Assistência Inteligente ao Processo de Engenharia de Software

Victor F.A. Santander *
Itana M.S. Gimenes **
Paulo Cesar Masiero *

* Instituto de Ciências Matemáticas de São Carlos - ICMSC/USP
{vfsantan, masiero}@icmsc.sc.usp.br

** Departamento de Informática
Universidade Estadual de Maringá - UEM
itana@din.uem.br

Resumo

Um assistente inteligente pode apoiar a execução do processo de software baseado nas técnicas da inteligência artificial. Este artigo apresenta uma aplicação das técnicas de planejamento para apoiar o processo de desenvolvimento de software baseado no método ***fusion***. O primeiro nível da biblioteca de operadores desenvolvida, bem como o funcionamento dos algoritmos da assistência inteligente são apresentados. Os resultados obtidos são discutidos através da avaliação do protótipo desenvolvido para validação das técnicas propostas.

Palavras-chave: processo de software, método fusion, assistente inteligente.

Abstract

An intelligent assistant can support the execution of the software process based on artificial intelligence techniques. This paper presents the application of planning techniques to support the process of developing software based on the ***fusion*** method. The first level of the designed library of operators as well as the features of the algorithms for intelligent assistance are presented. The lessons learned are discussed based on the evaluation of the prototype developed to validate the proposed techniques.

Keywords: software process, fusion method, intelligent assistant.

1. Introdução

O crescente aumento da complexidade e riscos das aplicações tem exigido cada vez mais técnicas e ferramentas da engenharia de software. A cada dia que passa aumentam as exigências em relação à qualidade e redução dos custos dos produtos de software. Dessa forma, defrontamo-nos hoje com uma situação em que um suporte adequado ao desenvolvimento de sistemas complexos, atendendo às exigências do mercado atual, é um aspecto fundamental. Devido a estes fatores, uma das áreas da computação que mais tem evoluído nos últimos anos é a engenharia de software.

No esforço de desenvolver sistemas de grande porte, com menor custo, menos propensos a falhas, mais eficientes e passíveis de certificação, torna-se primordial não apenas atentar para o produto de software, mas também para o processo através do qual produzimos o software. O processo de software abrange todas as atividades técnicas e administrativas envolvidas na construção e manutenção de um software operacional. A busca do entendimento e definição explícita desses processos tem-se tornado uma constante entre os profissionais da engenharia de software. Parte-se do princípio que à medida que conseguimos melhorar o processo de engenharia de software (processo de software), diminuímos substancialmente os problemas do desenvolvimento e manutenção. Consequentemente, geramos produtos de melhor qualidade e integridade.

Pesquisas realizadas no SEI (*Software Engineering Institute*)[1] mostram que o impacto negativo de um processo mal formulado pode ser maior, por exemplo, que a utilização de tecnologias inadequadas durante o processo [HER96]. Desta forma, os profissionais da engenharia de software vêm procurando novos modelos e métodos para tornar o processo de software mais visível, sistemático e assistido. Além disso, existe também uma preocupação em auxiliar a execução automática do processo de software. Nesse contexto surge a assistência inteligente ao processo de software, que é o tema central deste artigo.

Um item de extrema importância na produção de um software é o método de desenvolvimento adotado. A maioria dos métodos existentes possuem imprecisões semânticas e heurísticas que necessitam ser formalizadas para que os engenheiros de software, mesmo os inexperientes, possam utilizá-los de uma forma mais eficiente e produtiva.

Podemos destacar duas abordagens para os métodos de desenvolvimento de software: a abordagem funcional e a orientada a objetos. Esta última destaca-se entre os desenvolvedores de software, por trazer uma série de vantagens, tais como: manutenibilidade, flexibilidade e abstração de dados.

Este artigo apresenta a formalização do processo de desenvolvimento do método ***fusion*** [COL94] baseada nas técnicas de planejamento propostas por Huff [HUF89].

Na seção 2 são apresentadas as características gerais da assistência inteligente proposta para o processo de software. Na seção 3 descrevemos o primeiro nível da biblioteca de operadores e os algoritmos da assistência inteligente. Na seção 4 apresentamos o protótipo desenvolvido para a assistência inteligente ao método ***fusion***. Na seção 5 apresentamos os trabalhos relacionados e na seção 6 as conclusões.

[1] Carnegie Mellon University, USA.

2. Planejamento Aplicado ao Processo de Software

No processo de software tradicional o profissional encarregado do desenvolvimento tem controle sobre o processo. A execução de qualquer atividade passa por sua intervenção. Sabe-se, no entanto, que existem atividades que podem ser automatizadas. Nesses casos, o gerenciamento do processo ainda pode ser mantido sob responsabilidade do profissional. Um desenvolvedor de software pode ser auxiliado de várias formas, dentre as quais destacamos:

- possibilitar a geração de agendas das atividades a serem realizadas;
- registrar a história das atividades realizadas;
- fornecer propostas em relação aos possíveis cursos de ações das atividades.

