OBJETOS DE PROJETO E ENGENHARIA DE SOFTWARE: um enfoque prático

Carlos A. M. Pietrobon
Arndt von Staa

Departamento de Informática - PUC/RJ
Rua Marquês de São Vicente, 225, Gávea
22453 - Rio de Janeiro - RJ

RESUMO

Objetos de projeto e controle de versão e configuração, tais como usados atualmente, são inadequados para apoiar o desenvolvimento de software. Neste trabalho é apresentada uma nova maneira de se definir objetos de projeto e um novo papel a ser desempenhado pela gerência de versão e configuração.

1 - INTRODUÇÃO

Estão sendo desenvolvidos sistemas de software cada vez mais complexos. Para vencer a complexidade destas novas aplicações, tem sido proposto dividi-las em elementos (partes) de complexidade reduzida [BECK88] e, portanto, mais facilmente tratáveis. Estes elementos são chamados, aqui, de objetos de projeto. Grandes sistemas de software, hoje em dia, são constituídos de centenas, ou mesmo, milhares de objetos de projeto. São exemplos de objetos de projeto: os códigos dos programas, módulos e funções que compõem um sistema, o projeto estruturado, uma folha (nível) de um diagrama de fluxo de dados, etc.

Para se entender o que são estes objetos de projeto e as atividades relacionadas com eles, lançamos mão de uma analogia com a Engenharia Mecânica. Assim, tomando-se como exemplo um carro, pode-se vê-lo como um objeto de projeto complexo que é construído a partir de objetos menos complexos, tais como, motor, carroceria, caixa de marcha, etc. Estes, por sua vez, podem ser decompostos em partes sucessivamente mais simples até se chegar à objetos elementares, que não são compostos de nenhuma parte mais elementar, tais como, parafusos, engrenagens, etc.

Para permitir a construção do carro (ou de qualquer parte que não seja elementar), é necessário o conhecimento de quais são os objetos de projeto que o compõem, bem como, os diversos relacionamentos entre estes objetos e a funcionalidade de cada um. Este conhecimento habilita, além da construção propriamente dita, outras atividades tais como manutenção e controle de qualidade.

Na Engenharia de Software existem conceitos de objetos de

projeto e atividades de manipulação de objetos semelhantes aos da engenharia mecânica. No entanto, nesta última, estes conceitos e técnicas já são amplamente dominados. Já, na Engenharia de Software, não se tem o domínio da arte, pois, o processo de desenvolvimento de software não é completamente entendido e, portanto, não se conseguiu criar um Ambiente de Desenvolvimento de Software (ADS) completo.

Segundo [SHIL89] um ADS completo é aquele que suporta o crescimento do sistema de software desde o princípio, através do projeto e implementação, chegando à manutenção. Ele precisa suportar controle de versão e configuração, teste e depuração. Ele precisa suportar rastreabilidade, ou seja, deve permitir rastrear a especificação de uma característica através do projeto até a implementação e a habilidade de determinar, dada uma implementação, a partir de quais elementos da especificação e projeto ela é derivada.

Para controlar este conjunto de objetos (elementos), tem sido propostos gerenciadores de versão e configuração. Estes realizam o gerenciamento da integridade de sistemas com milhares de objetos, tirando este encargo das mãos do usuário. Na verdade, para sistemas muito grandes, sem automação é impossível controlar todas os objetos componentes existentes.

No entanto, a atual maneira de decompor o sistema sendo desenvolvido em partes menores, o conceito de objetos de projeto e o seu controle por parte dos gerenciadores de versão e configuração apresentam muitas limitações e/ou problemas que dificultam a manipulação do sistema em desenvolvimento. Por exemplo, falta o registro explícito dos relacionamentos (dependências) entre os diversos objetos, o que dificulta a propagação de alterações e a manutenção de consistência dos objetos do sistema. Falta, também, uma forma de armazenamento que permita um 'acesso direto' aos objetos de projeto, uma base de dados onde os objetos armazenados possam ser extraídos em função de suas propriedades. Isto seria fundamental para o reuso de software. Na seção 2 são levantadas outras limitações.

Em função das limitações e problemas existentes, é necessário um meio mais efetivo que permita aos diversos participantes do projeto documentar, particionar, interrelacionar, criar, eliminar, navegar, checar e reutilizar as diversas partes geradas nas diversas fases de desenvolvimento, de tal forma que, segundo a nossa visão, forneça um apoio mais realista ao desenvolvimento, manutenção e reprojeto de software.

