MVC: Um Modelo para Controle de Versões e Configurações*

Eliane Zambon Victorelli Dias † Geovane Cayres Magalhães ‡

Sumário

Este artigo apresenta um modelo para controle de versões e configurações em ambientes de desenvolvimento de software, o **MVC**. A modelagem tem três aspectos principais: a adaptação a diferentes ambientes, a distribuição e o compartilhamento dos dados. A adaptação a ambientes diversos é feita por meio dos parâmetros do modelo. Uma hierarquia de banco de dados possibilita a distribuição das informações, enquanto que o compartilhamento é controlado pelas partições de cada banco de dados e por um mecanismo de *check-out/check-in*.

1 Introdução

A necessidade de suportar adequadamente a evolução de um sistema de software, bem como de cada um de seus componentes separadamente, tem resultado no aparecimento de várias propostas para mecanismos de controle de versões e configurações.

Inicialmente, os mecanismos eram dirigidos ao desenvolvimento de software não distribuído [Ditt88, Wink87] ou destinados a um ambiente específico [Thom88]. No entanto, a tecnologia existente atualmente na área de desenvolvimento de software exige que o gerenciamento de versões e configurações suporte ambientes distribuídos [Chou86, Belk87, Cohe88, Vict89] e que as informações possam ser compartilhadas por diversos usuários [Beck90]. Além disso, o controle deve poder se adaptar às características de vários ambientes [Klah86, Schw89, Katz87].

Este artigo descreve o **MVC**: um modelo que propõe soluções consistentes para os problemas envolvidos no suporte a versões e configurações em ambientes distribuídos. Alguns dos problemas são resolvidos a partir de soluções existentes em outros modelos enquanto que para outros problemas novas soluções são propostas. São detalhados também os parâmetros usados pelo **MVC** para captar as características de cada ambiente.

2 Distribuição dos Dados

Em geral, os usuários de um ambiente de desenvolvimento de software são divididos em diferentes grupos de acordo com o projeto que desenvolvem. Algumas informações são compartilhadas por todos eles, outras são restritas aos componentes de um mesmo grupo, e existem também as informações particulares de cada usuário. Uma hierarquia de banco de dados, semelhante à proposta em [Chou86], promove uma distribuição adequada das informações.

Nesta hierarquia as informações são organizadas de acordo com os usuários que têm acesso a elas. Para tanto, são necessários três tipos diferentes de bancos de dados: os particulares pertencentes aos usuários comuns, os de projeto que pertencem aos grupos, e o banco de dados geral que armazena as informações compartilhadas por todos usuários do ambiente. O representante de um grupo é o dono do banco de dados do projeto associado, enquanto o representante do ambiente é o dono do banco de dados geral.

*Este trabalho faz parte de uma dissertação de mestrado parcialmente financiada pela Capes, FAPESP e CNPq

†Bolsista CNPq, Mestre em Ciência da Computação pela UNICAMP

‡Prof. do Depto. de Ciência da Computação da UNICAMP e Pesquisador do CPqD/TELEBRAS, Doutor em Ciência da Computação pela Universidade de Toronto

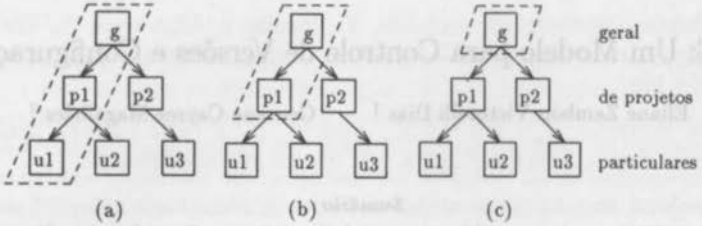

Figura 1: Bancos de dados acessíveis ao: (a) usuário u1, (b) responsável pelo projeto p1 e (c) responsável pelo ambiente.

Cada usuário tem acesso ao seu próprio banco de dados e aos que estejam nos níveis superiores ao seu na hierarquia. Contudo, ele não pode fazer acessos a bancos de dados que estejam no mesmo nível ou em níveis inferiores ao seu, como mostra a Figura 1.

Os objetos, as versões e as configurações armazenadas em um banco de dados estão divididos em várias partições. São elas que definem quais operações podem ser realizadas pelos usuários que têm acesso aos bancos de dados. Assim, a localização de um objeto na hierarquia de bancos de dados estabelece os usuários que têm a possibilidade de acessá-los, mas o tipo de acesso permitido é definido por sua partição. O controle de acesso através das partições será detalhado na Seção 7.

3 Objetos, Versões e Configurações

A descrição de um sistema de software é composta hierarquicamente. Ou seja, o objeto que representa um sistema pode ser único ou pode ser descrito em termos de objetos componentes, que por sua vez também podem ser compostos ou primitivos.

Através dos relacionamentos de composição é possível modelar a estrutura de um sistema em um instante do tempo, porém não é suficiente representar apenas o estado atual de um sistema. Para representar a evolução dos sistemas são usadas as versões e configurações.

Um estado particular de um objeto é chamado de versão, enquanto que o termo objeto se refere ao conjunto de todos os estados (ou versões) da entidade. A informação é armazenada nas versões; o objeto é apenas a unidade que compreende as versões e o que elas têm em comum [Klah86].