As técnicas de planejamento da inteligência artificial são aplicadas na solução de problemas complexos para os quais convêm subdividi-los em partes menores elaborando planos para obter a solução global [RIC88]. Essa flexibilidade na elaboração de planos e na obtenção de objetivos vai ao encontro dos requisitos inerentes ao processo de software. Nestes processos, a complexidade das atividades envolvidas pode ser diminuída através da divisão destas em partes menores.

As técnicas de planejamento podem ser aplicadas ao domínio específico do processo de software. Nesta abordagem visualiza-se o processo de software em termos dos objetivos e das ações que são necessárias para alcançar esses objetivos, conforme ilustrado na figura 1 [HUF89].

Figura 1. Uma Visão de Processo de Software.

Na figura 1 os Objetivos do desenvolvimento de software abrangem os aspectos funcionais. As Ações para alcançar estes objetivos consistem na maioria das vezes na invocação de ferramentas providas pelo ambiente de desenvolvimento. Exemplos dessas ferramentas são: editores, compiladores e ferramentas de análise e projeto. O Conhecimento do processo é capturado em operadores. Nestes operadores representamos cada atividade do processo de software como um objetivo a ser alcançado.

O conjunto dos operadores compõe a biblioteca de apoio à execução do processo. Dessa forma, toda e qualquer atividade envolvida no processo de software deve estar representada na biblioteca de operadores. Quaisquer alterações nas definições dos operadores afetará o processo a ser seguido.

[HUF89] propõe um formalismo para a especificação de operadores de processo de software, conforme ilustra a figura 2.

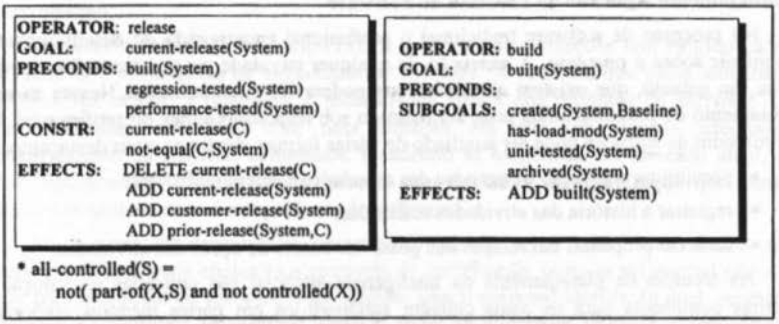

Figura 2. Exemplo de Operadores de Processo de Software.

As cláusulas definidas nesses operadores são:

- **Operador (Operator)**: nomenclatura que identifica o operador.
- **Objetivo (Goal)**: cláusula que define o objetivo de mais alto nível do operador.
- **Precondições (Preconds)**: definem o estado em que deve estar a execução do processo para que a ação representada pelo operador seja possível.
- **Restrições (Constr)**: restrições específicas à execução do operador representadas através de parâmetros.
- **Efeitos (Effects)**: mudanças no estado do processo resultantes da obtenção do objetivo de mais alto nível (cláusula objetivo).
- **Subobjetivos (Subgoals)**: decompõem o objetivo de mais alto nível em partes.

No trabalho realizado por Huff [HUF89] foram definidos os aspectos básicos referentes a assistência inteligente ao processo de software. O domínio de aplicação utilizado por Huff envolveu partes do processo de software restritas às atividades de compilar, ligar e testar uma nova versão de um sistema. Destacou-se nesse trabalho a necessidade da realização de estudos de processos de software reais, objetivando avaliar as técnicas de planejamento propostas de forma mais ampla.

3. Assistência Inteligente ao Processo de Desenvolvimento do Método *Fusion*

Adotamos como base para a aplicação das técnicas de planejamento o método orientado a objetos ***fusion*** [COL94]. Este método, como o próprio nome diz, é uma fusão dos vários métodos previamente existentes de orientação a objetos. O método ***fusion*** propõe uma abordagem bastante sistemática de desenvolvimento de software orientado a objetos, aproveitando de outros métodos algumas notações, modelos e abordagens, como por exemplo:

- do OMT de Rumbaugh [RUM91]: o modelo de objetos e o processo de desenvolvimento;
- de CRC (Class Responsibility Collaborator) [WIR90]: a interação entre os objetos;
- de BOOCH [BOO92]: a visibilidade entre os objetos;

- dos Métodos Formais: as pré e pós-condições .

O método ***fusion*** divide o processo de desenvolvimento de software em análise, projeto e implementação, fornecendo informações bastante precisas em relação ao que deve ser feito em cada fase. Além disso, fornece critérios que guiam o desenvolvedor na passagem de uma fase para outra no processo de desenvolvimento. Algumas vantagens do método são:

- as heurísticas de projeto e análise são explícitas, permitindo uma descrição precisa;
- existe um bom tratamento do mecanismo de herança;
- os pontos de verificações de consistência dos modelos são bem definidos;
- o processo que vai da análise ao projeto é claro e explícito.

