Aspectos de Sincronização em Modelos UML

Marco Costa
Universidade Lusíada, Lisboa, Portugal
mbcc@acm.org

Alberto Rodrigues da Silva
Instituto Superior Técnico, Lisboa, Portugal
alberto.silva@acm.org

Resumo

Existem já diversas ferramentas CASE que permitem a geração de código a partir de modelos UML. Para além desta capacidade outras ferramentas conseguem ainda produzir modelos a partir de código, ou mesmo fazer a geração *round-trip* que consiste em manter algumas referências entre os dois tipos de artefactos. No entanto esta ligação entre os modelos e o código é apenas uma ponta num *iceberg* muito superior: o da sincronização entre artefactos da mesma ou de diferentes linguagens. Apresenta-se neste artigo algumas características de um protótipo em fase de implementação que aborda esta temática.

Palavras-chave: UML, MDA, QVT, XML

1. Introdução

Os sistemas de informação têm sofrido alterações tecnológicas e metodológicas profundas ao longo dos últimos cinquenta anos. As diversas evoluções de hardware incrementaram em diversas ordens de grandeza o desempenho e a memória dos sistemas informáticos permitindo novas evoluções no campo do software. Por outro lado, o surgimento de novos paradigmas de programação e modelação levaram a alterações metodológicas importantes no próprio processo de desenvolvimento.

À medida que os sistemas de informação se foram tornando mais complexos, surgiu a necessidade de se encontrar formas de os desenvolver sistematicamente. O aparecimento das metodologias de desenvolvimento de sistemas de informação tornou estes mais simples de produzir e manter, tal como reduziu os custos a médio e longo prazo melhorando a sua qualidade. Assim, as tarefas que anteriormente se realizavam *ad hoc* passaram a integrar um processo de desenvolvimento. As vantagens da utilização de muitas destas metodologias foram sendo reconhecidas ao longo do tempo, embora por vezes com alguma dificuldade, pela indústria.

A par da criação de novas metodologias foram concebidas ferramentas automatizadas de apoio à modelação de sistemas, denominadas CASE (Computer Aided Systems Engineering). Durante a década de 1980 surgiram diversos produtos no mercado que após um sucesso inicial acabaram por não ter grande aceitação a longo prazo. O hardware parecia não acompanhar o desenvolvimento do software tornando estas aplicações muito lentas e difíceis de utilizar. Actualmente as ferramentas CASE atingiram já a fase em que a sua utilização passou a ser reconhecida como importante, mas ainda não como essencial. Encontram-se num estádio que permite já ao utilizador passar à fase de geração automática de código [Herrington 2003] a partir de modelos, embora esta característica ainda não esteja vulgarizada.

Segundo um inquérito recentemente realizado [Welsh 2003] a 183 empresas, na área do desenvolvimento de sistemas de informação, mais de metade (55%) utilizam ferramentas de modelação. Por outro lado, menos de um terço (31%) usam capacidades de geração automática

In *Actas da 6ª Conferência da Associação Portuguesa de Sistemas de Informação.*

© APSI 2005.

(que se limita normalmente à geração da estrutura das classes ou dos esquemas de bases de dados relacionais). Importa tentar entender as razões pelas quais algumas empresas desta área ainda não consideram ser importante a utilização de ferramentas que aparentemente as deveriam tornar mais produtivas.

Actualmente, algumas das tecnologias envolvidas no processo de desenvolvimento chegaram já a um ponto de razoável maturidade. São exemplos os sistemas de bases de dados relacionais, as interfaces gráficas, os compiladores, todos eles com mais de 30 anos de desenvolvimento. No entanto a Engenharia Informática ainda está longe de atingir a evolução relativa das Engenharias Civil e Mecânica que têm uma história mais alargada. Não é de estranhar, por isso, que o erro, em maior ou menor escala, esteja presente em grande parte dos projectos informáticos actuais. Este facto leva a que devam ser utilizadas técnicas propiciadoras de uma acção correcta naquilo que diz respeito a todas as condicionantes de um projecto de sistemas de informação (p.ex. tempo, custo, etc.) de forma a conseguir resultados satisfatórios.