Como existem várias versões do objeto composto, bem como dos objetos componentes, é possível compor o sistema de inúmeras maneiras. Uma configuração está diretamente relacionada à composição hierárquica. Ela associa uma versão de um objeto composto a versões específicas de seus componentes.

3.1 Os Dados Manipulados pelo MVC

A estrutura do software é definida no MVC através de referências genéricas. Cada versão faz referência a um conjunto de objetos usados por ela. A referência é feita genericamente a um objeto e não a uma versão específica dele. Desta forma, a versão que faz a referência pode ser associada à diferentes versões do objeto referenciado. Além disso, as versões de um mesmo objeto podem fazer referências a objetos diferentes. Enquanto a versão 1 do objeto X faz referência aos objetos Y e Z, a sua versão de número 2 pode fazer referências aos objetos Z e V, como é mostrado na Figura 2.

A versão que faz a referência é ligada às versões dos objetos referenciados através das configurações. A exemplo do **Adele** [Belk87] e do modelo apresentado em [Chou86], o **MVC** descentraliza as configurações associando-as diretamente às versões. Porém, de forma contrastante com estes gerenciadores, ele permite que cada versão tenha várias versões de configuração. Assim, a versão 1 de X pode ser ligada à versão 2 de Y e à 4 de Z por sua versão de configuração 1; enquanto que a versão de configuração 2 a associa à versão 3 de Y e a 3 de Z.

Cada versão de um objeto, ao ser criada, recebe um número que a distingüe das outras versões do mesmo objeto. Da mesma forma, cada configuração de uma versão também tem um número próprio. Assim, o nome de um objeto seguido de um número pode ser usado para identificar unicamente uma versão em todo o ambiente, enquanto que uma versão de configuração é identificada pelo nome do objeto e número da versão seguidos por seu próprio número.

Para facilitar o entendimento, neste texto as versões de configuração são tratadas como configurações enquanto que as versões de objetos são chamadas simplesmente de versões. E quando o assunto aborda indistintamente objetos, versões e configurações, eles são chamados de elementos.

Em uma configuração, a versão escolhida para o objeto referenciado pode, por sua vez, fazer referência a outros objetos. Portanto, ela também deve ter uma configuração para determinar as versões dos objetos por ela referenciados. Desta forma, uma configuração do sistema é composta por versões e por subconfigurações, que também podem ser compostas, formando uma hierarquia de configurações. No exemplo da Figura 2, a versão X[1] faz referência aos objetos Y e Z, sendo que a configuração X[1][1] determina que as versões Y[2] e Z[4] sejam usadas na construção de X. A versão Z[4] não faz referências a outros objetos, mas o objeto W é referenciado por Y[2]. A configuração Y[2][1] determina que W[1] seja usada, ao passo que a versão W[3] é a especificada na configuração Y[2][2]. Portanto, a configuração X[1][1] pode ser composta de maneiras diferentes.

O uso de configurações diferentes possibilita que cada usuário componha o sistema com as versões adequadas ao seu trabalho. Ele pode especificar a composição de uma configuração através de suas entradas, que serão detalhadas na Seção 6.1.

Além das configurações específicas de cada versão, existem as configurações globais. Cada banco de dados pode ter várias versões da configuração global. A configuração global é usada sempre que uma configuração de uma versão não especificar versões para todos os objetos referenciados por ela. No exemplo anterior, se a configuração X[1][1] só informasse que a versão 1 de X deveria ser ligada a Y[2] e não tivesse nenhum dado sobre qual versão de Z deveria ser usada, esta informação seria procurada na configuração global.

Os objetos, versões e configurações de cada ambiente, podem ter diferentes propriedades representadas por seus atributos. Alguns atributos básicos, como a identificação, autor e datas de criação e de atualização são mantidos automaticamente pelo **MVC**. No entanto, novos atributos podem ser definidos para os elementos de um ambiente em particular.

3.2 Estruturas de Derivação

Uma versão de um objeto pode ser criada a partir da derivação de outra versão do mesmo objeto. O processo de derivação das versões de cada objeto é documentado em um histórico. O **MVC** permite que os históricos de um ambiente sejam representados por estruturas na forma linear ou de árvore, e que o usuário navegue através desta estrutura por meio de operações apropriadas. A forma do histórico de versões é definida por um dos parâmetros que caracterizam o ambiente.

As versões de um objeto são organizadas pela sua estrutura de maneira global. Não existe uma estrutura para cada banco de dados, e sim uma única estrutura que organiza todas versões existentes de um objeto, em todos os bancos de dados.

De forma análoga, as configurações de cada versão também podem ser derivadas umas das outras, e portanto são organizadas por uma estrutura. A Figura 3 mostra o objeto X com suas versões, configurações e as respectivas estruturas de derivação.