Dessa forma, o processo de desenvolvimento do ***fusion*** possui as características adequadas para a aplicação das técnicas de assistência inteligente propostas neste artigo. O aspecto mais importante para o desenvolvimento de uma assistência inteligente a esse método é a definição de uma biblioteca de operadores que represente as atividades que podem ser realizadas no processo. No caso específico do método ***fusion***, os operadores descrevem as atividades de criação e verificação dos modelos das fases de análise e projeto.

3.1. Biblioteca de Operadores do Método *Fusion*

Existem basicamente sete modelos que são desenvolvidos durante as fases de análise e projeto do método ***fusion***: modelo de objetos, modelo de ciclo de vida, modelo de operações, grafos de interação de objetos, grafos de visibilidade, descrição das classes e grafos de herança. O desenvolvimento de cada um destes modelos é particionado (cláusula *subgoals*) em outras atividades, as quais por sua vez, originam outros operadores. Denominamos de primitivos os operadores que não contém subobjetivos e de complexos ou não primitivos aqueles que contém. O fato de se definir um operador como primitivo ou não, é de escolha exclusiva do especificador do processo e a decisão baseia-se na complexidade da atividade sendo representada.

A figura 3 representa os operadores pertencentes ao primeiro nível de abstração do processo de criação dos modelos do método ***fusion***. Cada operador foi especificado até o 3° ou 4° nível de abstração (de acordo com o operador), formando uma biblioteca com 74 operadores.

Os operadores definidos para o método ***fusion*** obedecem a uma ordem pré-definida de realização de atividades. A seqüência adotada tem como base a ordem de execução das atividades definida em Coleman [COL94]. No entanto, uma das vantagens da formalização das atividades do processo em operadores é permitir que mudanças na ordem de execução do processo sejam realizadas alterando-se apenas os elementos necessários no(s) operador(es). Como exemplo, podemos citar as mudanças que seriam necessárias para definir a seguinte ordem de criação de modelos: modelo de objetos, modelo de operações e modelo de ciclo de vida. Basicamente seria necessário mover a precondição *created_obj_mod* do operador ***create_lifcyc_mod*** para o operador ***create_oper_mod*** e eliminar a precondição *created_lifcyc_mod* do operador ***create_oper_mod***. Após isto deveríamos incluir a precondição *created_oper_mod* no operador ***create_lifcyc_mod***. Com essas mudanças determina-se que o modelo de operações deve ser criado antes do modelo de ciclo de vida e além disso, que o modelo de objetos deve ser criado antes do modelo de operações. As mudanças na ordem de realização das atividades do processo são de responsabilidade

exclusiva do especificador do processo e devem ter respaldo nas definições do processo de desenvolvimento sendo definido.

Figura 3. Operadores do primeiro nível de abstração do Processo de Desenvolvimento do Método ***Fusion***.

Destacamos que um dos fatores mais importantes na definição da biblioteca de operadores é possuir uma base de conhecimento sólida a respeito do método modelado. No nosso caso o método ***fusion***. Um requisito fundamental para isto é a experiência obtida com a utilização do método. Assim, o conhecimento do processo pode ser representado explicitamente, evitando-se omissões, as quais acarretam certamente deficiências no processo de aplicação do método.

3.2. Algoritmos da Assistência Inteligente

Definida a biblioteca de operadores para o método ***fusion***, os algoritmos da assistência inteligente denominados planejador e reconhecedor de planos interpretam as atividades do processo e propõem planos para o alcance dos objetivos.

A figura 4 representa, em forma de árvore, um exemplo do relacionamento existente entre precondições, subobjetivos e os operadores. O exemplo mostra as pecondições e subobjetivos do operador ***create_lifcyc_mod***. Precondições e subobjetivos relacionam-se com os operadores, estabelecendo os requisitos necessários à execução de cada operador. O relacionamento *achiever*, entre um operador e um subobjetivo ou precondição, denota que o efeito (cláusula *Effects*) do operador satisfaz o subobjetivo ou precondição associados.

Baseado nesses relacionamentos e no estado do processo, pode-se decidir a respeito da possibilidade de execução das atividades. Se o operador que representa a atividade é primitivo, verificam-se basicamente as precondições e restrições. Quando a atividade envolve operadores não primitivos, há necessidade de se verificar, além dos elementos verificados para os operadores primitivos, os subobjetivos da atividade.

O primeiro passo do **reconhecedor de planos,** quando o desenvolvedor deseja realizar uma atividade, é verificar se essa atividade já foi realizada. Isso é feito consultando-se a base de atividades já realizadas. Se a atividade não estiver presente na base, continua-se o processo de reconhecimento. Verificam-se então, as precondições e os subobjetivos do operador que representa a atividade. Se um destes elementos estiver pendente, um plano é montado para tentar realizar esses elementos e consequentemente possibilitar a realização da atividade. O plano é montado pelo algoritmo denominado **planejador** e contém todos os elementos necessários à execução da atividade. Após montado o plano, passa-se a verificar quais dos elementos que o compõe estão presentes na base de atividades já realizadas no processo.