O ADS apoiado em computador TALISMAN [SI90] possui um novo enfoque sobre objetos de projeto que endereça os requisitos supra citados. Assim, é objetivo deste trabalho descrever como TALISMAN considera e manipula os objetos de projeto. Para TALISMAN, um

objeto de projeto vem a ser uma estrutura complexa composta de vários campos (atributos) e que possui um nome que o identifica. Alguns destes campos determinam como e com quem o objeto se relaciona. Outros são campos de especificação e/ou de implementação. Por fim, cada objeto possui, ainda, campos indicadores das características do próprio objeto. Estes campos contém diferentes tipos de informações que objetivam apoiar as diversas atividades que podem ser executadas sobre o objeto.

Na seção 2 são levantados alguns problemas decorrentes do atuais conceitos de objetos de projeto e gerência de configuração e versão de software. Na seção 3 é definida uma terminologia básica. Na seção 4 é descrito como seriam os objetos de projeto para que permitam resolver os problemas identificados na seção 2. Na seção 5 são avaliadas diversas características do modelo proposto. Por fim, tem-se as conclusões.

2 - PROBLEMAS

Como mencionado anteriormente, a maneira convencional como são vistos os objetos de projeto e a gerência de configuração e versão de software, possui muitas limitações. A seguir citamos alguns dos mais evidentes:

a) A decomposição e o controle das partes resultantes, de forma automatizada, tem sido realizados quase que exclusivamente sobre os objetos gerados na fase de implementação do software, desprezando objetos gerados em outras fases do ciclo de desenvolvimento.

b) Não existe, agregada ao objeto, nenhuma informação que o explique ou o relacione a outros objetos do sistema. Assim, é difícil entender exatamente o que ele faz e/ou representa e/ou com quem se relaciona. As vezes existem comentários ao longo da representação. No entanto, estes costumam ser deficientes em termos quantitativos e qualitativos. Estas carências de informação são especialmente ruins para as atividades de manutenção, controle de consistência e recuperação de objetos armazenados.

c) Poucos tipos de relações entre objetos têm sido suportados pelas atuais ferramentas de controle de versão e configuração. Como exemplo destas relações suportadas, temos as de composição e precedência ('foi gerado a partir de' ou 'gera'). Isto acarreta diversos problemas que influenciam a produtividade e a qualidade do software sendo construído, como os citados a seguir:

c1) Aparecimento de inconsistências em diversas partes do software. Devido à falta de registro destas relações, o usuário fica com toda a responsabilidade de descobri-las e realizar as atividades que as mantém válidas. Como o usuário pode não perceber todas as relações entre os documentos, inconsistências podem ser geradas.

c2) Perda de parte do esforço dispendido durante a construção do software, devido a ausência do registro deste esforço. Durante a construção de uma dada representação o implementador possui ao seu dispor destas informações que, se existirem registradas e acessíveis, facilitarão em muito atividades futuras, como a manutenção. Por exemplo, que objetos se relacionamcom o objeto gerado.

c3) É difícil propagar alterações ou avaliar o impacto de alterações. Quando se faz uma alteração em uma parte do sistema, outras partes, que de alguma forma se relacionam com a parte alterada, tem que ser checadas e, possivelmente, alteradas. Se não existirem relações registradas explicitamente, o usuário tem que fazer a propagação na 'força bruta', o que implica em erros. Um outro problema que dificulta a propagação seriam relacionamentos feitos entre objetos de granularidades inadequadas (por exemplo à nível de módulo), que dificultam o trabalho de relacionamento, por parte do usuário, da porção alterada com o todo impactado.

d) Em geral, as relações registradas ocorrem entre objetos de projeto em um nível de abstração (granularidade) inadequado ou arbitrário. Por exemplo, as relações de composição e ordenação temporal relacionam objetos de projeto muito grandes, não considerando detalhes específicos (partes) de sua implementação. Isso não é adequado pois, durante o ciclo de vida do software, muitas atividades necessitam de informações específicas sobre determinadas partes do objeto. Por exemplo, é comum considerar-se um programa (módulo compilável independentemente) como uma parte individualmente identificável do sistema, armazenado em um arquivo e tendo uma versão para cada estado de desenvolvimento por que passou. Durante o seu ciclo de vida, muitas das alterações e implementações são feitas sobre partes do programa visando incluir ou alterar funcionalidades específicas. Assim, para o usuário, muitas vezes, mais importante é o conhecimento destas porções de funcionalidades específicas, do que o programa como um todo.

