Modelação de Workflows com UML e Ferramentas Declarativas

Rui Gamito, Luís Arriaga da Cunha, and Salvador Abreu

LNEC e Universidade de Évora
{rgamito,lac}@lnec.pt, spa@di.uevora.pt

Abstract. A linguagem UML é actualmente uma ferramenta largamente disseminada para a modelação de sistemas de informação. No entanto, entre a esquematização conceptual de um sistema e a sua implementação existe um passo crucial de transformação que nos propomos melhorar. Este artigo apresenta trabalho para o desenvolvimento de um sistema de geração de sistemas aplicacionais, directamente baseados na especificação UML de diagramas de actividade (geração de workflows) e diagramas de classe (geração de bases de dados).
A transposição entre a modelação e a implementação é feita através do XMI gerado por uma ferramenta de modelação UML, servindo de base para a geração do sistema final. Por sua vez, este último suporta-se num *backend* implementado na linguagem de Programação em Lógica ISCO, tirando partido da propriedades declarativas desta linguagem no acesso a bases de dados, e um *frontend* AJAX, facilitando alterações assíncronas que venham a ser realizadas sobre as especificações do sistema, depois de gerado.
Este artigo, sendo relativo a um *work in progress* não propõe ainda conclusões definitivas sobre as implicações desta abordagem, mas já apresenta alguns resultados preliminares relativamente a comparações com outro sistema de gestão de fluxo de dados.

1 Introdução

O uso de uma ferramenta de modelação UML como meio para produzir código de programação é um objectivo muito apetecido, mas não muito conseguido, simplesmente por a UML não ser uma linguagem gráfica de *programação* [10], mas sim de *modelação*. A falha entre a modelação e a produção de código funcional e auto-suficiente faz-se sentir quando atendemos ao facto de muitas ferramentas apenas gerarem XML a partir dos modelos, enquanto outras conseguem realmente gerar código, por exemplo Java, mas apenas para diagramas de classe e componentes, como é o caso do *Rational Rose* [7].

No que toca ao uso disseminado da UML [11], existe uma diferença considerável na aceitação do UML em diversas organizações. No entanto, os autores deste trabalho consideram que a linguagem UML é uma ferramenta que proporciona, entre outros, uma infra-estrutura rigorosa a um nível de abstracção elevado, bem como suporte de interoperabilidade e integração entre vários domínios,

a nível semântico, considerando válida a premissa de que o uso do UML é uma vantagem.

Ao longo deste documento é descrito o procedimento e metodologia usados para a tentativa de construção de um sistema de geração de sistemas aplicacionais a partir directamente da modelação de diagramas UML. O objectivo do trabalho é tornar possível a criação de sistema de controlo de fluxo de dados ou uma base de dados, somente a partir da modelação UML de diagramas de actividades e classes, respectivamente. Adicionalmente, faz também parte do objectivo disponibilizar um mecanismo de controlo, edição e visualização de *workflows* realizado em AJAX [12], permitindo a substituição de um editor UML externo. Tenta-se também que o editor seja simples, embora completo, fugindo um pouco da complexidade de outros *Workflow Management Systems* (WfMS), como por exemplo o *OpenFlow* [1], componente do *ZOPE*, um sistema de gestão de conteúdos.

O trabalho é na sua grande maioria baseado na linguagem ISCO [6], uma linguagem assente no Prolog ampliada sintacticamente para descrever classes, predicados e *goals* ISCO, sequências, restrições de integridade e regras de controlo de acesso.

Este documento está organizado da seguinte forma: na secção 2 é descrita a problemática das escolhas e decisões que influenciaram a evolução do trabalho até ao ponto em que se encontra actualmente; na secção 3 são apresentados os pormenores técnicos relevantes, tais como estruturas de dados, arquitectura da aplicação, regras de utilização e exemplos de código, aplicados quando possível a um exemplo concreto; na secção 4 são apresentados alguns resultados e métricas sobre XMI, XSLT e ISCO; na secção 6 são apresentadas algumas conclusões preliminares sobre o trabalho realizado e são mencionadas as possibilidades de alargamento da aplicação a novas funcionalidades.

2 Procedimentos e Métodos

O editor de UML eleito para realizar a modelação é o ArgoUML [14]. Esta escolha deve-se ao facto de ser *open source*, possuir exportação dos modelos para XMI [2], suportar OCL (*Object Constraint Language*) e possuir geração de código Java (relevante para efeitos de comparação com o que necessita de ser gerado para ISCO). A contribuir para a sua escolha está também o facto de ter uma interface relativamente simples e amigável, bem como o facto de ser implementado em Java, existindo portanto a facilidade de uso em qualquer sistema operativo. Foram analisadas outras ferramentas de desenho e modelação UML, nomeadamente *Dia* [8] e *Umbrello* [13], não sendo objectivo deste trabalho apresentar uma comparação entre estas.

De acordo com os objectivos do trabalho, apenas se têm em conta diagramas que possuam uma "consequência física" directa, isto é, os diagramas que conseguem realizar por si só uma representação completa de algum sistema, que se seja por sua vez passível de possuir uma representação em ISCO. Da totalidade de diagramas oferecidos pelo UML, escolhem-se então dois: diagrama de classes

(possuindo consequências directas sob a forma de base de dados) e diagrama de actividades (com a sua consequência directa sob a forma de *workflows*). Apenas estes dois modelos são considerados, atendendo à sua complexidade individual - especialmente no que diz respeito aos *workflows* - e por se considerar suficiente como representantes de modelos UML.