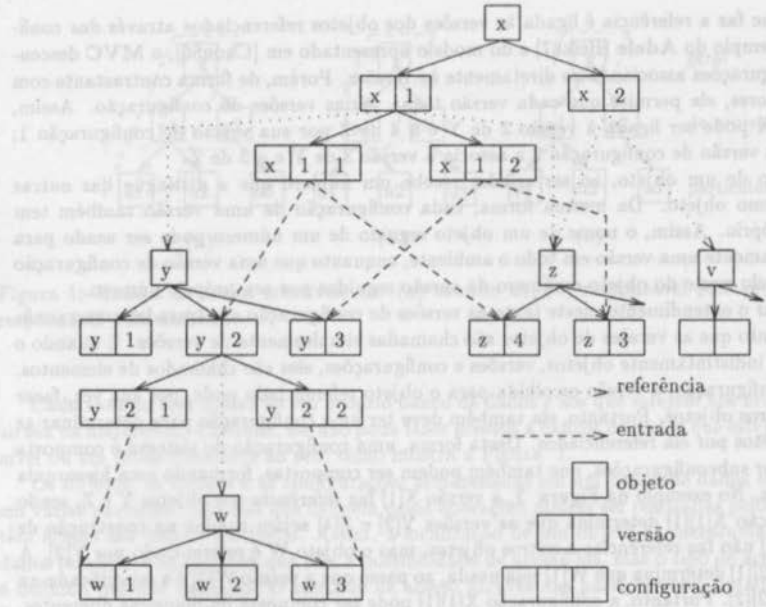

Figura 2: Diferentes configurações da mesma versão

Uma configuração global também pode ser derivada de outra configuração. Existe portanto, em cada banco de dados uma estrutura que organiza todas as configurações globais.

3.3 Inserção dos Dados

Cada usuário pode criar novos objetos, versões, configurações e configurações globais em seu banco de dados. A criação de um objeto inicializa a estrutura para organização de suas versões. Portanto, a inserção de uma versão só pode se dar depois que o respectivo objeto tenha sido inserido.

Como já foi mencionado na Seção 3.2, o processo de criação de versões define a estrutura de derivação do objeto. Quando uma versão é derivada de outra, ela é inserida na estrutura como sucessora da versão que lhe deu origem e, a princípio, ela se apresenta como uma cópia da sua antecessora e faz referência aos mesmos objetos que a versão antecessora.

As configurações são tratadas de forma semelhante às versões. Assim, a criação de uma configuração só pode ser efetuada depois que a respectiva versão tenha sido criada. E se a configuração é derivada de outra, ela é inserida na estrutura que organiza as configurações como derivada da que lhe deu origem e tem as mesmas entradas que ela.

Deve-se observar que quem tem acesso a uma versão, deve ter acesso também ao objeto correspondente. Portanto, na criação de uma versão o objeto deve estar armazenado no mesmo banco de dados que a versão ou em um de nível superior a este na hierarquia. Na derivação de uma versão, além do acesso ao objeto é necessário ter acesso à versão que vai lhe dar origem e conseqüentemente ela também deve estar armazenada em um destes bancos de dados. Analogamente, na criação de uma configuração, a versão e a configuração antecessoras têm que estar

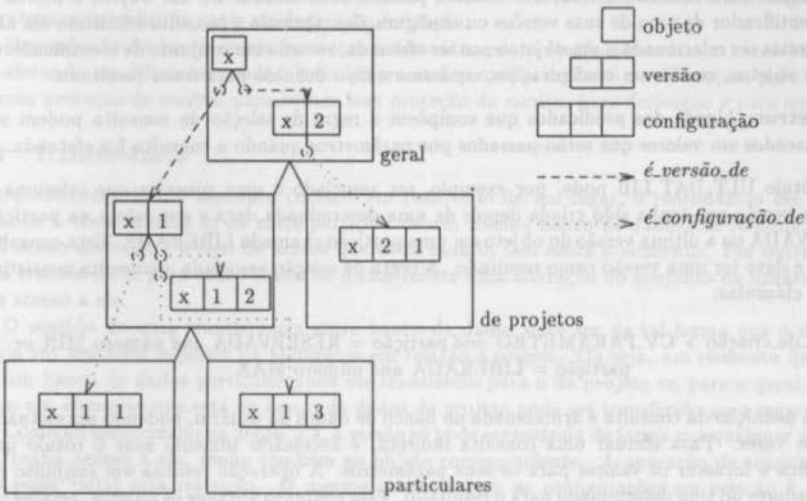

Figura 3: Um objeto com suas versões e configurações espalhadas pela hierarquia de banco de dados

acessíveis ao criador. A Figura 3 mostra uma possível distribuição das versões e configurações do objeto X.

Além disso, os objetos referenciados por uma versão e as versões que compõem uma configuração devem estar acessíveis a todos usuários que tenham acesso à versão e à configuração, respectivamente. Desta forma, as operações de inserção de referência e de entradas de configuração só têm sucesso quando estas condições são respeitadas.

4 Seleção

O **MVC** suporta regras de seleção que permitem recuperar objetos, versões e configurações através dos valores de seus atributos. Uma regra de seleção é constituida por uma seqüência de cláusulas, representando expressões lógicas do tipo *or*. Cada cláusula é composta por uma seqüência de predicados que consistem em expressões lógicas do tipo *and*. Um elemento é selecionado pela regra se ele satisfizer todos os predicados de uma de suas cláusulas.