e) Objetos possuem baixa extensibilidade, ou seja, são difíceis de serem adaptados à mudanças de especificação [SHIL89].

f) É difícil reutilizar os objetos. Embora o desenvolvimento de software tenha uma natureza repetitiva [MEYE88], normalmente todo novo desenvolvimento começa do nada. Por exemplo, sempre se usa padrões básicos para, leitura, pesquisa, ordenação, etc. No

entanto, estes sempre são recriados todas as vezes que deles se precisa. Seria altamente desejável se, a cada novo desenvolvimento, pudéssemos consultar um catálogo de objetos e, através da seleção e combinação adequada destes objetos, pudéssemos construir o novo sistema sem ter que 'reinventar a roda' toda vez. Para isso, é necessário a existência de uma base de dados de objetos de projeto e meios para acessar o objeto desejado através da especificação de características do objeto requisitado. Hoje em dia, em muitos ambientes de projeto, isto é difícil de conseguir, uma vez que se armazenam apenas as representações dos diversos objetos. Outras, e relevantes informações sobre os objetos encontram-se, muitas vezes, na cabeça de quem as criou.

g) As estruturas de armazenamento (organização física) dos objetos costumam ser diferentes para cada tipo de objeto. Isso exige diferentes métodos de acesso à base de objetos e dificulta a integração entre as diferentes ferramentas existentes no ambiente.

3 - DEFINIÇÕES

Devido a disparidade de significado de termos e conceitos existentes na literatura corrente, nesta parte são feitas várias definições que facilitarão o entendimento das idéias expostas no texto.

A representação do software segundo alguma linguagem de representação pode ser estratificada em diferentes camadas ou *níveis de abstração*. Assim, os objetos definidos em camadas mais altas, mais abstratas, são mais gerais e são construídos a partir de objetos definidos em camadas inferiores.

Um *objeto de projeto*, designa uma entidade ou porção do sistema sendo construído, em algum nível de abstração, em alguma linguagem de representação, que possa ser univocamente identificada e manipulada. Os objetos de projeto podem ser vistos como instâncias no nível a que pertencem. Por exemplo, os diversos programas que compõem um sistema são objetos de projeto no nível programa.

O nível de abstração, que pode ser pensado como sendo uma classe, segundo a terminologia orientada à objetos, define atributos que todas as instâncias dele terão. A nível de instância estes atributos são chamados campos. Os campos de um objeto de projeto são:

- NOME: é uma chave que permite identificar univocamente o objeto no universo de objetos que compõem o sistema.

- SEQÜÊNCIAS: são seqüências de caracteres. Podem representar sinônimos pelos quais o objeto pode ser, alternativamente, reconhecido.

- TEXTO: contém a descrição (formal ou não) de um dado aspecto do objeto, podendo ser vazio.

- VALOR FORMATADO: é uma estrutura de dados que contém informações a serem usadas pelo objeto associado. Por exemplo, coordenadas, em um diagrama, do ícone correspondente ao objeto

- RELAÇÃO: é um conjunto de tuplas com as seguintes informações: tipo da relação e o objeto com o qual o objeto associado se relaciona (por exemplo, <'gera', obj.proj.10>)

Um *objeto de projeto elementar* é o objeto de projeto que não faz referência a (não é composto por) nenhum outro objeto mais elementar. Isto quer dizer que ele não possui nenhuma relação do tipo 'é composto por'.

Um *objeto de projeto não elementar* é o objeto de projeto que referencia (é composto por) outros objetos de projeto mais elementares.

Uma *forma completa* é um estado de um objeto de projeto, indicando que ele está completamente formado e possui qualidade aceitável, segundo uma dada especificação de qualidade. Ao manter (corrigir, evoluir) um objeto de projeto, ele será transformado de uma forma completa para outra. Note que, ter sido eliminado, é uma forma completa final na seqüência de evolução.

Uma *versão* de um objeto de projeto é uma das possíveis formas completas, que um objeto de projeto pode ter ao longo do tempo.

