Mecanismo de gerenciamento de versões e configurações do A_HAND

Eliane Zambon Victorelli* Geovane Cayres Magalhães+°
Rogério Drummond+

15 de agosto de 1989

Sumário

O ambiente A_HAND tem como objetivo dar suporte ao desenvolvimento de sistemas de software. Em um ambiente deste tipo, o controle de versões e configurações é uma função muito importante, pois um sistema de software sofre alterações constantemente.

Este artigo apresenta a definição do mecanismo de gerenciamento de versões e configurações do ambiente A_HAND. Este mecanismo é uma extensão ao modelo proposto em [Ditt88], para adaptação às características do ambiente. A extensão incorpora versões de interface e suporte a desenvolvimento em grupos de projeto.

1 Introdução

Nos últimos anos o avanço acelerado da tecnologia na área de hardware tem estimulado a pesquisa relacionada ao desenvolvimento de software. Para que sistemas de software grandes e complexos possam ser desenvolvidos, é necessário que exista um ambiente de desenvolvimento de software adequado. Este tipo de ambiente tem como objetivo proporcionar um conjunto integrado e coerente de ferramentas, para automatizar alguns aspectos não criativos das várias tarefas realizadas durante o ciclo de vida de um projeto de software. Espera-se que o seu uso cause um aumento substancial na produtividade e na qualidade do produto.

O controle de versões é uma das funções que deve ser suportada por um ambiente de desenvolvimente de software. Esse controle é útil, tanto na fase de desenvolvimento como na de manutenção, pois um sistema de software sofre alterações constantemente. Muitas versões de um sistema de software precisam ser experimentadas antes de se selecionar uma, que satisfaça os requisitos de

*Mestranda do DCC/UNICAMP; +Professor do DCC/UNICAMP; °Pesquisador do CPqD/TELEBRAS

projeto. Um sistema complexo consiste de componentes de níveis mais baixos, sendo que um componente pode ser compartilhado por várias pessoas, pode ter diferentes representações e pode também, consistir de componentes de níveis inferiores. Quando um componente é alterado, os componentes de níveis superiores que o referenciam podem se tornar inválidos e precisar de alterações [Bato85,Chou86,Kim87,Kim88].

Assim, muitas versões são geradas durante o desenvolvimento de um produto de software e, o projetista precisa de um sistema de controle e armazenamento para que possa manipular mais facilmente as versões desejadas.

Em um ambiente de desenvolvimento de software, muitos projetistas trabalham em conjunto, dividindo ou compartilhando as responsabilidades pelos diversos componentes do sistema de software. Para a formação de um sistema como um todo, versões de vários projetistas precisam ser integradas. Além disso, cada projetista quer "ver" o sistema à sua maneira, o que resulta na necessidade de diferentes configurações do mesmo. Portanto, um ambiente de desenvolvimento de software deve ter um mecanismo de gerenciamento de configurações, a fim de tornar o trabalho dos projestistas mais cômodo [Belk86,Cohe88,Ditt88,Katz86,Thom88].

Este artigo apresenta a definição do modelo de controle de versões e configurações do ambiente de desenvolvimento de software A_HAND, que está sendo desenvolvido no Departamento de Ciência da Computação da Unicamp. O modelo foi definido com base na ánalise de vários aspectos de um ambiente de programação. Este aspectos incluem:

- identificação e controle de versões,
- estabelecimento de configurações do sistema de software adequadas as necessidades de cada usuário, e
- mecanismos de cooperação entre integrantes de um mesmo grupo de projeto.

2 Conceitos Básicos

2.1 Versões, Configurações e Equivalências

Em um sistema de software os componentes, também chamados de objetos de projeto, estão organizados em três planos distintos e ortogonais entre si [Beck89,Katz86,Katz87]:

2.1.1 Plano de Versões

Ao longo de sua vida, um objeto de projeto é representado por diversas versões diferentes. O plano de versões as organiza em um histórico.

O objetivo do histórico de versões é documentar as diversas etapas ou fases envolvidas na concepção de um objeto de projeto, organizando versões através de relacionamentos de derivação ou sucessão. As versões podem ser organizadas

como uma seqüência linear, como uma árvore ou como um grafo. No caso de sequência, a derivação é baseada na data de criação e, portanto, uma nova versão é sempre disposta no histórico como derivada da última existente. Porém, um objeto de projeto pode ter mais de uma versão sendo trabalhada ao mesmo tempo, e mais de uma versão pode ser derivada da mesma versão antecessora. Um histórico de versões na forma de árvore permite organizar estes conceitos. Finalmente, um histórico de versões na forma de grafo permite representar que uma versão foi derivada a partir de duas ou mais versões antecessoras.