Um predicado tem a forma <atributo><operador><valor>, onde o <operador> é representado por =, >, <, >=, <= , <>, MAX ou MIN. Os operadores MAX e MIN selecionam os elementos que contenham o valor máximo ou mínimo encontrado para o atributo. Portanto, os predicados baseados nestes operadores dispensam a parte que determina o valor.

Além das cláusulas e predicados que constituem a regra de seleção, a especificação de uma consulta deve conter:

Rótulo : A identificação da consulta é feita por um rótulo.

Tipo do resultado: A consulta pode retornar identificadores de objetos, de versões ou de configurações.

Tipo da consulta: Uma consulta é direta quando é relacionada a um objeto específico. Para efetuar uma consulta direta, um usuário passa o identificador de um objeto e recebe o identificador de uma de suas versões ou configurações. Quando a consulta é indireta ela não precisa ser relacionada a um objeto e, ao ser efetuada, retorna um conjunto de identificadores de objetos, versões ou configurações, conforme o tipo definido para o seu resultado.

Parâmetros: Alguns dos predicados que compõem a regra de seleção da consulta podem ser baseados em valores que serão passados por parâmetros quando a consulta for efetuada.

O rótulo ULT_DAT_LIB pode, por exemplo, ser associado a uma consulta que seleciona a primeira versão que tenha sido criada depois de uma determinada data e que esteja na partição RESERVADA ou a última versão do objeto em uma partição chamada LIBERADA. Esta consulta é direta e deve ter uma versão como resultado. A regra de seleção associada a consulta consistirá de duas cláusulas:

data_de_criação > CV_PARAMETRO *and* partição = RESERVADA *and* número MIN *or*
partição = LIBERADA *and* número MAX

Uma definição da consulta é armazenada no banco de dados do usuário, podendo ser efetuada inúmeras vezes. Para efetuar uma consulta indireta, é necessário somente usar o rótulo que a identifica e fornecer os valores para os seus parâmetros. A operação retorna um conjunto de identificadores do tipo determinado para o resultado. Esse conjunto abrange os objetos, versões ou configurações que satisfazem uma das cláusulas definidas para a consulta. Se a consulta for direta, é preciso fornecer também o identificador de um objeto e um único identificador é retornado. Uma consulta previamente definida pode também ser usada em entradas de configuração, conforme será descrito na Seção 6.1.

No exemplo acima, o usuário poderia efetuar a consulta fornecendo o rótulo ULT_DAT_LIB, a data para a comparação e a identificação de um objeto. A data passada substituiria CV_PARAMETRO na regra de seleção. E então, a consulta procuraria entre as versões do objeto especificado, uma que satisfizesse a primeira cláusula. Caso não fosse encontrada nenhuma versão nestas condições, a segunda cláusula seria usada.

5 Operações entre Bancos de Dados

5.1 *Check-out* e *Check-in*

O controle de acessos concorrentes aos objetos, versões e configurações é feito através das operações de *check-out* e *check-in*.

Através da operação de *check-out* o usuário toma emprestado um elemento de outro banco de dados para o seu. A operação instala uma cópia do elemento no banco de dados do usuário, onde serão realizadas as futuras leituras, atualizações, resoluções e derivações. Além disso, a realização de um *check-out* em um elemento faz com que o original fique protegido, impedindo que outros usuários o acessem de forma conflitante.

Quando o usuário realiza um *check-in*, as atualizações que foram feitas na cópia são passadas para o original e a cópia é destruída. O *check-in* também faz com que a proteção do original seja liberada.

Algumas vezes, enquanto a cópia de um elemento é lida, é necessário que o original não seja atualizado por outros usuários. Outras vezes, a atualização do original não interfere no trabalho que está sendo realizado na cópia, e assim a proteção para escrita não é requisitada na operação de leitura. Por outro lado, sempre que uma atualização está sendo feita na cópia, é necessário proteger o original de forma a impedir atualizações simultâneas por outros usuários. Além disso, quando o usuário vai derivar uma versão, é necessário que nenhum outro esteja percorrendo a

estrutura de versões do objeto, pois os resultados podem ser diferentes, conforme a navegação na estrutura seja realizada antes ou depois da derivação.

Dependendo do que se deseja fazer com o elemento, e da proteção necessária, o *check-out* pode ser efetuado em diferentes modos: para leitura com proteção de escrita e derivação, para leitura só com proteção de escrita, para leitura sem proteção de escrita, para derivação e para escrita.

5.2 Transferência

A transferência de um elemento consiste em removê-lo de um lugar, e reproduzí-lo em outro. Quando a transferência se dá entre partições de um mesmo banco de dados, na verdade, o que está sendo alterado é o tipo de acesso que cada usuário tem sobre o elemento. Por outro lado, uma transferência para outro banco de dados reflete uma alteração do conjunto de usuários que têm acesso a ele.

O sentido de uma transferência entre banco de dados deve ser de tal forma que o destino esteja em um nível superior da hierarquia em relação a origem. Ou seja, um elemento que está em um banco de dados particular pode ser transferido para o de projeto ou para o geral, assim como um elemento que está no banco de dados de projeto pode ser transferido para o geral.