Uma *configuração* é a composição de diversos objetos, cada qual em uma dada forma completa. A configuração é determinada por um 'script'. A construção de uma representação física para um objeto de projeto não elementar é função da determinação de uma configuração. Esta determinação é feita selecionando-se um 'script' apropriado. Esta representação física será constituída por um conjunto de objetos diferentes, uma versão para cada objeto de projeto de nível inferior considerado.

Um *script* é um agregado (lista) que contém uma relação de formas completas de objetos que formam um objeto composto.

4 - MODELO

Nesta seção é apresentado o modelo do objeto de projeto usado por TALISMAN. Esta descrição é feita, principalmente, sob um ponto de vista funcional.

Como já mencionado, a divisão do sistema em partes e a manipulação destas partes, como se faz atualmente, são inadequadas para prestar um apoio efetivo ao processo de desenvolvimento de software. Trabalhar com partes muito grandes, ou muito abstratas, pode não apoiar adequadamente as atividades de desenvolvimento de software pois, muitas destas atividades são realizadas dentro de um escopo mais restrito. Por exemplo, o usuário não modifica um programa mas, uma ou mais funções executadas pelo programa. Por outro lado, nada impede que programas inteiros, com centenas de linhas de código, sejam considerados como unidades de manipulação. Tudo depende da aplicação sendo desenvolvida.

Assim, além de ser difícil fixar um tamanho ou nível de abstração como o ideal para apoiar as atividades de desenvolvimento de software, percebe-se que isto realmente não é importante. O importante é que, qualquer que seja a parte do software, independente da linguagem de representação usada, do nível de abstração considerado ou do seu tamanho físico, esta parte tenha tanta importância e tenha tratamento semelhante a outros quaisquer objetos de projeto. Uma das dificuldades das ferramentas de controle de versão e configuração existentes é justamente esta: elas não reconhecem objetos outros que não sejam sistemas e programas. Os outros objetos são controlados pelo usuário.

Deste modo, para tentarmos definir como seriam os objetos de projeto práticos para a engenharia de software - que permitem obter as diversas características citadas na introdução, particularmente o reuso, a compatibilidade, a extensibilidade e o registro de informações - fazemos três observações:

a) O software pode ser estruturado segundo diferentes níveis de abstração ou detalhamento (visão top-down), dependendo, esta estratificação, da linguagem de representação usada.

b) O software pode ser organizado segundo diferentes visões. Por exemplo: visão de interface abstrata, visão de interface concreta, visão funcional, visão dos dados, etc.

c) O processo de desenvolvimento de software pode ser visto como um processo de transformação entre representações.

Considerando a primeira característica (existência de vários níveis de abstração), podemos imaginar o sistema como sendo o

objeto de mais alto nível de abstração. Este objeto, por sua vez, é construído a partir de objetos sucessivamente mais elementares até se chegar aos objetos de mais baixo nível de abstração, que não possuem referências a (não são compostos a partir de) nenhum outro objeto mais elementar.

Considerando a segunda característica, pode-se imaginar uma estruturação segundo diferentes visões. A decomposição dependerá, pelo menos, da linguagem de representação usada e do tipo da aplicação. Por exemplo, podemos dividir um sistema, do ponto de vista funcional e dos dados, como na Figura 1 abaixo:

VISÃO FUNCIONAL	VISÃO DE DADOS
sistema	base de dados
programa	visão da base de dados
processo	região crítica
módulo	arquivo físico
pacote	tipo de dados
bloco	lista de declarações
instruções	declarações elementares

Figura 1: Duas possíveis estruturações de um software

Tendo em mente a estruturação funcional do sistema de software, podemos fazer várias considerações. Uma delas aponta que o objeto mais elementar, qual seja, o bloco, é aquele que, efetivamente, contém a informação que será usada para montar a representação do sistema, uma vez que é ele que contém um conjunto de instruções. Levar a granularidade a nível de instrução aumenta em muito o 'overhead' uma vez que, raras vezes uma única linha (instrução) corresponde a uma funcionalidade completa e relevante. Em adição, a granularidade a nível de bloco cria uma flexibilidade com relação ao número de instruções. Evidentemente, cria, também, a necessidade de um critério 'tamanho funcional' vinculando o tamanho do texto a exatamente uma funcionalidade. Outra vantagem de se considerar 'bloco' (i.e. funcionalidade elementar) como objeto elementar, é que nos tornamos independentes da linguagem de programação utilizada.