2.1.2 Plano de Configuração

Objetos de projeto podem ser tanto compostos quanto primitivos, conforme posssam ou não ser decompostos em outros objetos de projeto. Objetos compostos são construídos a partir de referências a outros, por sua vez compostos ou primitivos. As hierarquias de composição são grafos acíclicos direcionados, nos quais cada nó representa um objeto de projeto, e cada arco representa o uso de um objeto na definição do outro objeto a ele conectado. Uma configuração está intrinsicamente ligada à composição hierárquica, pois ela liga uma versão de um objeto composto a versões específicas de seus componentes. Cada objeto que participa de uma configuração tem suas versões dispostas dentro de seu próprio plano de versões.

Existem duas maneiras de especificar versões em configurações: estaticamente, definindo ligações entre versões específicas dos objetos, ou dinamicamente, estabelecendo as ligações somente entre os objetos de projeto, de forma que as versões a serem usadas serão determinadas no momento de execução. Ligações dinâmicas em configurações servem para que diversas composições possam ser analisadas sem que a versão composta seja alterada.

2.1.3 Plano de Equivalência

Um mesmo objeto de projeto pode ser descrito através de diferentes representações, as quais caracterizam diferentes perspectivas. Essas representações diferentes dão origem a hierarquias de composição paralelas, chamadas hierarquias de representação. Equivalências identificam correspodências entre versões em diferentes representações, enquanto que um plano de versões documenta a evolução de um objeto de projeto dentro de uma mesma representação.

2.2 Sistemas de Software

Um sistema de software pode ser visto como um grafo acíclico direcionado, onde os nós folhas são módulos básicos e os nós internos são subsistemas ou módulos compostos. Cada subsistema é realizado por uma composição de seus nós sucessores, os quais podem ser outros subsistemas ou módulos básicos.

Todo módulo pode fornecer e requerer recursos de outros módulos. Esta troca de recursos é que define a composição hierárquica de módulos de um

sistema de software. É útil pensar nos módulos como tendo interfaces através das quais eles interagem com os outros módulos. A interface de um módulo é portanto, uma descrição dos recursos requeridos e fornecidos por ele, e o corpo é a implementação dos recursos que ele pode fornecer.

Existem várias implementações possíveis dos recursos de uma interface. Cada uma destas implementações toma a forma de corpo do módulo e constitui uma versão [Belk86].

Por muito tempo se considerou que a interface do módulo era determinada precisamente pela sintaxe dos recursos que ela provê e requer, e que todas as implementações compartilhavam a mesma interface, mas isto não é adequado. Um vez que cada implementação representa uma maneira diferente de fazer a mesma coisa, é natural que diferentes implementações necessitem de recursos diferentes e que forneçam os recursos de maneiras também diferentes.

A noção de versão de interface leva em conta o fato que as interfaces mudam, embora menos freqüentemente que as implementações. Assim sendo, cada versão de interface de um módulo tem as mesmas funcionalidades lógicas, e todas elas são reunidas em um único objeto de projeto. Portanto, no contexto de sistemas de software, um objeto de projeto corresponde ao conjunto de todas as versões de um módulo que tenham a mesma especificação de propriedades funcionais [Belk86].

3 Características do Ambiente

A linguagem Cm - C modular e polimórfico - foi projetada para ser a linguagem básica do ambiente A_HAND. Em Cm os objetos básicos manipulados são classes. As classes contém todas as definições de estruturas e funções necessárias para a sua utilização, constituindo-se em unidades de software que podem ser desenvolvidas separadamente e depois integradas para formar um sistema complexo. Desta forma, uma classe é potencialmente um programa e pode importar um conjunto de outros módulos. Os módulos importados são chamados de subprogramas ou subsistemas e, por sua vez, podem importar outros módulos até se chegar a módulos básicos que não fazem importações [Silv88]. O último módulo a fazer importações é a raiz do grafo que representa o sistema.