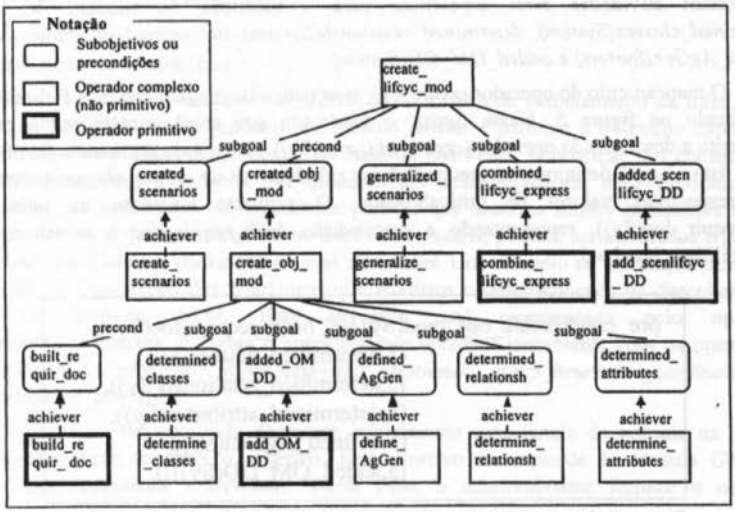

Figura 4. Uma visão de Operador do Método ***Fusion.***

A assistência inteligente permite que as atividades pendentes no plano sejam realizadas pelo desenvolvedor. É importante destacar que grande parte das atividades do processo são auxiliadas pela assistência inteligente de modo complementar, não sendo automatizadas completamente. Isso acontece porque na maioria das vezes são necessárias informações e decisões do desenvolvedor.

4. Prototipação da Assistência Inteligente ao Processo do Método *Fusion*

Com base na biblioteca de operadores e nos algoritmos planejador e reconhecedor de planos, apresentados na seção anterior, uma ferramenta de auxílio à execução do processo de software pode ser construída. Chamamos esta ferramenta de assistente inteligente ao processo de software do método ***fusion***. Através desta ferramenta o engenheiro de software será auxiliado no desenvolvimento dos modelos, aplicação de heurísticas e verificação de consistência do método ***fusion***.

A linguagem escolhida para o desenvolvimento do protótipo do assistente inteligente foi Prolog. A prototipação foi realizada no ambiente MS-Windows, utilizando o ambiente de programação LPA Win_Prolog [WES95]. Os operadores do processo de desenvolvimento foram mapeados para a linguagem Prolog juntamente com os algoritmos da assistência inteligente, conforme descritos nas seções seguintes.

4.1. Mapeamento dos Operadores do Método *Fusion*

Todos os operadores definidos para as fases de análise e projeto do método ***fusion*** são mapeados para a linguagem Prolog. Tomemos como exemplo o operador *create_obj_mod*, ilustrado na figura 3. Ele define os elementos necessários à execução da atividade que cria o modelo de objetos. Este possui como precondição a existência do documento de requisitos do usuário (*built_requir_doc*). A atividade é particionada em cinco subobjetivos, os quais representam atividades mais específicas para a obtenção do modelo de objetos: *determined_classes(System), determined_relationsh(System), determined_attributes (System), defined_AgGen(System)* e *added_OM_DD(System)*.

O mapeamento do operador *create_obj_mod* para a linguagem Prolog é feito conforme apresentado na figura 5. Nessa figura, o predicado **pre_cond** contém como primeiro argumento a descrição do operador (*create_obj_mod(Sy)*). O segundo argumento do predicado é uma lista na qual definimos as precondições e subobjetivos de *create_obj_mod*, bem como seus respectivos valores de criticabilidade. O primeiro elemento da lista é (6, built_requir_doc(Sy)), representando a precondição *built_requir_doc* e o seu valor de criticabilidade igual a 6.

```
pre_cond(create_obj_mod(Sy),[(6, built_requir_doc(Sy)),
                             (5,determined_classes(Sy)),
                             (5,determined_relationsh(Sy)),
                             (5,determined_attributes(Sy)),
                             (5,defined_AgGen(Sy)),
                             (5,added_OM_DD(Sy))]).
```

Figura 5. Representação em Prolog do Operador *create_obj_mod*.

O valor de criticabilidade representa de forma hierárquica a ordem de importância do elemento na execução do operador. Se observarmos a figura 3 veremos que a única precondição do operador *create_obj_mod* é *built_requir_doc* e os demais elementos envolvidos na implementação são os subobjetivos. Dessa forma, a criticabilidade maior está associada com a precondição *built_requir_doc*, pois para poder realizar qualquer subobjetivo é necessário antes satisfazer a precondição.