Outra observação relativa à Figura 1 aponta que, como os objetos elementares (blocos) possuem as porções de representação a partir das quais os objetos de projeto não elementares são construídos, a representação da aplicação e todos os objetos de projeto não elementares não são armazenados diretamente. Quando se deseja construí-los fisicamente, um dos vários 'scripts' possíveis associado ao objeto de projeto é selecionado e percorrido. Cada

'script' determina uma versão. Cada um contém a relação de todos os objetos menos elementares que compõem o objeto ao qual está associado. A partir daí, o 'script' dos objetos referenciados são percorridos para se obter a relação de objetos mais elementares. O processo continua até se obter um conjunto só de objetos elementares.

Considerando a terceira característica (transformação de representações), vemos que o desenvolvimento de software passa por diversas etapas (estados). Em cada estado o software (ou parte dele) é representado em alguma linguagem de representação. O resultado de cada etapa pode ser visto como o resultado da transformação de um ou mais objetos definidos em etapas anteriores, acrescidos de novos fatos, fornecidos pelo desenvolvedor. Na verdade, esta transformação é o resultado de várias transformações mais elementares, cada uma agindo sobre porções bem determinadas do(s) objeto(s) de entrada para gerar, também, uma parte específica no objeto de saída.

A cada uma destas transformações, no nosso modelo, o desenvolvedor registra os relacionamentos entre o objeto criado e os outros existentes que, de alguma forma, relacionam-se com ele. Assim, a medida que se constroem os diversos objetos que compõem o sistema propriamente dito, o desenvolvedor vai construindo uma rede onde os nodos são os objetos e os arcos são os relacionamentos entre objetos. Como entre dois objetos podem existir mais de um relacionamento, podem haver mais de um arco conectando dois objetos.

A criação e manipulação destas relações é feita através de um editor de estruturas [STAA90a]. Este editor considera o software como uma organização em rede onde cada nodo (objeto) é anotado. Um nó anotado possui diversos campos associados, cada um correspondendo a uma determinada classe de informação. Um desses campos seria o de relações onde, o usuário, baseado no seu conhecimento da atividade que está realizando, registraria as relações identificadas. A edição, propriamente dita, se dá através de uma ferramenta chamada formulário que especifica e coordena a edição dos campos apropriados para cada nó.

Desta maneira, o sistema não é mais um conjunto de objetos isolados, mas, um conjunto de objetos explicitamente relacionados. Isto facilita sobremaneira diversas atividades, entre elas, a manutenção da consistência. Quando um objeto é alterado, é possível saber com precisão que outras partes do sistema são, ou podem ser, impactadas pelas alterações. Além disso, navegando pelos documentos relacionados, pode-se fazer as alterações que levem o sistema a um estado consistente. Para que isso seja possível, é necessário que o ADS saiba com quem os objetos se relacionam ou, como aponta [HARR89], que a representação interna do sistema de software seja condutiva a este tipo de manipulação. Assim entre os campos do objeto está o campo relação, que contém

um conjunto de tuplas que informam com quem o objeto se relaciona e qual o tipo de relação. O tipo é importante para se determinar as operações a serem executadas sobre o objeto.

Quanto à arquitetura do sistema, o usuário não está obrigado a decompor o software em partes segundo um modelo bem determinado a priori (como o visto na Figura 1). Ele o decompõe segundo suas necessidades e bom senso. A estratificação é, agora, mais uma maneira de se visualizar o software do que um método que orienta sua construção. O fato de um objeto 'ser composto' por outros é apenas mais um tipo de relação entre dois objetos que tem que ser mantida pelas ferramentas de controle de versão e configuração.

Quanto ao problema de versões, no nosso caso, não se versiona o objeto diretamente, mas, seus campos. No caso de objetos não primitivos, para cada versão tem-se uma entrada no campo relação que aponta para o objeto 'script' que contém todas as informações para se construir a versão. Para objetos elementares, no campo relação tem-se uma entrada para cada versão que aponta diretamente para o objeto com as informações (representações) a nível elementar, que caracteriza o objeto.