Neste trabalho são abordadas técnicas de Engenharia mas são necessários diversos tipos de técnicas igualmente importantes para que um projecto possa ser considerado bem sucedido. Dificuldades em áreas como a Gestão de Recursos Humanos, o *Marketing* ou a Gestão Financeira, exteriores ao âmbito deste trabalho, podem ser responsáveis, directa ou indirectamente, pelo insucesso dos projectos. Por exemplo, uma gestão financeira errada pode levar a que o projecto esgote o orçamento antes do previsto, tal como uma campanha de *marketing* errada pode afastar o mercado esperado para um certo produto, levando a que este não seja vendido suficientemente e terminando o seu ciclo de vida antes do tempo esperado. Por isso não são apenas necessárias técnicas de Engenharia para o processo de desenvolvimento dos sistemas de informação, tal como não o são para qualquer outro tipo de área ou produto. Pode-se porém indicar, de uma forma genérica, algumas características comuns às melhores técnicas de Engenharia [Berard 2005]: (1) Podem ser descritas quantitativamente, tal como qualitativamente; (2) Podem ser usadas repetidamente, com resultados semelhantes; (3) Podem ser ensinadas num período razoável; (4) Podem ser aplicadas por outras pessoas com um nível de sucesso aceitável; (5) Atingem melhores resultados que outras técnicas, de forma consistente e significativa, ou que uma abordagem *ad hoc* e (6) São aplicáveis numa percentagem grande de casos.

Desta lista de requisitos facilmente se observa que grande parte das práticas que normalmente associamos à actividade de desenvolvimento de sistemas de informação estão mais na categoria da Arte do que da Técnica. Este facto é mais evidente nas actividades que envolvem níveis elevados de abstracção, como é o caso da análise e concepção de sistemas de informação, ou da captura de requisitos. Apesar de serem usadas linguagens e ferramentas que apoiam estas actividades, tornando-as mais objectivas os resultados estão ainda muito aquém daquilo que seria de esperar de uma actividade de Engenharia.

2. Model Driven Architecture

Actualmente o UML *(Unified Modelling Language)* é reconhecidamente a linguagem padrão na modelação de sistemas de informação. Este facto tem levado a um grande desenvolvimento de técnicas e ferramentas que gravitam em torno dos padrões estabelecidos pela OMG *(Object Management Group)*. Com o aparecimento de um conjunto de standards tais como CWM *(Common Warehousing Model)*, EDOC (*Enterprise Distributed Object Computing*), MOF *(Meta-Object Facility)*, etc., que cobrem áreas diferentes do desenvolvimento de sistemas de informação deu-se a essa família de standards o nome de MDA *(Model Driven Architecture).* Assim, o UML é um membro da família MDA, embora talvez o mais importante.

Segundo a abordagem MDA existem três tipos de modelos que representam o sistema de informação (Figura 1). O modelo independente da computação (CIM, *Computation Independent*

Model), já conhecido como modelo de domínio, representa o sistema sem existir qualquer menção à tecnologia envolvida na implementação ou às linguagens de programação utilizadas. Este trabalho deve ser realizado por alguém que conhece profundamente o domínio em causa sendo o resultado um conjunto de diagramas de alto nível em UML. O modelo independente da plataforma (PIM, *Platform Independent Model*) representa um modelo conceptual do sistema onde existem todas as suas funcionalidades mas ainda não foram tomadas quaisquer opções tecnológicas. Este modelo representa o sistema nas suas componentes funcionais e estruturais, detalhadamente. O modelo específico à plataforma (PSM, *Platform Specific Model*) representa a implementação do sistema para uma dada plataforma, incluindo todas as decisões tecnológicas inerentes.

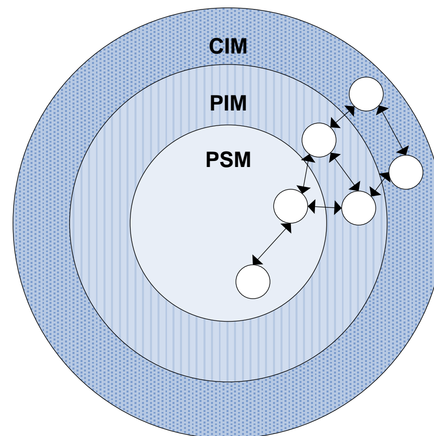

Figura 1 – Representação das três camadas do MDA e dos mapeamentos possíveis entre elementos destas camadas

