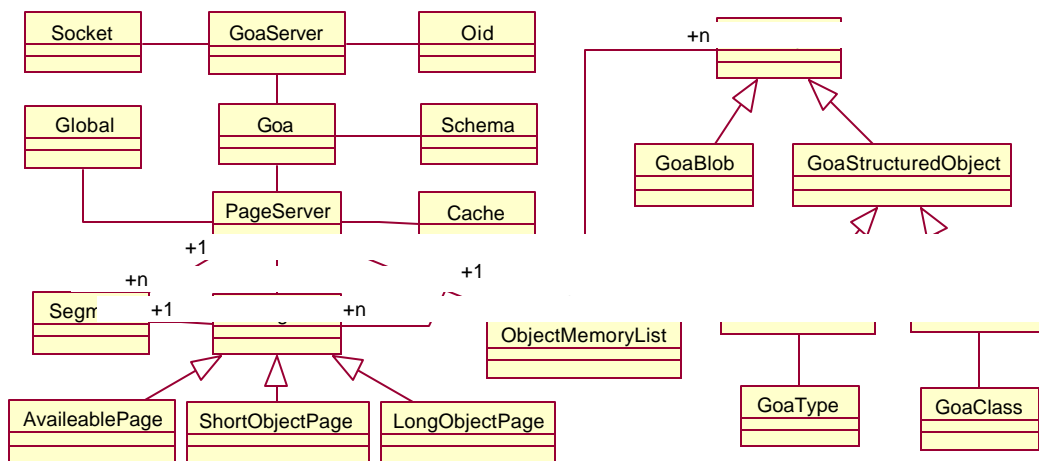


Figura 1 - Modelo de classes do GOA++

3. Remoção de Objetos

Cada componente de software é representado internamente por um ou mais objetos. Para cada ação executada sobre um componente, deve existir uma ação similar no SGBD. Desta forma, quando um componente é removido, um ou mais objetos devem ser removidos do SGBD. Assim, foi implementado no GOA++ um algoritmo de remoção de objetos. Este algoritmo, para não prejudicar o desempenho do GOA++, somente marca os objetos como removidos e possibilita que um coletor de lixo seja ativado em um momento oportuno.

Dentre os vários algoritmos para implementar um coletor de lixo (ex. Contador de Referências, Copying Collectors, etc.) foi escolhido o Mark & Sweep. O Mark & Sweep[5] consiste em uma busca (Mark) através das raízes de persistência, marcando recursivamente todos os objetos atingíveis. Em seguida, uma segunda busca (Sweep) é efetuada, passando por todos os objetos armazenados e verificando aqueles que não foram marcados na busca anterior. Caso um objeto não esteja marcado, significa que ele não pode ser atingido (objeto lixo), então, pode ser removido.

Algumas vantagens do algoritmo Mark & Sweep são: detectar objetos que não estão sendo referenciados, independentemente do grafo que estes formam (o algoritmo Contador de Referência não detecta sub-grafos desconexos cíclicos) e não necessitar de espaço em disco extra para a sua execução (o algoritmo Copying Collectors necessita do dobro de espaço em disco para a sua execução). Algumas modificações no algoritmo original foram implementadas para permitir que a parte Mark do algoritmo seja executada mais de uma vez sem que a parte Sweep seja executada. Desta forma, é possível medir a quantidade de objetos lixo na base e decidir se é ou não um momento adequado para eliminá-los. Este serviço pode ser visto na Figura 3a.

4. Gerência de Objetos em Memória

A primeira abordagem do GOA++ [6] para a gerência de objetos na memória baseou-se no emprego de apontadores (padrão C++) para referenciar diretamente os objetos persistentes carregados no cache. Quando um objeto era requisitado, um cópia do mesmo era carregada no cache, bem como de todos os objetos relacionados. Em seguida, um apontador passa a conter diretamente o endereço de memória do objeto no cache. Uma desvantagem dessa abordagem é que os objetos referenciados também são carregados, embora nem sempre utilizados pela aplicação. Conseqüentemente, muitas páginas de memória cache são alocadas desnecessariamente. No caso de aplicações que manipulam um grande volume de objetos, esse problema apresenta impacto considerável no desempenho.