Desta maneira, o ADS não possui todas as versões (representações) de todos os objetos construídos mas, as informações que permitem construí-los, se for desejado. Só os objetos elementares têm suas versões (representações) mantidas na íntegra. Isto acarreta economia de memória (um objeto pode ser usado por vários objetos), simplicidade de construção (não se tem que processar deltas (diferenças entre duas versões)[TCHY85]) e facilidade para manter a consistência (alterações em um único ponto se reflete em todas as partes do sistema onde é usado).

Finalmente, o ambiente de desenvolvimento pode apresentar uma espécie de comportamento ativo [AGHA86], que é independente da ação do usuário. Em função do tipo do objeto, são definidas várias atividades que devem ser realizadas quando forem detectadas alterações no estado do objeto e/ou do sistema. Assim, sempre que, em função das operações de acesso do usuário forem detectadas alterações no estado do sistema, o ADS (não o objeto propriamente dito), dispara operações que executam assincronamente ao trabalho do usuário. Como exemplo, temos a propagação de alterações, testes de consistência, etc. Muitas destas operações assíncronas fazem uso da malha de relacionamentos criados durante o desenvolvimento dos diversos objetos.

5 - ANÁLISE

Com o modelo apresentado, estamos dando um outro enfoque ao problema de controle de versão e configuração, integrando-o ao

processo de desenvolvimento de software. Assim, analisamos abaixo a adequação do modelo a diversos enfoques do problema de desenvolvimento de software.

1) Representação de um objeto de projeto

A representação de um objeto de projeto será construída a partir de partes elementares existentes na base de software. Desta forma, até a sua construção efetiva, o objeto de projeto é apenas uma descrição de que partes o compõem. Isto evita desperdício de memória devido a manutenção de diversas representações (versões), efetivamente construídas, do mesmo objeto. Além do mais, se duas versões têm uma parte em comum, esta parte só existe armazenada uma única vez na base de software.

2) Controle de versão e configuração

Estes controles, agora, têm conotações bastante diferentes das comumente usadas. O controle de versão não se faz mais a nível de representação dos macro objetos de projeto (programas para o caso de um linguagem de representação de código fonte), mas, em todos os níveis da representação. No entanto, não se faz a duplicação do objeto em si, mas, altera-se seus campos. Para definir a interdependência, cria-se um novo 'script'. Este 'script' relaciona as versões de objetos interrelacionados e/ou as versões dos campos.

Cabe observar que o 'script' pode, em muitas ocasiões, ser montado através de algum mecanismo de inferência. Isto torna desnecessário (ou menos necessário) o uso de listas explícitas.

Já, o controle de configuração, se preocupa em manter 'scripts' e gerar (construir) uma configuração de um objeto complexo. Assim, um objeto de projeto pode possuir diferentes visões de implementações ('scripts'). Cada uma delas mostra uma maneira diferente de se construir a configuração. Este controle impedirá, por exemplo, que um objeto de baixo nível seja removido, enquanto algum objeto de projeto não elementar o referenciar.

3) Processo de desenvolvimento de software

Durante o desenvolvimento do software, os diversos objetos são obtidos, muitas vezes, a partir de outros, de tal forma que, podemos registrar graficamente estas relações de derivação como um grafo. Além disso, transformações entre objetos de alto nível de abstração podem ser vistas como resultado de uma série de transformações realizadas sobre partes mais elementares da sua representação, que, quando completamente realizadas, produzem um novo objeto consistente interna e externamente.

Cada uma destas subtransações extraem campos de um ou mais

objetos, possivelmente acrescentam mais alguma informação, fornecida pelo usuário, geram e/ou substituem campos no objeto resultante. Assim, com a aplicação sucessiva destas subtransações, um novo objeto complexo vai aos poucos sendo construido. Ora, o registro das atividades realizadas por estas transações é particularmente útil sob o ponto de vista do desenvolvimento do software pois, entre outras coisas, está se registrando porções de software que se relacionam, em um nivel em que estas porções são realmente usadas, o que vai permitir que:

a) o usuário (implementador ou manutenedor), valide modificações ou avalie o impacto de alterações feita ou por fazer, por poder acessar direta e objetivamente as diversas porções do software envolvidas, via o objeto visão.

b) o sistema automaticamente realize alguns tipos de controles

c) o usuário navegue de forma 'on-line' pela documentação do software, ou seja, que ele acesse qualquer objeto, qualquer que seja a representação usada para descrever este objeto.