Apesar de existirem três camadas, estas relaccionam-se através de possíveis mapeamentos entre os seus elementos. Cada camada pode incluir um conjunto de artefactos (diagramas, descrições, etc.) que representa os conceitos dentro dessa camada. A separação entre estes três níveis conceptuais leva à dificuldade de manter a coerência dos artefactos produzidos. Dois artefactos de níveis diferentes podem estar relaccionados (p.ex., uma classe UML "Pessoa" do PSM, existente numa ferramenta CASE, pode estar relaccionada com uma classe C# "Pessoa", implementada num ficheiro "Pessoa.cs"). Usualmente as acções sobre estes dois artefactos são realizadas em fases distintas do processo de desenvolvimento, por vezes até por equipas diferentes. Se as decisões numa das fases implicarem alterações ao artefacto da outra camada e se estas alterações não forem feitas, a coerência entre os dois elementos do modelo foi perdida.

O problema da coerência foi, quanto a nós, um dos factores mais importantes para a perda de confiança na utilização das ferramentas CASE no passado. Como a produção dos artefactos não está sincronizada, as acções realizadas sobre um modelo não têm reflexo imediato nos outros modelos que dele dependem. É necessário um conjunto de passos posteriores para manter a coerência entre os diversos modelos. Como estas actividades são muitas vezes feitas manualmente, com o apoio ocasional de alguma ferramenta, estão sujeitas a erro ou mesmo a não serem realizadas de todo. Por isso não é de estranhar que num sistema já implementado a documentação técnica do sistema esteja normalmente desactualizada. Aquilo que deveria ser, em nosso entender, um reflexo do sistema, surge como uma imagem desfocada, à qual se deixa de atribuir grande importância. Este é o primeiro passo para que os métodos de Engenharia sejam afastados em detrimento dos processos *ad hoc*.

Está em discussão no seio da OMG um conjunto de propostas para produzir um standard para a forma como as relações entre os modelos (tal como entre os artefactos associados) podem ser explicitadas. Este conjunto de propostas que definirá o QVT (*Query-View-Transformation*)

[QVTP 2003] deverá resolver o problema anterior fazendo com que de forma automática (ou pelo menos mais automática como veremos) se consigam manter este tipo de relações.

3. O Modelo QVT

O QVT representa uma tentativa de se encontrar uma linguagem standard para expressar transformações entre modelos, de uma forma simples e tendo por fim a sua execução automática [Tratt 2003]. A sigla QVT representa três conceitos: Pesquisa (*Query*), Vista (*View*) e Transformação (*Transformation*). Nas pesquisas existindo um modelo como input pode-se seleccionar um subconjunto dos seus elementos através de uma dada condição. Uma vista é um modelo derivado de outros modelos. As transformações são usadas para a partir de um modelo de input provocar-lhe uma alteração ou criar um novo modelo a partir deste. Apesar do QVT estar dependente destes três conceitos, o da Transformação é o mais importante.

A definição das transformações deve ter algumas características que são importantes para a futura aceitação do standard. Os artefactos gerados (diagramas, código, descrições, etc.) devem ser legíveis, tanto quanto possível, i.e. as transformações devem usar uma notação simples de escrever e de ler, tanto no que diz respeito ao texto como aos diagramas produzidos.

Algumas características da modelação orientada por objectos como a herança, a composição e a agregação devem ser possíveis de utilizar dentro das próprias definições do QVT. Deve ser possível realizar uma composição de duas ou mais transformações de forma a produzir-se uma transformação mais complexa. As transformações podem igualmente existir entre diversos níveis de abstracção tal como podem existir diversos níveis de abstracção para uma transformação. Assim sendo é necessário que exista um mecanismo de generalizar e especificar transformações. Deve também ser permitido reutilizar transformações, seja no mesmo projecto ou noutros.

4. Transformações, Relações e Mapeamentos

Uma **transformação** entre dois domínios A e B pode ser decomposta em relações e mapeamentos[1] conforme sugerido na Figura 2. As **relações** são especificações de transformações multi-direccionais entre dois ou mais modelos, sendo usadas, p.ex., para analisar a consistência dos modelos.

As relações não podem criar, eliminar ou modificar um modelo ou as suas partes. Inversamente os **mapeamentos** são implementações das transformações, potencialmente unidireccionais, podendo devolver valores.