Como foi mencionado na Seção 3.3, a versão só pode ser inserida de forma que qualquer usuário que tenha acesso a ela, tenha também ao objeto correspondente. A operação de transferência não pode violar esta restrição. O mesmo acontece com as configurações em relação à versão correspondente. Além disso, uma versão só pode ser transferida se não fizer referência a outros objetos que estejam armazenados em bancos de dados de níveis inferiores ao seu destino. O mesmo acontece na transferência de uma configuração em relação às versões e configurações referenciadas por suas entradas.

É importante notar que, ao contrário da configurações comuns, as globais não podem ser transferidas para outros bancos de dados. Uma vez criadas, elas devem permanecer no mesmo banco de dados até serem removidas

6 Configuração

6.1 Entradas de Configuração

O trabalho de um usuário gera uma necessidade em relação à composição do sistema, que normalmente é diferente das exigências dos outros usuários. Como o MVC suporta diferentes configurações para a mesma versão, cada usuário pode criar novas configurações do sistema que utilizem as versões que ele deseja.

No entanto, especificar uma a uma as versões que devem compor o sistema, é muito trabalhoso. Além disso, quando for usada uma versão que faz referências a outros objetos, é necessário também especificar versões para todos os objetos referenciados. Assim, uma especificação por extenso está sujeita a muitos erros. Por outro lado, compor o sistema somente através de valores *default* também não é suficiente.

Conforme foi ilustrado pela Figura 2, as referências feitas por uma versão determinam os objetos usados por ela, ao passo que as versões dos objetos referenciados que compõem o sistema são determinadas através das entradas de configuração. Para facilitar o trabalho do usuário, o MVC oferece entradas de configuração de diversos tipos. Elas permitem que ele determine facilmente a composição desejada para o sistema.

A Figura 2 ilustra entradas que ligam uma configuração a uma versão de outro objeto. Mas dependendo do seu tipo, as entradas podem determinar a composição da configuração de várias outras maneiras. Os tipos de entradas suportados pelo MVC são:

Entrada direta: A referência é feita a um objeto específico. Ela tem uma das seguintes formas:

total: É determinada uma subconfiguração de um dos objetos referenciados para fazer parte da configuração. Na configuração X[1][1] da Figura 2, o usuário poderia inserir uma entrada direta total para a subconfiguração Y[2][2]. Desta forma, ele garantiria que a versão 2 de Y fosse usada, e que ela fosse composta por W[3].

parcial: Uma versão de um dos objetos referenciados é escolhida para fazer parte da configuração. Este tipo de entrada pode ser usado tanto para as versões que não fazem referência a outros objetos, como para as que fazem. No último caso, o gerenciador determina a subconfiguração a ser usada através de critérios que serão apresentados na Secção 6.3. No exemplo descrito acima, Y[2] e Z[4] são entradas diretas parciais da configuração X[1][1].

de consulta: Uma consulta direta é relacionada à configuração. Ela deve ser efetuada, resultando em uma versão ou em uma subconfiguração para fazer parte da composição do sistema. O usuário poderia determinar que a última versão liberada do objeto Z seja usada na configuração X[1][1], através da entrada Z[ULT_LIB]. Para inserir a entrada seria necessário que o rótulo ULT_LIB identificasse uma consulta definida adequadamente.

Entrada indireta: É determinado um conjunto de versões ou de subconfigurações que podem ser usadas, se na mesma configuração não houver entrada direta para algum dos objetos referenciados. As formas em que podem ser usadas as entradas indiretas são:

de inclusão: Associa a configuração a uma outra configuração, cujas entradas podem ser usadas. Em X[1][2] poderia existir uma entrada incluindo a configuração X[1][1]. E então, se as entradas diretas de X[1][2] não fossem suficientes para determinar versões para os objetos Y e Z, as entradas da configuração X[1][1] seriam usadas.

de consulta: Uma consulta indireta determina um conjunto de versões ou subconfigurações, que podem ser usados. Uma configuração pode ter uma entrada indireta, como por exemplo [ULT_AUTOR][José]. Supondo que o rótulo ULT_AUTOR, esteja associado a uma consulta que seleciona todas as últimas versões criadas por um determinado autor, a configuração usaria a última versão criada por José, para os objetos referenciados que não tenham entradas diretas correspondentes.

Entrada *default*: Um número de versão, um número de configuração ou uma consulta direta é especificado. Caso as entradas diretas e indiretas não sejam suficientes para determinar as versões para todos os objetos referenciados, a entrada *default* é usada para cada um deles.

6.2 Ativação

Em determinadas situações, a versão ou configuração ativa é escolhida para fazer parte da composição do sistema, conforme será descrito na Seção 6.3.

Quando um usuário vai trabalhar com uma versão ele a traz para o seu banco de dados através do mecanismo de *check-out/check-in*. Para o usuário que efetuou a operação, aquela versão se torna ativa em relação ao objeto. A versão permanece ativa para o usuário até que ele faça um *check-in* naquela versão ou um *check-out* em outra versão do mesmo objeto.

As configurações comuns e as globais também são ativadas através da operacões de *check-out* e *check-in*. No entanto, as configuracões têm relacionamentos de ativação tanto com o objeto quanto com a versão correspondente.