A solução adotada para este problema foi a implementação de uma lista de objetos em memória, a qual contém informações sobre todos os objetos requisitados e referenciados no cache. Os apontadores de objetos passam a referenciar entradas desta lista. Cada entrada da lista de objetos é um descritor contendo: o OID, o endereço do objeto no cache, um contador de referências, se o objeto foi requisitado e se esse está sendo alterado ou não (*lock*). Quando o objeto é requisitado verifica-se através desta lista se o mesmo está contido em alguma página já carregada no cache, caso não esteja, a lista carrega a página com o objeto desejado. Este mecanismo permite que as páginas do cache possam ser liberadas sem comprometer o acesso aos objetos nelas armazenados.

Com esta nova abordagem, o antigo servidor de páginas do GOA++ torna-se um gerente de objetos que encapsula a requisição e liberação de objetos feita pela aplicação. As aplicações que utilizam o GOA++ possuem uma interface bem definida com este servidor de objetos requerendo e liberando objetos através de seus identificadores (a liberação pode também ser feita com um simples delete no ponteiro C++ para o objeto). O servidor de objetos por sua vez é quem faz uso da lista de objetos em memória, ao cache e ao disco (Figura 2).

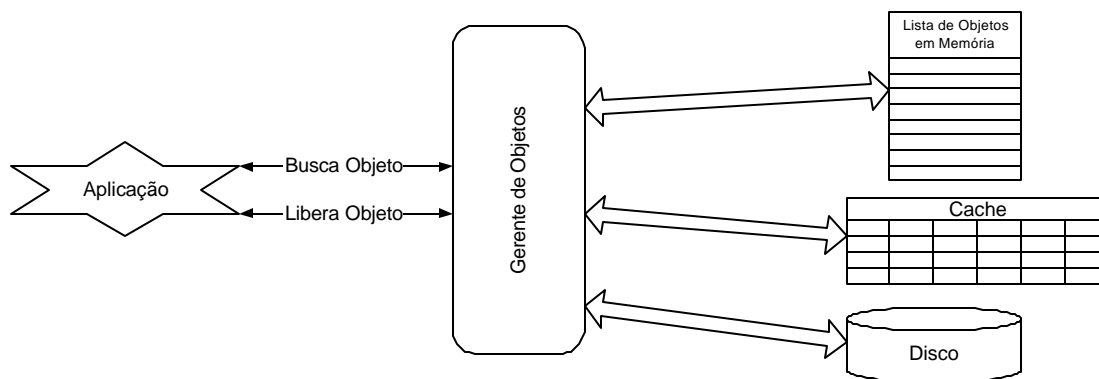


Figura 2 -Implementação da gerência de objetos em memória.

Uma requisição de um objeto recebida pelo Gerente de Objetos passa inicialmente por uma busca do objeto na lista de objetos em memória. Uma vez encontrado, o objeto é retornado e seu contador de referências é incrementado e, em caso contrário, procura-se o objeto no cache e conseqüentemente no disco, caso não encontrado. O objeto é então inserido na lista de objetos em memória e retornado ao requerente. Uma liberação de um objeto passa também inicialmente por uma busca na lista de objetos em memória (note que o objeto deve necessariamente estar presente na lista), o contador de referências do mesmo é decrementado e se esse valor atingir zero, o objeto, caso tenha sido modificado em seu percurso pela memória, é então escrito no cache para posterior escrita no disco.

5. Uso do GOA++ pela infra-estrutura de Reuso Odyssey

O GOA++ possui diversos papéis no suporte a infra-estrutura Odyssey (Figura 3b). A principal função do GOA++ é servir como repositório dos componentes criados no processo de engenharia de domínio. Além disso, o GOA++ é utilizado como um repositório de metadados dos mediadores, tradutores e fontes de dados distribuídas com integração na arquitetura de mediação, atuando também como um repositório dos modelos ontológicos e da base de conhecimento utilizada pelo Sistema Multi-Agente para o apoio ao usuário (agentes de interface adaptativa e de navegação inteligente).