Uma relação deve ser uma forma ainda abstracta de definir a respectiva transformação (Figura 3). Neste modelo, tanto quanto possível, não devem ser tomadas decisões tecnológicas. Estas devem pertencer aos diversos mapeamentos que são produzidos.

---

[1] Embora o termo matemático usual para “mapeamento” seja “aplicação” optou-se pelo primeiro por existir uma utilização frequente, no domínio da informática, do segundo com um significado totalmente distinto.

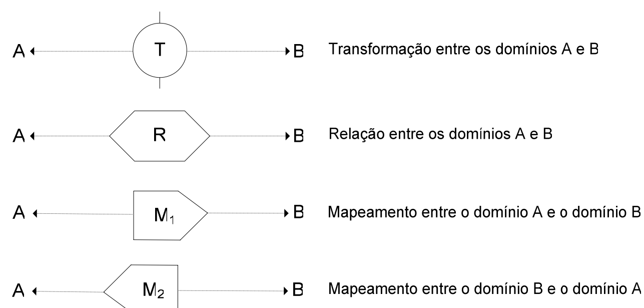

Figura 2 – Notação gráfica das Transformações, Relações e Mapeamentos em diagramas QVT

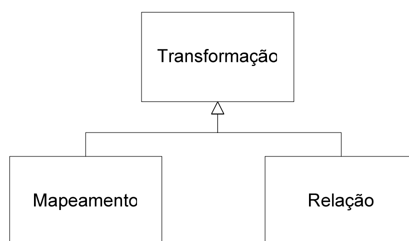

Figura 3 – Metamodelo com Transformações, Mapeamentos e Relações

Ao nível do metamodelo a metaclasse Transformação é derivada da metaclasse Classe o que faz com que possa ter atributos, herança, associações, etc, como qualquer outra classe UML. A semântica para estes conceitos encontra-se ainda em definição.

5. Implementação de uma Aplicação de Sincronização

Actualmente começam a surgir no mercado algumas ferramentas (e.g. GMT [Bettin et al. 2003], UMT-QVT [UMT-QVT 2005]) que propõem mecanismos de transformação que referem de alguma forma o QVT. No entanto estas capacidades ainda estão muito pouco desenvolvidas, sendo as soluções razoavelmente diferentes de fabricante para fabricante. O facto de o QVT ainda não ser um standard, mas sim um conjunto de propostas, não contribui para a convergência destas ferramentas. Para além de uma ausência de normas comuns (e.g. a notação gráfica utilizada ou a linguagem textual para representar as transformações) estes produtos têm ainda aspectos frequentes em aplicações num estádio de desenvolvimento muito inicial (e.g. instalação difícil e demorada, parametrização complexa, reduzidas interfaces com mais do que uma plataforma).

Tendo em atenção a necessidade já referida de constituir uma aplicação deste tipo foi iniciada a implementação de um protótipo denominado UML Mapping Architect. Pretende-se que esta aplicação demonstre os benefícios que a sincronização poderá trazer ao processo de desenvolvimento de sistemas de informação. A aplicação integra diversas áreas:

Modelador - Através de uma interface gráfica a ferramenta permite constituir um conjunto de diagramas UML padrão (e.g. diagrama de classes na Figura 4) e diagramas QVT. Os **diagramas QVT** são extensões dos diagramas de classes e utilizam uma notação própria. Estes diagramas permitem representar graficamente, conforme anteriormente mostrado, transformações, relações e mapeamentos entre diferentes conceitos. O tratamento deste aspecto será melhorado nas próximas versões do programa encontrando-se ainda num estado inicial.

Gerador – A aplicação gera código a partir dos modelos criados com o módulo *Modelador*, numa ou mais linguagens de programação (e.g. C#). Através de *templates* de geração automática são criados ficheiros com código. A geração automática não se limita à geração de código a partir de modelos – na verdade grande parte da interface gráfica (mais de 50% da

aplicação) é gerada em tempo de execução através de um ficheiro XML que pode facilmente ser editado para satisfazer as necessidades dos utilizadores. Para tal utilizaram-se as capacidades de reflexão da linguagem C# [Dollard 2004].

Sincronizador – Através de um conjunto de regras de sincronização são mantidas as relações entre o código e os modelos de forma a que a informação dos modelos não esteja desactualizada.