Os subobjetivos recebem um valor de criticabilidade de acordo com a sua importância na ordem de execução para a realização da atividade. Se todos os subobjetivos tiverem o mesmo nível de importância, a criticabilidade associada será a mesma. No exemplo da figura 5 todos os subobjetivos possuem a mesma criticabilidade. No entanto, a ordem dos subobjetivos na lista determina a sequência de execução.

4.2. Execução de Atividades do Processo de Desenvolvimento do Método *Fusion*

Quando executamos o protótipo da assistência inteligente ao método ***fusion***, partimos do estado definido do processo. Assume-se que no estado inicial, o documento de requisitos do usuário, representado pelo operador *build_requir_doc*, está sempre disponível. O estado do processo é atualizado com a execução das atividades.

As atividades das fases de análise e projeto do método ***fusion*** foram simuladas tendo como base o sistema ATM[†]. Este sistema permite que um cliente realize algumas transações bancárias automaticamente. O principal objetivo da simulação foi validar as técnicas propostas neste trabalho para a execução assistida das atividades do processo do método ***fusion***. As atividades do processo, tais como: a criação de modelos, atualização do dicionário de dados e a aplicação de heurísticas, foram simuladas tendo como produto resultante os modelos e as informações desenvolvidas para o sistema ATM.

Nas seções seguintes descrevemos o processo de aplicação de heurísticas e de verificação da consistência dos modelos.

4.2.1 Aplicação de Heurísticas

Uma das maiores vantagens da utilização da técnica de planejamento da inteligência artificial na formalização do processo do método ***fusion*** é permitir a definição explícita e sistemática das heurísticas do processo. Na maioria das vezes, heurísticas são consideradas como complementares e somente são aplicadas por desenvolvedores mais experientes. A formalização das heurísticas em operadores permite que usuários menos habituados com o método de desenvolvimento utilizado, no nosso caso ***fusion***, possam ser auxiliados e guiados no processo, facilitando a realização de suas atividades. Um exemplo da simulação deste tipo de atividade é a aplicação das heurísticas que auxiliam o desenvolvedor na descoberta das classes do sistema ATM. Esta atividade está representada pelo operador ***apply_heuristics_classes.*** A figura 6 apresenta este operador juntamente com os operadores de heurísticas do processo: *apply_heuristics_relationsh*, *apply_heuristics_cardinalities* e *apply_heuristics_attributes*.

O protótipo desenvolvido apresenta inicialmente uma janela de entrada na qual o desenvolvedor deve fornecer o objetivo. Este objetivo corresponde à cláusula *Goal* do operador que representa a atividade. Neste caso, o desenvolvedor fornece o objetivo

[†] Automatic Teller Machine

applied_heuristics_classes(atm), que corresponde a aplicar as heurísticas na descoberta de classes. É importante observar que a variável *System* presente no operador *apply_heuristics_classes,* é instanciada com o termo *atm,* declarando dessa forma, que o sistema ATM está sendo desenvolvido. O protótipo consulta a base de atividades realizadas, procurando pela atividade *apply_heuristics_classes(atm)*. Caso ainda não tenha sido realizada, inicia-se o processo para a sua realização. Neste caso, a única restrição à execução da atividade é a precondição *built_requir_doc(atm)* (ver figura 6), que é satisfeita pelo estado inicial do processo.

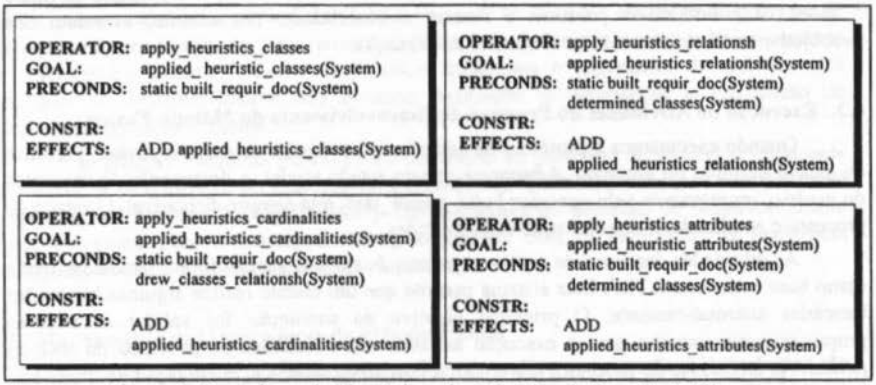

Figura 6. Operadores de Heurísticas dos Modelos da Fase de Análise.

Dessa forma, o desenvolvedor pode aplicar as heurísticas na descoberta de classes. Antes de permitir a realização da atividade, o protótipo da assistência inteligente solicita a confirmação da realização da atividade. O protótipo exibe para o desenvolvedor, em janelas de edição, o documento de requisitos do usuário para o sistema ATM e as heurísticas que auxiliam o desenvolvedor na descoberta das possíveis classes do sistema. Através de uma opção identificada por **Aplicar Heurísticas,** o desenvolvedor pode interagir com a assistência inteligente, acrescentando em outra janela as possíveis classes para o sistema ATM. Isto pode ser visto na figura 7, gerada pelo protótipo desenvolvido.