O conceito de herança múltipla de tipos também é suportado pela linguagem. Uma classe pode herdar várias características de outras classes, e ainda passar suas características e as que foram herdadas para seus próprios herdeiros. Como em Cm classes definem construtores de tipos, as relações de herança dão origem a uma hierarquia de tipos [Silv88].

Assim, um sistema de software em Cm consiste de um grafo acíclico, definido pelas relações de importação e herança.

Os programas são representados internamente através de sua árvore sintática acrescida de atributos. O programa apresentado na tela para o usuário será uma decompilação desta estrutura interna. Será ainda, mantida uma estrutura

paralela contendo informações sobre os símbolos do programa, a qual é uma hierarquia isomorfa à hierarquia dos módulos, sendo que para cada módulo haverá uma tabela de símbolos locais, módulos importados e símbolos exportados [Drum87a,Drum87b].

Todas as funções oferecidas pelo ambiente de programação operam na mesma estrutura de dados, isto é, na representação interna do programa e nos seus atributos associados.

4 Controle de Versões

Quando se lida com versões, três tipos diferentes de informações estão envolvidas:

- a informação de projeto, suprida pelo projetista ou por seu software;
- informação adicional a respeito dos objetos de projeto, isto é, pelo menos algum tipo de identificação;
- e informação representando o relacionamento entre objetos de projeto e versões.

Se não existisse suporte para versões, o usuário teria que se responsabilizar pela manutenção explícita das informações dos dois últimos tipos [Ditt88].

4.1 Numeração

Uma vez que o ambinete de programação terá controle sobre a sintaxe do Cm e sobre a estrutura dos programas, o controle de versões poderá ser implementado de modo a explorar as características dos módulos. Ou seja, o número da versão de um módulo consistirá de dois campos (E.I), atualizados automaticamente pelo ambiente. O campo I (versão de implementação) é acrescido de 1 quando a versão sofre alterações somente nas informações internas ao módulo, mantendo inalterada a interface do módulo em relação a versão anterior. Sempre que uma versão sofre alterações na interface do módulo (parâmetros formais do módulo ou de suas funções exportáveis) o campo I é zerado e o campo E (versão de interface) é acrescido de 1.

Este tipo de controle é necessário para evitar recompilações desnecessárias. Um módulo M alterado somente internamente (isto é, E constante) precisa ser recompilado mas não os módulos que usam M [Drum87b].

Os números das edições terão um campo adicional ao número da versão da qual elas são derivadas. As edições de uma versão serão numeradas em sequência neste campo adicional. Por exemplo: as edições da versão particular que é derivada da versão congelada 3.2 terão números 3.2.1, 3.2.2 e assim por diante.

4.2 Identificação

Um objeto de projeto OP é representado por um conjunto de versões, e uma identificação é associada a esse conjunto.

Uma versão é unicamente identificada, dentro do conjunto de versões de um dado objeto, pelo seu número. O par (OP, número de versão) identifica unicamente a versão no banco de dados [Ditt88]. Números de versões são gerados automaticamente, segundo o mecanismo descrito anteriormente.

Outro tipo de identificação possível é através de um par (OP, rótulo), onde rótulo especifica a versão através de associação a uma característica da versão. Os rótulos existentes serão: CORRENTE, ULTLIB e ULTEF. O significado desses rótulos será detalhado adiante.

4.3 Tipos de versões

Um sistema de software é geralmente desenvolvido por um grupo de projetistas, e posteriormente liberado para uso geral. Cada projetista é dono de um conjunto de versões, que só podem ser referenciadas por ele mesmo. Para que estas versões possam ser liberadas, elas devem ser totalmente verificadas; para tanto devem ser combinadas com as versões de outros módulos, que ela referencia. Mas estas versões referenciadas podem pertencer a outros projetistas, e por sua vez, também não podem ser liberadas antes de serem testadas. Além disso, dois ou mais usuários que estejam desenvolvendo um objeto em conjunto precisam ter acesso às mesmas versões. Portanto, em um ambiente de desenvolvimento de software os integrantes de um mesmo grupo de projeto devem poder compartilhar algumas versões ainda em desenvolvimento [Katz84,Katz87].

Assim, três tipos de versões são necessárias: particulares, efetivas e liberadas.

4.3.1 Capacidades de Cada Tipo de Versão

Uma versão pode ter diferentes capacidades conforme o seu tipo. As versões particulares só podem ser referenciadas por quem as criou, isto é, seu dono. As versões efetivas podem ser referenciadas pelos integrantes do grupo de projeto a que o seu criador pertence. E finalmente, as versões liberadas podem ser referenciadas por todos os usuários do sistema.