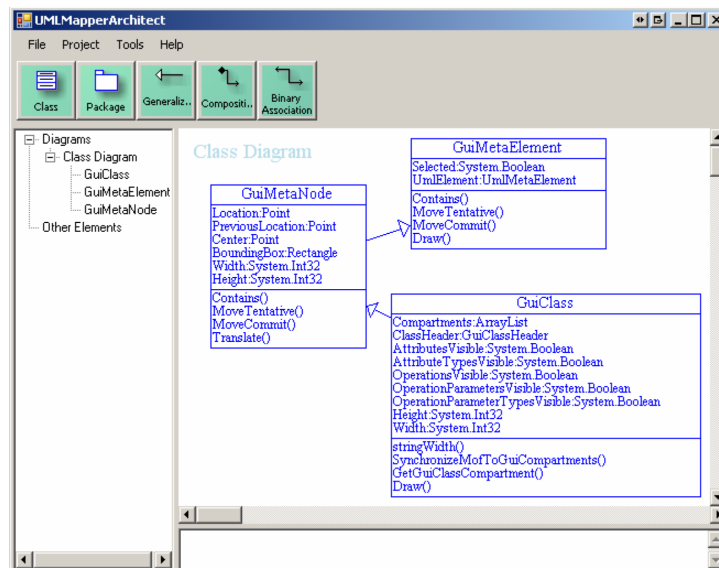

Figura 4 – Ecrã de modelação

Sistema de regras – O sistema de regras permite, a partir de um conjunto de regras e de uma tabela de relações, tomar decisões e desencadear acções segundo determinadas condições. Este sistema é aberto ao utilizador, sendo expandido à medida que a sua utilização aumenta.

Métricas – A aplicação permite realizar contagens automáticas dos elementos do sistema definidos num metamodelo guardado em XML.

Descreve-se neste artigo com mais detalhe o módulo S*incronizador* pela sua importância no contexto da sincronização de modelos UML.

6. Arquitectura do módulo *Sincronizador*

Num contexto em que existem artefactos de diversos tipos (e.g. ficheiros de código fonte, ficheiros com diagramas dos diversos modelos associados a uma ou mais aplicações) parece-nos importante assegurarmos que cada um destes artefactos não entra em contradição com algum dos restantes. Este tipo de verificação, denominada de **análise de coerência**, implica que existam mecanismos para validar as alterações provocadas no sistema, seja por um utilizador, seja por outras ferramentas de forma automática.

Tradicionalmente esta actividade é feita quase exclusivamente de forma manual. Existem porém algumas ferramentas que tentam auxiliar em tarefas muito específicas como a geração de código a partir de modelos de alto nível e a tradução de código entre linguagens. Esta abordagem apresenta a dificuldade de se desenrolar em momentos diversos sendo que, para sistemas de razoável dimensão, em fase de desenvolvimento, é praticamente impossível considerar a transição entre estados como atómica, i.e. a alteração de um artefacto pode simultaneamente coincidir com outra alteração num artefacto relacionado. Nesse caso não se consegue ter a

certeza de que o sistema está sincronizado a qualquer momento. Na verdade, quando estas actividades são feitas com predominância de processos manuais ou através de sequências de processos semi-automatizados mas desencadeados de forma manual, pode ser difícil conseguir obter nalgum momento uma imagem totalmente sincronizada do sistema.