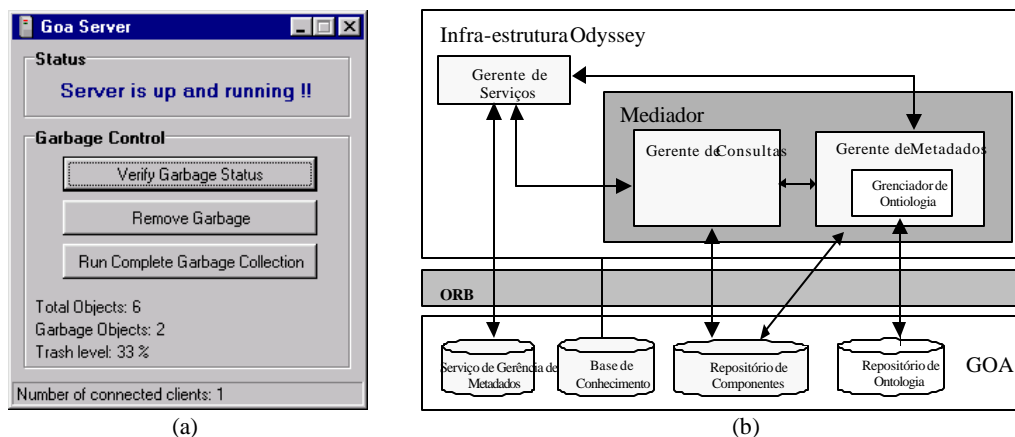


Figura 3 – (a) Janela do Servidor; (b) Suporte Goa ao ambiente Odyssey.

Outra característica importante oferecida pelo GOA++ ao Odyssey é a recuperação de componentes via OQL. A consulta a seguir recupera todos os casos de usos de um domínio que estejam relacionados a uma determinada *feature* do mesmo domínio: **select f in UseCases from NoFeatures f where f.Nome = "FeatureLegis";**

Agradecimentos

Gostaríamos de agradecer ao Cnpq e especialmente a Renato Campos Mauro, grande incentivador do Projeto GOA++ e projetista principal das versões [4 e 6]. Estendemos o agradecimento a todos os alunos da Linha de Banco de Dados da COPPE que vêm dedicando um grande esforço na consolidação e nas diversas extensões do GOA.

Referências

- [1] Braga, R.M.M., Werner, C.M.L. Mattoso, M.L.Q. "Odyssey: A Reuse Environment based on Domain Models" *Anais da IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99)*, IEEE CS Press, Richardson, Texas, março 24-27, 1999, pp.50-57.
- [2] Baião, F.A. Mattoso, M.L.Q. Zaverucha, G. "Horizontal Fragmentation In ODMs: New Issues and Performance Evaluation" *Proc. 19th IEEE International Performance, Computing, and Communications Conference -- IPCCC 2000*, IEEE CS Press, Phoenix, Arizona, fev 2000, pp.108-116.
- [3] Ogasawara, E. S. Mattoso, M.L.Q. "Uma Avaliação Experimental sobre Técnicas de Indexação em Bancos de Dados Orientados a Objetos", *XIV Simp Bras de Banco de Dados*, SBC, Florianópolis, out, 1999, pp.285-298.
- [4] Mauro, R.C. Mattoso, M.L.Q. et al. "GOA++: Tecnologia, implementação e extensões aos serviços de gerência de objetos", *Anais do XII Simpósio Brasileiro de Banco de Dados*, Fortaleza, outubro, 1997, pp.272-286.
- [5] Wilson, P.R.; "Uniprocessor Garbage Collection Techniques"; *International Workshop on Memory Management*; St. Malo; França; Setembro; 1992.
- [6] Mauro, R.C. Mattoso, M.L.Q. "GOA++ e suas Ferramentas", *Anais da 1ª Mostra Brasileira de Software Acadêmico e Comercial do XIII Simp Bras de Banco de Dados*, SBC, Maringá, outubro, 1998, pp.83-88.
- [7] Cattel, R.G., Barry, D.K., 1997. *The Object Database Standard: ODMG 2.0* - Morgan Kaufmann Publishers.