Uma versão efetiva é criada por promoção de uma versão particular. Esta promoção pode ser automática ou feita pelo dono da versão, sendo que este deve ser integrante do grupo ao qual ela passa a pertencer.

Tanto versões liberadas como versões efetivas não podem ser atualizadas, isto é, estão congeladas. Quando algum projetista quiser fazer uma modificação em uma versão liberada ou efetiva, precisa derivar uma versão particular desta versão.

A criação de uma versão particular pode se dar a partir de um arquivo de trabalho ou de derivação da última versão disposta na sequência linear em que

o histórico é organizado. Se a versão for criada por uma derivação, é feita uma cópia da versão da antiga, da qual ela se originou. No caso da versão antiga ser uma versão particular, o criador da nova versão deve ser também o seu dono, e esta versão é efetivada automaticamente.

Cada projetista com direito de acesso só pode ter uma versão particular de cada objeto, mas vários projetistas podem ter versões particulares da última versão congelada de um objeto de projeto em paralelo. Isto significa que só existe concorrência no último nível. Quando uma destas versões for congelada, todas as outras continuam a existir, mas passam a ser derivadas desta que foi recentemente congelada.

Uma versão particular pode ainda ter várias edições controladas pelo sistema. Quando o projetista achar que uma das edições deve se tornar uma versão, esta edição é armazenada como derivada da versão que deu origem a versão particular que esta edição representava. Neste momento todas as outras edições que representavam esta versão particular serão eliminadas e as versões particulares de outros projetistas, em ramificações paralelas a desta versão, passam a ser derivadas desta nova versão. Consequentemente ela é efetivada, congelada e colocada a disposição do grupo.

4.4 Associação de Rótulos

A última edição da versão particular criada por um projetista será referenciada pelo rótulo CORRENTE, para aquele projetista. A versão particular só recebe um número de versão no momento em que for congelada. Um projetista que não está editando uma versão particular de um objeto de projeto, pode associar o seu rótulo CORRENTE com uma versão específica, desde que faça um pedido explícito para isto. No caso de um projetista fazer uma referência a versão CORRENTE, se este rótulo não estiver associado a nenhuma versão, a associada com ULTLIB será usada.

O desenvolvimento de um objeto de projeto gera, normalmente, muitas versões deste, até que ele chegue a um estado, onde esteja consistente e maduro o suficiente, para ser liberado. A versão liberada mais recentemente é chamada de última liberada, e associada ao rótulo ULTILIB. Pode-se ainda, especificar em uma configuração um número seguido de ULTLIB, como por exemplo: 3.ULTLIB, que significa a versão liberada mais recentemente com versão de interface 3.

O rótulo ULTEF, do mesmo modo, é associado a última versão que tiver sido efetivada, isto é, posta a disposição do grupo.

5 Referências Genéricas e Configurações

Em um sistema de software, quando um módulo contém uma referência a outro módulo e existem muitas versões deste último, não é desejável ter que associar uma versão específica à referência, pois isto tornaria o processo de evolução do sistema muito inflexível. Uma referência genérica é uma referência a um objeto de projeto independente da versão a ser usada.

Quando não se deseja fazer ligações estáticas entre objetos, os objetos podem ser refenciados genericamente, e conectados dinamicamente através de uma definição de configuração. Contudo, quando uma nova versão de um objeto de projeto é congelada, a configuração deve referenciar estaticamente todos os seus componentes [Ditt88].

5.1 Definição de Configuração

Uma definição de configuração é uma relação binária, um conjunto de pares do tipo (identificador do objeto, identificador de versão), ou (OP,v), mais uma opção "default". Cada par especifica qual versão deve ser usada para um objeto, quando este for referenciado. Se não existe um par para um determinado objeto, a opção "default" é usada. Uma definição de configuração póde ser criada em qualquer instante e armazenada no banco de dados, mas a definição só tem efeito quando for ativada. Só uma definição de configuração está ativa em um certo momento, e a ativação de uma definição automaticamente desativa qualquer outra.

A definição de uma configuração consiste de um nome, um default, um conjunto de entradas diretas, um conjunto de entradas indiretas e um conjunto de entradas de inclusão, sendo que qualquer um destes conjuntos pode estar vazio.