6.3 Resolução de Configuração

Para se usar um sistema é necessário determinar exatamente uma versão de cada objeto que o compõe, portanto é preciso resolver a sua configuração. A resolução de uma configuração

percorre suas entradas determinando uma versão para cada objeto referenciado, e quando a versão escolhida faz referência a outros objetos, é determinada também uma configuração para esta versão, que será resolvida recursivamente.

Quando todas as entradas de uma configuração já tiverem sido resolvidas, e ainda não tiver sido determinada uma versão para cada objeto referenciado, as entradas da configuração global ativa são usadas.

A utilização das versões e configurações ativas em uma resolução pode ser feita com diferentes prioridades. Se a prioridade for 0, antes de percorrer as entradas diretas, o mecanismo procura por versões ou configurações ativas dos objetos referenciados. Se elas forem encontradas, serão usadas. Posteriormente, as entradas da configuração em resolução são percorridas para determinar versões para os objetos que não tenham versões ou configurações ativas. Na resolução com prioridade 1, as entradas diretas são resolvidas e só depois são consideradas as versões e configurações ativas. Elas são procuradas depois das entradas indiretas se a prioridade for 2, e depois da entrada *default* caso a prioridade seja 3. Quando a prioridade é 4, as versões e configurações ativas só são usadas se depois de resolvidas todas as entradas da configuração global, as versões dos objetos referenciados não estiverem todas determinadas. Finalmente, elas nunca são consideradas se a prioridade for 5. Independendemente da prioridade usada, as versões ativas são procuradas antes das configurações ativas.

Para permitir que o usuário possa trabalhar com uma configuração que foi resolvida, o gerenciador realiza um *check-out* em cada versão ou configuração envolvida na resolução, instalando uma cópia no banco de dados do usuário com a proteção especificada por ele.

Através da operação de resolução, é criado no banco de dados do usuário um objeto chamado *configuração_resolvida*, que contém exclusivamente entradas diretas. Em outras palavras, uma *configuração_resolvida* é composta por versões que não fazem referências e por outras *configurações_resolvidas*. Por meio das entradas da *configuração_resolvida* o usuário conhece a composição do sistema, e tem acessó a todas as versões e configurações que sofreram *check-out* devido à resolução.

6.4 Um Exemplo de Configuração

Um bom exemplo do processo de resolução é o da configuração Z[2][2] mostrada na Tabela 1, pelo usuário identificado por id1. Nesta resolução é usada a prioridade 1 para versões e configurações ativas. Na Tabela 2 são mostrados os resultados obtidos para as consultas efetuadas durante a resolução e na Tabela 3 está a configuração global ativa, enquanto que as versões e configurações ativas para o usuário id1 estão na Tabela 4. Neste exemplo, o rótulo ULT_LIB e ULT_AUTOR estão definidas da mesma forma que nos exemplos anteriores.

A versão 2 de Z faz referência aos objetos A, B e C, e a configuração Z[2][2] tem entradas diretas para A e C.

A primeira entrada determina que a configuração C[1][4] deve ser usada. A versão C[1] faz referência a D e portanto, deve-se resolver sua configuração. Em C[1][4] existe uma entrada direta D[ULT_LIB]. Como pode ser visto na Tabela 3 a consulta D[ULT_LIB] resulta na versão 5. Esta versão faz referência aos objetos G, H e I; o que implica na necessidade de resolver a sua configuração. Como a entrada que resultou em D[5] não especificou a configuração, a configuração ativa em relação a versão D[5] será usada, que segundo a Tabela 4 é D[5][2].

A configuração D[5][2] não tem entradas diretas, e não existem versões ou configurações ativas dos objetos G, H e I. Esta configuração consta de uma entrada indireta [ULT_AUTOR][José] e no conjunto das últimas versões de José, encontra-se a versão H[3]. Como falta determinar uma versão para G e I, a configuração global é consultada. Nela existe uma entrada direta que determina a versão G[9] como componente do sistema. A configuração global tem também a entrada *default* [ULT_LIB] que, aplicada ao objeto I, resulta na versão 6. Como G[9], H[3] e I[6] não fazem referências a outros objetos, não é preciso resolver as suas configurações.

Versão	Ref.	Configuração		
		Núm.	Tipo	Entrada
Z[1]	A, B	2	Dir.	A[4]
			Def.	[ULT_LIB]
Z[2]	A, B, C	1	Dir.	C[1][2]
			Ind.	[1][2]
		2	Dir.	C[1][4]
				A[4][5]
			Def.	ULT_LIB]
A[4]	E	5(R)	Dir.	E[3][2]
B[9]	-	-	-	-
C[1]	D	2	Dir.	D[5]
		4	Dir.	D[ULT_LIB]
D[5]	G, H, I	1	Def.	[ULT_LIB]
		2	Ind.	[ULT_AUTOR][José]
E[3]	F	2(R)	Dir.	F[5]
F[5]	-	-	-	-
G[9]	-	-	-	-
H[3]	-	-	-	-
I[6]	-	-		-

Tabela 1: Exemplo de algumas configurações, versões e objetos.
Obs: O número da configuração seguido por um R indica que ela está no estado *resolvida*

Continuando a resolução de Z[2][2], encontra-se a entrada direta para a configuração A[4][5]. Esta configuração está no estado *resolvida* e portanto as suas entradas consistem de versões que não fazem referências a outros objetos ou em outras configurações no mesmo estado. Através de sua entrada direta encontra-se a configuração E[3][2]. A configuração E[3][2], por sua vez, tem uma entrada direta para a versão F[5], que não faz referências.