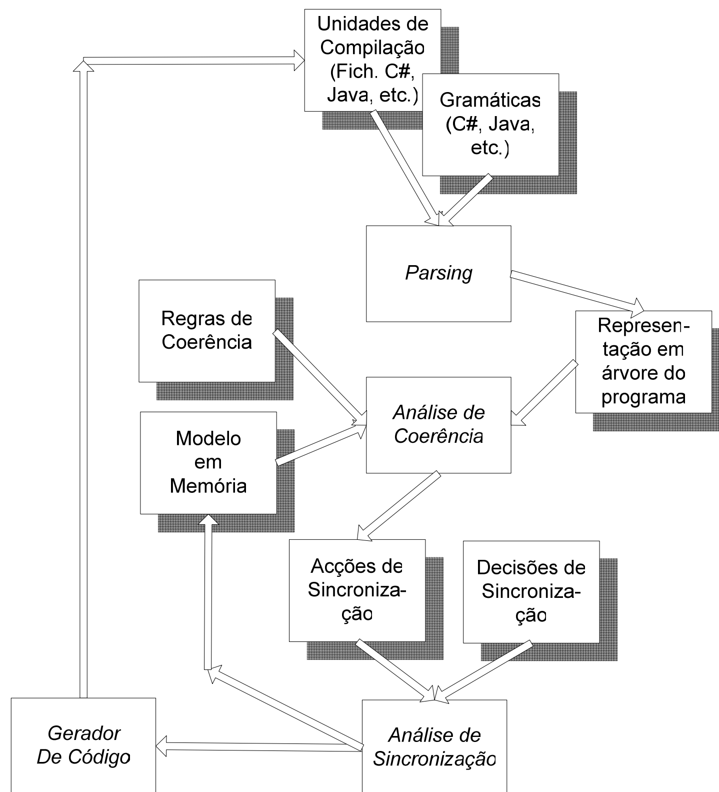

Figura 5 – Arquitectura do módulo Sincronizador

A sincronização dos diversos elementos dos modelos obedece aos seguintes tipos:

- Intermodelos: Dois elementos de modelos diferentes podem ser dependentes entre si. Não se considera aqui a representação duplicada do mesmo conceito em diagramas distintos que é tratada trivialmente pelas ferramentas CASE. Estas dependências podem ser apenas semânticas, sem que exista qualquer relação sintáctica entre os dois conceitos. Nestes casos a ferramenta necessita de intervenção humana para conseguir determinar este tipo de dependências.

- Intramodelos: Dentro de um modelo podem existir relações de dependência entre diferentes elementos do modelo (e.g. no diagrama de sequência uma mensagem para um objecto deve corresponder à definição de uma operação na classe correspondente do diagrama de classes). Esta dependência pode ser verificada automaticamente sem grande dificuldade mas podem existir também outras dependências definidas manualmente (e.g. um parâmetro de entrada de um método de uma classe pode estar dependente do nome de uma classe do mesmo modelo).

- Código-Modelo: Um elemento do modelo pode estar dependente de um elemento do código e vice versa. Neste tipo de dependência não basta tratar a representação do modelo é necessário também que exista uma representação em memória do próprio código. Isso pode ser conseguido recorrendo-se a um *parser* que a partir de uma gramática da linguagem e do ficheiro fonte produz uma representação em objectos dos elementos sinctáticos existentes no código. Estes elementos do código podem então ser mapeados em elementos do modelo recorrendo-se a um conjunto de regras de coerência já definidas. Estas regras podem ser instanciações de outras

mais genéricas ou podem ser definidas caso a caso. Este tipo de dependência poderia ser considerado um caso particular das dependências intermodelos, se não se distinguisse o código dos modelos.