Uma entrada direta (OP, v): especifica explicitamente qual versão v deve ser usada para um objeto de projeto OP.

Uma entrada indireta (OP, dc): especifica uma definição de configuração dc para um objeto de projeto OP.

Uma entrada de inclusão (dc, p): especifica uma prioridade p para uma definição de configuração dc. A prioridade define a ordem global das entradas de inclusão.

Cada projetista pode fazer as suas definições de configuração, e.o ambiente gera números sucessivos para definições feitas: DC1, DC2, DC3 e assim por diante.

5.2 Definição de Configuração "Default"

Quando um projetista está desenvolvendo objetos, que fazem parte de um sistema de grande porte, normalmente ele quer que a configuração usada para

testes do sistema se constitua das versões mais recentes dos módulos que ele estiver desenvolvendo, e as últimas versões liberadas para todos os outros objetos. Portanto, esta será a definição de configuração "default":

- para os objetos que tiveram uma versão associada com um rótulo CORRENTE, para aquele projetista, a versão associada a este rótulo será utilizada.
- para todos os outros objetos referenciados, as últimas versões liberadas serão utilizadas.

5.3 Construção de Configuração

No momento de utilização de uma definição de configuração, constroi-se uma de configuração de execução, a partir da definição de configuração ativa no momento. A configuração de execução é a que será usada para resolução das referências genéricas.

Para a construção da configuração de execução, define-se um conjunto S, que conterá os pares desta configuração. Inicialmente S está vazio. Em seguida, as entradas da definição são percorridas e, para cada uma delas, adiciona-se ao conjunto S uma, nenhuma ou várias entradas de acordo com as seguintes regras:

Para entradas diretas (OP,i): a entrada com o número de versão explícito é copiada em S, inalterada. Se uma entrada direta especifica ULTLIB ou ULTEF procura-se o número i correspondente à ultima versão liberada ou efetivada respectivamente, e o par (OP, i) é então inserido em S. Se a entrada direta especifica CORRENTE, procura-se a versão(.edição) i associada a esse rótulo para aquele objeto de projeto e para o usuário que está usando a configuração, e o par (OP, i) é inserido em S.

Para entrada indiretas (OP, dc): a versão do objeto OP a ser usada é a mesma que seria usada se a definição de configuração dc estivesse ativa. O algoritmo descrito é aplicado recursivamente, até se encontrar o número de versão i do objeto OP definida na configuração dc, e o par (OP, i) é adicionado a S.

Para entradas de inclusão (dc, p): as entradas de inclusão são consideradas na ordem de prioridade indicada por p. O algoritmo é aplicado recursivamente para construir o conjunto de pares S' correspondendo a definição de configuração dc. É feito então, o "merge" de S com S'. Se existe uma entrada para um objeto de projeto OP em S, uma entrada para OP em S' é ignorada durante o "merge". Se existe mais de uma entrada de inclusão, o algoritmo é aplicado a cada definição de configuração, sendo que as entradas conflitantes serão consideradas segundo a prioridade p.

Uma configuração de execução pode ser armazenada no banco de dados, o que evita que o procedimento acima tenha que ser repetido cada vez que a definição for usada.

No congelamento de um objeto, é especificada uma configuração de execução que deve ser associada ao objeto liberado, de maneira que as ligações com os objetos referenciados sejam feitas estaticamente.

Um exemplo da aplicação do algoritmo acima é dado, supondo-se que as seguintes definições de configuração tenham sido feitas:

	default	diretas	indiretas	inclusão
DC1		(OP4,3.2) (OP5,2.0) (OP6,1.0.1) (OP7,CORRENTE)		
DC2	CORRENTE	(OP4,ULTLIB) (OP5,ULTEF) (OP6,2.0)		
DC3	ULTLIB	(OP2,ULTEF)	(OP4,DC1)	
DC4	CORRENTE	(OP2,ULTLIB) (OP3,CORRENTE)	(OP4,DC3)	(DC2,2) (DC1,1)