Verificamos neste contexto que é possível estender a formalização e aplicação das heurísticas definidas para o método ***fusion*** para outros métodos. Algumas heurísticas são comuns à maioria dos métodos orientados a objetos utilizados atualmente. Um exemplo da possibilidade de reaproveitamento de operadores encontra-se entre o método OMT [RUM91] e o ***Fusion***, especificamente no modelo de objetos. A maioria dos operadores de heurísticas utilizados na criação do modelo de objetos do ***fusion***, poderia ser utilizada no OMT. O auxílio à descoberta de classes, atributos de classes e estabelecimento de estruturas de heranças são atividades comuns que podem ser auxiliadas pelas mesmas heurísticas em ambos os métodos.

4.2.2. Verificação de Consistência dos Modelos

O processo do método ***fusion*** inclui um conjunto de verificações de consistência dos modelos construídos. Estas verificações são formalizadas através de operadores, garantindo assim a sua execução como atividades incluídas do processo.

Algumas verificações são aplicadas no final de uma fase específica do processo, procurando verificar inconsistências entre os modelos construídos. Outras são aplicadas a um modelo específico, verificando aspectos internos do modelo ou comparando-o com outros já criados. Como exemplo, podemos citar a fase de análise. Nesta fase são criados: o modelo de objetos, o modelo de ciclo de vida, o modelo de operações e os cenários. Cenários são utilizados como informações auxiliares para a criação dos modelos.

Quando é encerrada a fase de análise, o operador *check_analysis_mod*, que representa as verificações a serem efetuadas no final da fase de análise, deve ser executado. Estas verificações estão incluídas como último subobjetivo do operador *create_oper_mod*, o qual representa o modelo de operações (ver figura 3). Essa atividade é considerada obrigatória para a conclusão da fase de análise.

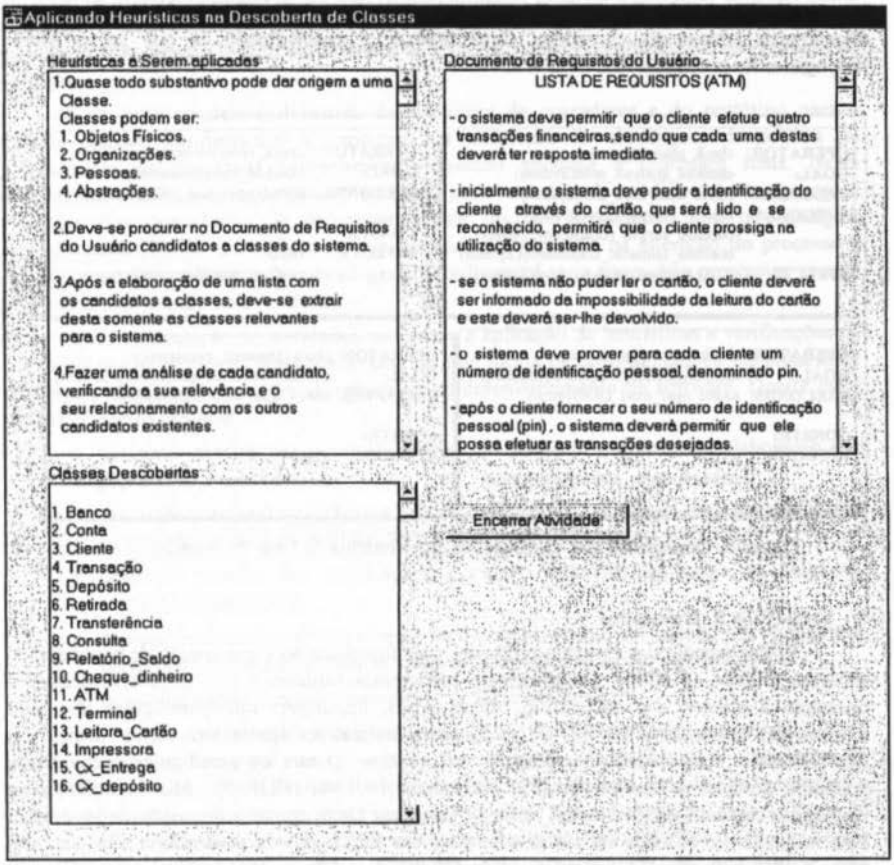

Figura 7. Aplicando heurísticas na descoberta de classes.