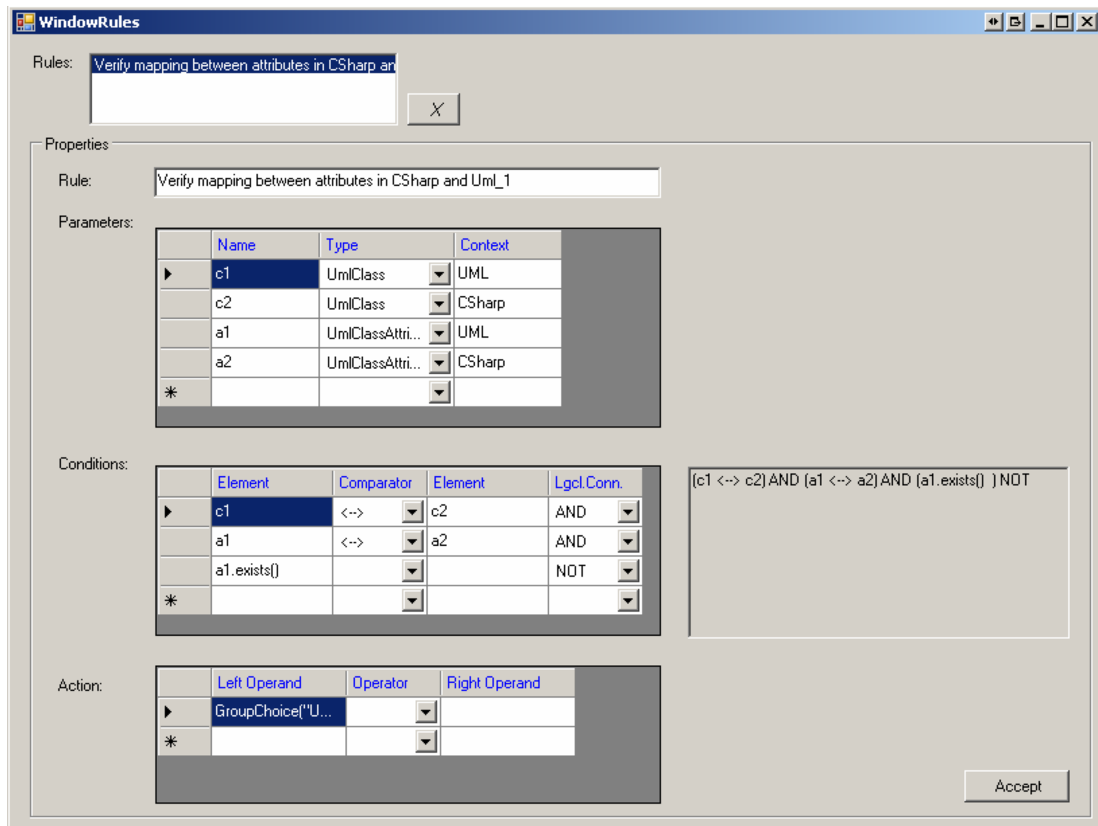

Figura 6 – Ecrã da aplicação que permite introduzir regras de coerência entre elementos dos modelos

- Código-Código: Este tipo de relações pode ser transformado em duas relações do tipo anterior facilitando-se a sua implementação. Nesse caso guardam-se duas relações Código-Modelo, supondo um mesmo modelo (PSM).

Na Figura 5 apresenta-se a arquitectura do módulo Sincronizador colocando em relevo os processos importantes (rectângulos), os artefactos produzidos por estes (rectângulos sombreados) e a sua ordem de execução (arestas).

O sincronizador tem também a capacidade de a partir das alterações do modelo verificar o estado da sincronização dos elementos do sistema e provocar a intervenção do utilizador caso estejam a ser quebradas as regras definidas. Estas decisões de sincronização podem levar à criação de novas regras de coerência que alimentam a análise respectiva (Figura 6).

Apesar de estas acções serem levadas a cabo em tempo de execução pressupõem que existe uma sequência nos pedidos ao utilizador. Em alguns casos o utilizador pode eliminar alguns pedidos posteriores se tomar determinadas decisões. Como exemplo, se não sincronizar uma classe no código com uma classe com o mesmo nome no modelo também não poderá sincronizar os atributos de ambas as classes.

Os diversos tipos de ficheiros criados pela ferramenta são ficheiros em formato XML que incluem a informação conceptual do projecto, informação gráfica do projecto, regras de coerência, acções de sincronização, e os aspectos gráficos da maior parte dos ecrãs da aplicação que são gerados automaticamente.

```
<Actions>
    <GroupChoice id="1" name="Uml Class doesn't include a CSharp attribute">
        <Input inputType="UmlClass"/>
        <Input inputType="UmlClass"/>
        <Input inputType="UmlClassAttribute"/>
        <SynchronizationAction guiLabel ="Add attribute to UML class."
```

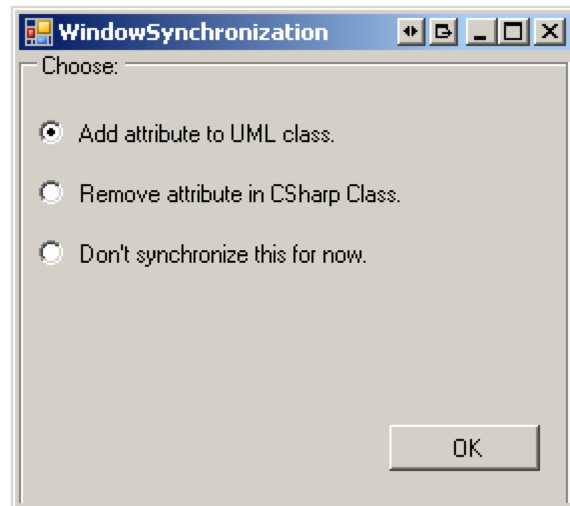

Figura 7 – Código XML correspondente a uma acção de sincronização e ecrã gerado automaticamente pela aplicação

A Figura 7 representa uma parte do ficheiro XML que descreve uma decisão de sincronização. Neste caso um atributo de uma classe C# não existe na classe UML correspondente (o utilizador recebe essa informação num ecrã anterior). Após ser tomada a decisão a aplicação terá de agir em conformidade através de um grupo de acções pré-estabelecidas e altamente parametrizáveis. Mais uma vez é usada a capacidade de reflexão da linguagem C# na implementação desta funcionalidade.