Em Z[2][2] não existe entrada direta para B, mas existe uma versão ativa do objeto B para o usuário id1. Assim, a versão usada será B[9], que não faz referência a outros objetos.

Conclui-se dessa maneira o processo. A Tabela 5 mostra qual seria a *configuração_resolvida* encontrada no banco de dados do usuário id1, após a operação. Observe que a notação [rel n] no lugar do número da configuração indica que a configuração é o resultado da resolução da configuração n.

6.5 Transformação da *Configuração_resolvida*

Quando o usuário resolve uma configuração, não é gerada uma nova versão de configuração. Assim, se em seguida ele resolver outra vez a mesma configuração, a *configuração_resolvida* que estava no seu banco de dados é substituida pelo resultado da operação.

Se o usuário fizer a resolução com proteção para escrita, ele pode depois transformar a *configuração_resolvida* em uma configuração comum. A partir de então a nova configuração passa a ser acessível a outros usuários. Ela é disposta na estrutura de configurações como sucessora da configuração que foi resolvida inicialmente. O mesmo se dá com todas as suas subconfigurações. Se o usuário não julgar necessário guardar a versão da *configuração_resolvida*, ele deve liberá-la, tirando desta forma a proteção de todas versões e configurações que estavam protegidas devido à

Consulta	Resultado
B[ULT_LIB]	B[9]
D[ULT_LIB]	D[5]
I[ULT_LIB]	I[6]
[ULT_AUTOR][José]	A[6]
	B[4]
	C[3]
	H[3]

Tabela 2: Resultados das consultas efetuadas

Tipo entrada	Entrada
Dir.	A[3]
	B[5]
	G[9]
Def.	[ULT_LIB]

Tabela 3: Composição da configuração global ativa para o usuário id1

Objeto	Versão Ativa
D	D[3]
B	B[9]

Objeto	Configuração Ativa
D	D[3][2]

Versão	Configuração Ativa para a Versão
D[4]	D[4][3]
D[5]	D[5][2]

Tabela 4: Versões e configurações ativas para o usuário id1

Versão	Configuração	
	Número	Entrada
Z[2]	[rel 2]	A[4][rel 5]
		B[9]
		C[1][rel 4]
A[4]	[rel 5]	E[3][rel 2]
B[9]	-	-
C[1]	[rel 4]	D[5][rel 2]
D[5]	[rel 2]	G[9]
		H[3]
		I[6]
E[3]	[rel 2]	F[5]
F[5]	-	-
G[9]	-	-
H[3]	-	-
I[6]	-	-

Tabela 5: *Configuração_resolvida*

resolução.

Cada configuração tem um atributo que mantém o seu estado. Uma configuração está no estado *resolvida*, como A[4][5] do exemplo anterior, quando é criada a partir da transformação de uma *configuração_resolvida*, ou é derivada de outra configuração neste estado. Ela conserva este valor para o atributo, até que suas entradas ou as referências da versão à qual ela é relacionada sejam atualizadas. Então, ela passa para o estado *não_resolvida*.

Através deste atributo, na transformação de uma *configuração_resolvida* em uma configuração comum é possível manter o controle das configurações já resolvidas, evitando a inserção de configurações identicas às que lhes deram origem. Além disso, para o usuário transformar uma *configuração_resolvida*, ele deve ter permissão para derivação da configuração original. Não haverá problemas, se ele usar uma configuração que se mantém totalmente *resolvida*, para a qual ele não tenha permissão para derivação. Neste caso ele não vai criar uma nova configuração, só vai usar a existente em uma entrada de outra configuração, o que sempre é permitido quando se tem acesso à configuração.

7 Controle de Acesso

Conforme já foi mencionado, a distribuição dos dados apresentada na Seção 2 estabelece os usuários que têm acesso a um elemento, mas não o tipo de acesso. Através dos parâmetros que caracterizam o ambiente, o MVC divide os elementos de cada banco de dados em várias partições, e cada partição define o tipo de acesso permitido a cada usuário. Ou seja, através das partições são dadas permissões aos usuários de maneira a preservar as capacidades dos elementos que ela armazena.

As operações controladas em cada partição são: inserção, eliminação, transferência e atualização de qualquer elemento e também a derivação de configurações e de versões. A operação de leitura é sempre permitida a todos que tenham acesso ao banco de dados.

Para derivar uma versão ou configuração, o usuário deve ter direito de derivação na partição

em que está a antecessora, e direito de inserção na que será colocada a nova versão ou configuração. A transferência envolve o direito de transferência da partição onde está o elemento, e o direito de inserção na partição para onde o elemento vai.

Normalmente, a definição das permissões se baseiam nos tipos de usuários que têm acesso ao banco de dados, ou seja, uma permissão é dada a todos os usuários particulares que têm acesso ao banco de dados ou aos representantes dos projetos e do ambiente. É possível também associar uma permissão a um atributo que identifique um único ou um grupo de usuários. Uma partição pode, por exemplo restringir a operação de atualização de uma versão somente ao seu criador, através do seu atributo *autor*.