A figura 8 apresenta o operador *check_analisys_mod* e seus subobjetivos *check_completeness, check_simple_consistency* e *check_semantic_consistency*. Aplicar as verificações implica neste caso em verificar a completude em relação aos requisitos; verificar a consistência simples, a qual representa a sobreposição entre os modelos de análise; e verificar a consistência semântica, a qual tenta garantir que as implicações dos modelos sejam consistentes. O processo para realizar verificações é análogo ao descrito na seção anterior para aplicar as heurísticas na descoberta das classes do sistema ATM. O desenvolvedor deve entrar com o objetivo *checked_analysis_mod(atm)* na janela de entrada do protótipo. A precondição para a realização das verificações nos modelos da fase de análise é que as informações do modelo de operações tenham sido incluídas no dicionário de dados. Supondo que essa atividade já tenha sido realizada, o desenvolvedor pode proceder a verificação, guiado pelo protótipo. Permite-se que o desenvolvedor consulte os modelos sendo verificados, juntamente com as informações necessárias às verificações. Com estas verificações, os problemas encontrados podem ser corrigidos, diminuindo-se a possibilidade de que falhas sejam propagadas no processo.

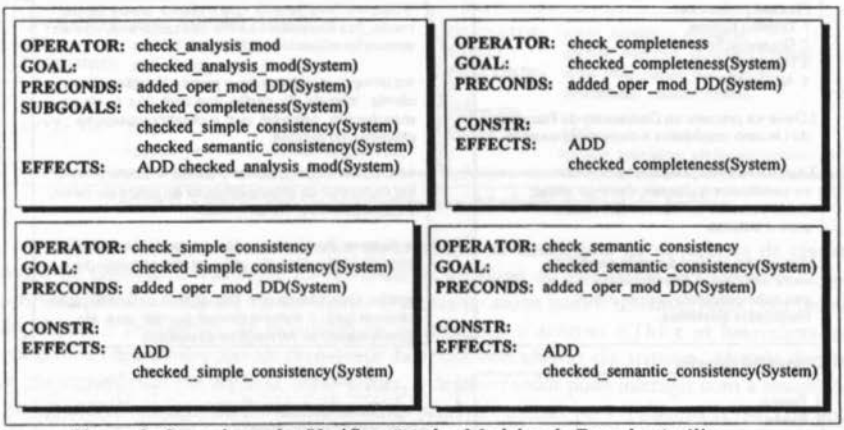

Figura 8. Operadores das Verificações dos Modelos da Fase de Análise.

5. Trabalhos Relacionados

Vários paradigmas têm sido propostos para formalização e automação de processos de software [FIN94, FUG96]. Os principais paradigmas utilizados são: o procedimental, o orientado a objetos e o declarativo. Além destes, linguagens multiparadigmas têm sido amplamente utilizadas [CON94]. Os paradigmas declarativos apresentam como vantagens a flexibilidade e simplicidade na alteração do processo. Dentre os paradigmas declarativos, podemos destacar o Marvel [KAI90] e Grapple [HUF88] [HUF89] Marvel utiliza uma linguagem chamada MSL (Marvel Strategy Language) para apoiar a descrição de processos. Esta linguagem é baseada em idéias adaptadas das linguagens da inteligência artificial, tais como OPS5 e Prolog. O projeto Grapple [HUF89] propõe, como descrito neste artigo, a formalização do processo baseada nas técnicas do planejamento.

Porém, estes paradigmas têm sido aplicados a pequenos exemplos, tais como processos de edição, compilação e alteração, ou mesmo o exemplo clássico do processo *change request* [KEL90].

Este artigo apresenta uma evolução dos trabalhos desenvolvidos por Huff, aplicando as técnicas propostas a um processo real. Esta aplicação permitiu a avaliação da complexidade da definição de atividades em operadores, da formalização de heurísticas e de procedimentos de verificações.

6. Conclusões

Através da formalização do processo de software utilizando técnicas de planejamento da inteligência artificial, pudemos descrever de forma bastante clara e precisa os elementos do processo de desenvolvimento do método ***fusion***, tornando possível a execução assistida deste. Capturar o conhecimento do processo em operadores ainda é uma tarefa difícil, principalmente quando se tem um domínio de aplicação com a complexidade do processo de software.

Com base no desenvolvimento da biblioteca de operadores e do protótipo para o método ***fusion***, confirmamos a vantagem da utilização da abordagem declarativa de programação de processos (planejamento), quando aplicada a processos reais. Nesta abordagem, a ordem de execução do processo pode ser alterada modificando-se apenas as cláusulas dos operadores. Se comparamos este tipo de abordagem com o paradigma procedimental [SUT89], verificamos que o trabalho dispendido na alteração do processo é consideravelmente reduzido. Na abordagem procedimental seria necessário prescrever grande parte do processo para cada mudança.

Com a simulação de atividades, tais como a aplicação de heurísticas e verificações de consistência dos modelos, pudemos avaliar de que forma os profissionais da engenharia de software podem melhorar o seu desempenho no desenvolvimento de software. Dentre estas formas podemos destacar:

- descrição precisa das possíveis ordens das atividades, através da elaboração dos planos;
- descrição precisa dos modelos a serem construídos no processo de desenvolvimento do método, através da inclusão desses elementos nos planos;
- descrição precisa das verificações a serem realizadas ao longo do processo, evitando-se omissões;
- captura das heurísticas do processo de desenvolvimento do método em operadores, permitindo que mesmo desenvolvedores inexperientes façam uso destas;
- possibilidade de execução automática de algumas tarefas.