7. Conclusão

A actividade de desenvolvimento de sistemas de informação tem progredido ao longo dos anos principalmente naquilo que diz respeito às linguagens utilizadas e às metodologias adoptadas. No entanto as ferramentas de apoio à modelação continuam a realizar quase o mesmo tipo de actividades (embora mais rápida, fácil e correctamente) do que no passado. É necessário o aparecimento de uma nova classe de aplicações que consiga produzir, gerir e manter ao longo do tempo as relações entre os diversos elementos do sistema de informação. Não se trata apenas da modelação, mas sim de todas as actividades inerentes à especificação, desenvolvimento e manutenção do sistema de informação.

A importância deste problema pode ser facilmente verificada se se analisar aquilo que se passa na Construção Civil, p.ex. com a construção de um edifício. Se um arquitecto realizar um projecto e não existir ninguém que verifique a sua execução, podem existir desvios significativos em relação àquilo que foi determinado inicialmente. Estes desvios podem levar inclusivamente a um aspecto diferente daquele que o arquitecto tinha em mente quando produziu o projecto, ou mesmo a erros estruturais que eventualmente podem criar problemas

para o trabalho posterior (e.g., se for necessário acrescentar uma tomada numa certa parede já terminada é conveniente que os esquemas que localizam as redes do edifício estejam actualizadas, se tal não acontecer corre-se o risco de provocar um dano numa das redes, com maior ou menor importância consoante a rede atingida). Para que aquilo que é efectivamente construído corresponda ao projecto deve existir um acompanhamento da obra por parte de um técnico especializado que emprega técnicas de Engenharia para realizar essa actividade. Aquilo que se pretende com esta nova categoria de aplicações, denominadas *aplicações de sincronização de modelos*, é realizar de forma sistematizada e o mais automatizadamente possível as acções de acompanhamento do desenvolvimento e manutenção de forma a que todas as decisões tomadas a um nível não sejam comprometidas pela implementação a outro nível. Pode porém, devido a condicionantes específicas ser necessário contrariar alguma das regras de sincronização entre modelos (e.g. durante a migração de uma aplicação). Nesses casos passa a existir um registo automático dessas decisões, útil para desencadear posteriormente novas acções de sincronização.

No contexto dos sistemas de informação a forma de o conseguir passa por determinar e tratar as relações entre elementos de um sistema de informação. Com esta abordagem pretende-se que sejam utilizadas técnicas mais próximas da Engenharia no desenvolvimento e manutenção de sistemas de informação. Não se trata de limitar a criatividade ou a liberdade de desenvolver soluções engenhosas para os problemas que vão surgindo. Trata-se, pelo contrário, de uma forma de criar e manter mais facilmente, com melhor qualidade e com menores custos, os projectos de sistemas de informação.

8. Referências

Berard, E., *Information Technology Management*, The Object Technology, http://www.itmweb.com/essay553.htm, 2005

Bettin, J. et al., *Generative Model Transformer: An Open Source MDA Tool Initiative,* OOPSLA, http://www.softmetaware.com/oopsla2003/pos10-bettin.pdf, 2003

Dollard, K., *Code Generation in Microsoft .NET*, Apress, 2004

Herrington, J. *Code Generation In Action,* Manning Pub. Co, 2003

QVTP, *Revised submission for MOF 2.0 Query / Views / Transformations RFP*, QVT-Partners, http://qvtp.org/, 2003

Tratt, L., *QVT: The High Level Scope*, http://qvtp.org/, 2003-05-30

UMT-QVT, *UML Model Transformation Tool,* http://umt-qvt.sourceforge.net/, 2005

Welsh, T., *How Software Modeling Tools Are Being Used,* Cutter Consortium, Enterprise Architecture Advisory Service Executive Update Vol. 6 , N. 9, http://www.cutter.com/research/2003/edge031230.html, Dezembro de 2003