Assumindo ainda, a seguinte situação para os indicadores de versão corrente, de última versão liberada e de última versão efetivada para o projetista P1:

objeto	CORRENTE P1	ULTLIB	ULTEF
OP2	-	1.1	1.2
OP3	1.4.1	1.2	1.3
OP5	9.0.2	7.4	8.3
OP6	-	6.0	6.1
OP7	4.2.1	3.0	3.0

Quando DC4 for ativado pelo projetista P1, o algoritmo será executado produzindo a seguinte visão de execução:

default CORRENTE

objeto	versão
OP2	1.1
OP3	1.4.1
OP4	3.2
OP5	8.3
OP6	2.0
OP7	4.2.1

Assim, o projetista P1 terá o sistema construido com a versão 1.1 do objeto de projeto 2, a versão 3.2 do objeto 4, a versão 8.3 do objeto 5, a versão 2.0 do

objeto 6, e as suas versões CORRENTES dos objetos de projeto 3 e 7 que são as edições derivadas das versões 1.4 e 4.2 respectivamente, de número 1.

6 Conclusão

O mecanismo definido teve como objetivo a simplicidade de uso. O projetista deve ter controle sobre as versões de seus objetos naturalmente, isto é sem que ele tenha preocupações com o mecanismo no momento de criar ou usar uma versão. Além disso, o sistema de controle poderá ser transparente para um usuário que não o conheça.

Extensões ao mecanismo devem incluir manutenção de alternativas e equivalências de versões de objetos, e controle de versões de definição de configuração. No entanto, estas extensões devem ser feitas sem se perder de vista a simplicidade de uso.

Referências

[Bato85] Batory, D. S., Kim, W., Modeling Concepts for VLSI CAD Objects, ACM Transactions on DAtabase Systems, vol. 10, n. 3, (Sep 1985), pp 322 - 346

[Beck89] Beck, K., Tratamento de versões em Ambientes de Projeto, Banco de Dados Para Aplicações não Convencionais, IV EBAI, Termas do Rio Hondo, (Jan 1989), pp 59 - 76

[Belk86] Belkhatir, N., Estublier, J., Experience with a Database of Programs, SIGPLAN Notices, vol. 22, n. 1, (Jan 87), pp 84 - 91

[Chou86] Chou, H., Kim, W., A Unifying Framework for Version Control in a CAD Environment, Proceedings of the 12th International VLDB Conference, Kyoto, (Aug 1986), pp 336 - 344

[Cohe88] Cohen, E. S., Soni, D. A., Glwecker, R., Hasling, W. M., Schwanke, R. W., Wagne, M. E., Version Management in Gypsy, ACM, (1988), pp 201 - 214

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

[Drum87a] Drummond, R., Liesenberg, H., A_HAND: Ambiente de Desenvolvimento de Software Baseado em Hierarquias de Abstração em Níveis Diferenciados, Anais do IV Encontro de Trabalho do Projeto Ethos, Petrópolis, RJ, (Abr 1987), pp 313 - 322

[Drum87b] Drummond, R., Liesenberg, H., Requisitos para um Ambiente de Desenvolvimento de PROGRAMAS, I Encontro IBM de Ciência e Tecnoiogia em Informática, Rio de Janeiro, RJ, (Nov 1987)

[Katz84] Katz, R. H., Lehman, T. J., Database Support for Versions and Alternatives of Large Design Files, IEEE Transactions on Software Engineering, vol SE-10, n. 2, (Mar 1984), pp 191 - 200

[Katz86] Katz, R. H., Chang, E., Bhateja, R., Version Modeling Concepts for Computer-Aided Design Databases, SIGMOD Record, vol. 15, n. 2 (Jun 86), pp 379 - 386

[Katz87] Katz, R. H., Bhateja, R., Chang, E. E., Gedye, D., Trijanto, V., Design Version Management, IEEE Design & Test (Feb 1987), pp 12 - 22

[Kim87] Kim, W., Banerjee, J., Chou, H., Garza, J. F., Woelk, D., Composite Object Support in an Object Support in an Object-Oriented Dtabase System, Proceedings of the OOPSLA'87, Orlando, (Oct 1987), pp 118 - 125

[Kim88] Kim, W., Chou, H., Banerjee, J., Operations and Implementation of Complex Objects, IEEE Transactions on Software Engineering, vol. 14, n. 7, (Jul 1988), pp 985 - 995

[Silv88] Silva, F. B., Liesenberg, H., Drummmond, R., Programação em Cm, Relatório Fapesp, (1988)

[Thom88] Thomas, D., Johnson, K., A Configuration Management System fom Team Programming, Proceedings of OOPSLA'87, (Sep 1987), pp 135 - 141