Em relação às avaliações apresentadas por Huff [HUF89] podemos acrescentar alguns aspectos críticos inerentes ao domínio de aplicação envolvendo o processo de desenvolvimento do método ***fusion***. Uma das maiores dificuldades é conciliar as exigências e as limitações de desenvolvedores. Uma pergunta que surge neste contexto é "qual é o nível de liberdade apropriado que deve-se atribuir ao desenvolvedor na realização das atividades de um método específico?". Esta decisão é tomada pelo especificador do processo e não necessariamente vai satisfazer a todos os desenvolvedores.

Outra questão importante é o particionamento das atividades do processo de desenvolvimento. A maioria das atividades pode ser particionada em subobjetivos. Para o método ***fusion*** as atividades foram particionadas tomando como base as definições encontradas em Coleman [COL94]. Até que ponto deve-se particionar uma atividade e representar este particionamento em operadores é um aspecto ainda bastante subjetivo. Isto depende em grande parte da experiência do especificador do processo.

Em Huff [HUF89], a interface da assistência inteligente não foi tratada em detalhes, principalmente pelo fato do domínio de aplicação escolhido não ter requerido uma interface aprimorada. Já no protótipo, aqui apresentado, observamos quão importante é a construção de uma interface que suporte o nível de interação exigido entre o desenvolvedor e a assistência inteligente. Dessa análise, verificamos a necessidade de melhorar a interface do protótipo desenvolvido.

Estes pontos incentivam-nos a realizar trabalhos futuros visando melhorar e aprimorar a assistência inteligente ao processo de software e de modo particular o protótipo desenvolvido.

7. Referências Bibliográficas

[BOO92] BOOCH,G., *The Booch Method: Process and Pragmatics*, Santa Clara, Calif.: Rational, (1992).

[CHR95] CHRISTIE,A.M., *Software Process Automation*, Springer-Verlag Berlin Heidelberg, (1995).

[COL94] COLEMAN, D., P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes, *Object-oriented development, The Fusion Method*, Prentice-Hall, (1994).

[CON94] CONRADI,R., et all, "EPOS: Object-Oriented Cooperative Process Modelling", in FINKELSTEIN, A,. J.Krammer, B. Nuseibeth (eds.), *Software Process Modelling and Technology*, Jonh Wiley and Sons Inc., (1994).

[FIN94] FINKELSTEIN, A,. J.Krammer, B. Nuseibeth (eds.), *Software Process Modelling and Technology*, Jonh Wiley and Sons Inc., (1994).

[FUG96] FUGGETA, A,. A. Wolf, *Trends In Software Process*, John Wiley & Sons, (1996).

[HER96] HERBSLEB, J.D., D. R. Goldenson, "A Systematic Survey of CMM Experience and Results", *Proceedings of ICSE-18, IEEE Computer Society,* March 25-29, (1996).

[HUF88] HUFF, K.E., e V.R. Lesser, "A Plan-Based Intelligent Assistant that Supports the Software Development Process", Peter Henderson(editor), *ACM SIGSoft/SIGPLAN Software Engineering Symposium on Practical Software Development Environment*, pages 1-13, ACM Press, Boston MA, November, 1988.

[HUF89] HUFF, K.E., "Plan-Based Intelligent Assistance: An Approach to Supporting the Software Development Process" , *Ph.d Thesis, Dept. of Computer and Information Science,* The University of Massachusets , September(1989).

[KAI90] KAISER, G.E., N.S. Barghouti e M.H. Sokolski, "Preliminary Experience with Process Modelling in the Marvel Software Development Environment kernel", In Bruce D.Shuver (editor), *23rd Annual Hawaii International Conference on System Sciences*, Volume II, pages 131-140, January, (1990).

[KEL90] KELLNER, M.I., et al, "Software Process Modeling Example Problem", *Proceedings of 6th International Software Process Workshop: Support for the Software Process*, Held at Hakodate, Hokkaido, Japan, October 28-31,(1990).

[RIC88] RICH, E., *Inteligencia Artificial*, São Paulo, Mc-Graw-Hill, (1988).

[RUM91] RUMBAUGH, J., et al, *Object-Oriented Modelling and Design*, Englewood Cliffs, New Jersey, Prentice Hall, (1991).

[SUT89] SUTTON, S.M., Jr, D. Heimbigner, L.J. Osterweil. "APPL/A : A Prototype Language for Software Process programming", *Departament of Computer Science, University of Colorado*, October, (1989).

[WES95] WESTWOOD, David, *LPA-PROLOG Thecnical Reference*, Logic Programming Associates Ltd, (1995).

[WIR90] WIRFS-BROCK, R., B.Wilkerson, L. Wiener, *Designing Object-Oriented Software*, Prentice Hall International, Englewood Cliffs, NJ, (1990).