Cada uma das partições de um banco de dados, e as respectivas operações, deve ser definida separadamente. O controle de acesso do ambiente poderia ser definido de tal forma que na partição *Em_teste* do banco de dados de projeto, só o representante do grupo poderia inserir novos objetos. A inserção de versões poderia estar restrita ao responsável pelo objeto, ao passo que a inserção de configuração poderia ser feita tanto pelo representante do grupo como pelos usuários particulares que tem acesso ao banco de dados. A atualização dos objetos seria permitida somente ao representante do grupo, e a atualização das versões e configurações seria um privilégio dos respectivos autores. Em um caso assim, as seguintes definições deveriam ser feitas:

```
particao Em_teste bd projeto

insercao
objeto (projeto);
versao (objeto.responsavel);
configuracao (projeto, particular);

atualizacao
objeto (projeto);
versao (versao.autor);
configuracao (configuracao.autor);
```

8 Considerações Finais

Algumas diferenças entre o MVC e o outros modelos podem ser ressaltadas.

As versões no MVC estão armazenadas em uma hierarquia de bancos de dados como no modelo descrito em [Chou86]. No entanto, os problemas encontrados no processo de configuração deste modelo foram resolvidos através do suporte a versões de configuração e do uso de estruturas de derivação e numeração únicas para todas as versões de um objeto.

Embora a sintaxe das regras de seleção propostas pelo MVC seja semelhante à do **shape** [Mahl88], ela se destina a recuperar um objeto ou um conjunto deles com determinadas características. No **shape** as regras são usadas como uma descrição de várias versões com características distintas, que formam uma configuração. No MVC isto não é necessário, pois versões diferentes são selecionadas por entradas de configuração diferentes. As contrário do **shape**, do **Adele** [Belk87] e do **Gypsy** [Cohe88], no MVC a definição dos predicados é feita uma única vez e pode ser usada sempre que necessário.

O MVC se enquadra entre os gerenciadores que controlam o acesso através dos domínios de cada usuário, como é o caso do **Adele**. O seu mecanismo de controle de acesso não é tão flexível quanto ao encontrdo no **Adele**, porém ele permite um maior controle sobre o estado de cada versão.

Finalmente, o maior contraste do MVC com os outros modelos é a sua capacidade de se adaptar às características de cada ambiente através de seus parâmetros. Os atributos definem as

propriedades dos elementos de cada ambiente; a forma das estruturas de versões pode restringir o mecanismo de derivação e as partições e as respectivas permissões refletem o controle de acesso usado no ambiente.

9 Agradecimentos

Agradecimentos são devidos ao projeto ETHOS pela concessão da utilização dos seu recursos de hardware e de software, que possibilitaram a realização deste trabalho.

Referências

[Belk87] Belkhatir, N. & Estublier, J., Experience with Data Base of Programs, Proceedings of the ACM SIGSOFT / SIGPLAN Software Engineering Symposium on Pratical Software Development Environments, Palo Alto, California, December 1986, p. 84 - 91

[Beck90] Becker, K. & Golendziner, L. G., Tratamento de Versões em um Ambiente de PAC para Sistemas Digitais, Anais do X Congresso da SBC, Victória, ES, Julho 1990, p. 424 - 439

[Chou86] Chou, H. T. & Kim, W., A Unifying Framework for Version Control in a CAD Environment, Proceedings of the 12th VLDB Conference, Kyoto, August 1986, p. 336 - 344

[Cohe88] Cohen, E. S., et. al., Version Management in Gypsy, ACM, 1988, p. 201 - 215

[Ditt88] Dittrich, K. R. & Lorie, R. A., Version Support for Engineering Database Systems, IEEE Transactions on Software Engineering, vol. 14, n. 4, April 1988, p. 429 - 436

[Katz87] Katz, R. H., et. al., Design Version Management, IEEE Design & Test, February 1987, p. 13 - 22

[Klah86] Klahold, P., Schlageter, G. & Wilkes, W., A General Model for Version Management in Databases, Proceedings of the Twelfth International Conference on Very Large Data Bases, Kyoto, August 1986, p. 319 - 327

[Mahl88] Mahler, A. & Lampen, A., An Integrated Toolset for Engineering Software Configurations, ACM, 1988, p. 191 - 200

[Schw89] Schwanke, R. W., et. al., Configuration Management in BiiN SMS, ACM 1989, p. 383 - 393

[Thom88] Thomas, D. & Johnson, K., Orwell - A Configuration Management System for Team Programming, OOPSLA 88 Proceedings, September 1988, p. 135 - 141

[Vict89] Victorelli, E. Z., Magalhães, G. C. & Drummond, R., Mecanismo de Gerenciamento de Versões e Configurações do A_HAND, Anais do III Simpósio Brasileiro de Engenharia de Software, SBC, Recife, PE, Outubro 1989, p. 269 - 280, selecionado para publicação na Revista Brasileira de Computação, vol 5, n. 2, dezembro 1989, p. 3 - 9

[Wink87] Winkler, J. F. H., Version Control in Families of Large Programs, Proceedings of 9th International Conference on Software Engineering, Monterey, California, April 1987, p. 150 - 161