4) Criação de uma base de software e o reuso de objetos

Uma vez que os objetos fazem referências diretas a outros objetos é necessário que os objetos de projetos estejam armazenados em uma base de dados (ou de software) onde possam ser acessados diretamente ('on-line'). Além disso, como, ao ser armazenado, o objeto deve ser identificado segundo suas características, ele pode ser acessado futuramente, via estas características, para ser reusado.

5) Tratamento de acesso concorrente

Para prevenir acesso concorrente a uma dada porção do sistema, usa-se um mecanismo de bloqueio ('lock'). No entanto, como as alterações são feitas à nível bastante elementar ou especifico, o bloqueio ocorre sobre uma parte relativamente pequena do produto de software. Além disso, existe uma cópia da porção (conjunto de objetos) sendo alterada, na base de dados pública, que pode ser usada até a nova versão estar disponivel. Assim, diminui-se bastante os problemas decorrentes de transações longas.

6) Avaliação de impacto de alterações e controle de consistência

A criação de uma rede de relações ligando todos os objetos permite, a partir de um dado objeto, atingir todos os outros que mantêm com ele algum relacionamento. Assim, de forma parcialmente automática, é possivel se propagar ou avaliar alterações por todas as partes envolvidas. Isso permite evitar muitos dos erros de consistência. A propagação automática é conseguida através de operações associadas ao objeto que são disparadas sempre que algum

estado for alterado. Por fim, como um objeto é definido apenas uma vez e usado várias vezes por objetos de nível superior, alterações feitas neste objeto se manifestam em todos os lugares onde ele é usado.

6 - CONCLUSÃO

Neste trabalho foi mostrada uma nova maneira de se encarar os objetos de projeto existentes em um ADS e, um papel mais ativo por parte das ferramentas de controle de versão e configuração.

O objeto de projeto, segundo a visão TALISMAN, é uma estrutura semanticamente complexa com informações sobre sua representação, operações que podem ser realizadas e dados para realizar tais operações. Devido ao registro de relações entre objetos, é possível a automação de diversas operações.

O uso de objetos de projeto como definidos aqui, exige e habilita a construção de uma base de software de projeto onde objetos podem ser reusados.

Ao usuário não se impõem nenhum método de desenvolvimento rígido. Ele tem toda a liberdade de estruturar o seu sistema conforme lhe interessar.

Finalmente, apontamos que, exceto o controle de versão, estes conceitos estão implementados no ADS assistido por computador chamado TALISMAN [STAA90b].

7 - BIBLIOGRAFIA

AGHA86 - AGHA, G. - Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, MA, 1986

BECK88 - BECKER, K. - Tratamento de Versões em um Ambiente de PAC para Sistemas Digitais. Dissertação de Mestrado da UFRGS, Porto Alegre, dez. 1988, 170pp.

HARR89 - HARRISON, W. H.; SHILLING, J. J.; SWEENEY, P. F. - Good News, Bad News: Experience Building a Software Development Environment Using the Object-Oriented Paradigm. In: Object-Oriented Programming: Systems, Languages and Applications, Conference Proceedings, Sigplan Notices, 24(10), p. 85-94, Oct. 1989.

MEYE88 - MEYER, BERTRAND. - Object-Oriented Software Construction,

Prentice Hall, International Series in Computer Science, 1988.

SHIL89 - SHILLING, J.J. - Three Steps to Views: Extending the Object-Oriented Paradigm, In: Object-Oriented Programming: Systems, Languages and Applications, Conference Proceedings, Sigplan Notices, 24(10), p.353-361, Oct.1989.

SI90 - TALISMAN Manual de Referência, STAA Informática, Rio de Janeiro, 1990.

STAA90a - STAA, ARNDT v. - Um Meta-editor de Estruturas, In: Anais do IV Simpósio Brasileiro de Engenharia de Software. Águas de São Pedro, p. 193-202, Out. 1990.

STAA90b - STAA, ARNDT v. - Resumo do Ambiente Talisman, In: Anais do IV Simpósio Brasileiro de Engenharia de Software. Águas de São Pedro, p. 273-274, Out. 1990.

TCHY85 - TCHY, W. - RCS - A System for Version Control, Software Practice & Experience, Chichester, 15(7): 637-54, July, 1985.