É importante referir neste ponto que, apesar de se considerarem dois tipos de diagramas, o principal foco de trabalho é na definição de *workflows* a partir dos diagramas de actividade.

Surge então a necessidade de realizar um mapa exaustivo da representação de ambos os diagramas em XMI. A definição deste mapa é extremamente importante para permitir extrair o máximo de informação do XMI.

Para a recolha de informação do XMI a escolha residiu prontamente no uso de XSLT[1] [3] [4] Com o auxílio das transformações XSLT gera-se directamente código ISCO.

As estruturas de dados utilizadas assentam na base *Object-Relational* do ISCO, sob a forma de uma estrutura de classes. Uma vez que se opta por gerar ISCO a partir do XMI, as relações são automaticamente criadas e populadas. Na construção do modelo *Object-Relational* é definida uma separação clara entre o que é informação estática e o que é informação dinâmica, sendo a última correspondente às instâncias de execução dos *workflows* (execução de actividades e eventos, incluindo a chegada de informação de eventos externos).

No que toca ao mecanismo de execução dos *workflows*, este é totalmente construido em ISCO e permite criar as devidas instâncias dos modelos, bem como executá-los, obedecendo à sua lógica definida no UML.Tratando-se de um *work in progress*, apenas alguns padrões de controlo de fluxo são suportados de momento, nomeadamente os padrões *"Sequence"*, *"Parallel Split"*, *"Synchronization"* e *"Exclusive Choice"* [15] [16]. É no entanto objectivo que exista um suporte para a grande maioria.

Para a apresentação amigável da ferramenta é também usada uma das propriedades proeminentes do ISCO: a sua interface cómoda com o PHP. Desta forma temos a prespectiva extremamente aliciante de controlar o *workflow* via internet.

O modelo da aplicação é até este ponto o exemplificado na figura 1

[1] Já que XMI é uma formatação XML e o XSL é perfeito para lidar com esta formatação

Na figura 1 nota-se um problema bem claro de falta de integração entre a modelação e a visualização. É necessário uma ferramenta externa para produzir o modelo e qualquer necessidade de ajuste do mesmo tem que ser realizada pela mesma ferramenta externa. Por outro lado, para visualizar e interagir com as instâncias dos *workflows* é necessário outro suporte, já que a) não existe retro-propagação de informação (ISCO $\Longrightarrow$ ArgoUML) e b) o ArgoUML não possui, obviamente, um mecanismo de execução e controlo de fluxo. Para piorar, qualquer edição do modelo do *workflow* põe em risco possíveis instâncias que já estejam a "correr", dado que a forma de *input* de informação proveniente de um XMI está implementada de forma completamente invasiva.

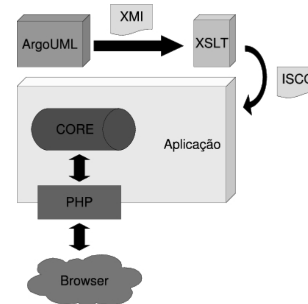

Fig. 1. Layout inicial da aplicação

São consideradas duas opções para resolver este problema:

- Continuar com o ArgoUML para re-edição dos modelos, e capacitar a introdução de dados do lado do ISCO de um mecanismo menos invasivo que se pudesse ajustar às mudanças, bem como preservar a informação possível de quaisquer instâncias activas;
- Desenvolver um editor integrado na aplicação, embora mantendo activa a entrada de XMI de forma genérica e não apenas para o ArgoUML.

A hipótese escolhida é a segunda, permitindo quebrar a dependência de uma ferramenta externa[2] e abrindo ao mesmo tempo a possibilidade de construir uma ferramenta completa de modelação, edição e gestão de *workflows* completamente integrada.

Nesta fase é feito um ajuste aos objectivos do trabalho, agora mais centrado nos *web-services*. Consideram-se hipóteses de interface para o utilizador com PHP simples, *applets* de Java e AJAX, tendo sido o último o escolhido. Após testes com AJAX apresenta-se claro que é a ferramenta ideal já que, aproveitando as suas capacidades de comunicação assíncrona com o servidor, se consegue proporcionar a melhor experiência ao utilizador tendo, por exemplo, a visualização em tempo real dos acontecimentos na execução do *workflow*. O facto da generalidade dos browsers suportarem JavaScript por omissão também ajudou na escolha do AJAX em detrimento das *applets* Java. No que diz respeito ao PHP, não se pode ignorar a vantagem da comunicação com o ISCO, pelo que é também mantido este componente.

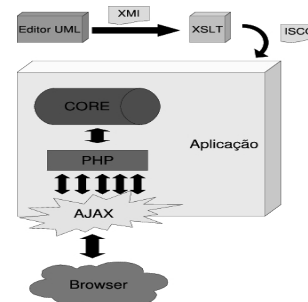

Fig. 2. Layout final da aplicação

A figura 2 representa o estado da arquitectura nesta fase.

[2] Embora mantendo-se o suporte para a entrada de modelos na aplicação a partir da interpretação de XMI

3 Modelo e Representação

Os autores de [9] sustentam que os diagramas de actividade suportam a maioria dos padrões de controlo de fluxo comuns, e mesmo outros que não são tipicamente suportados por WfMS comerciais. No entanto os diagramas de actividade *standard* apresentam algumas falhas de formalização (através de premissas OCL), sendo que a sua interpretação pode ser ambígua quando traduzida para linguagem natural. Embora estas falhas, exploradas ao longo de [9] sejam relativas apenas à *UML Revision 1.4*, podemos ver em [17] que estas não foram completamente resolvidas na revisão 2.0.

3.1 Exemplo

A figura 3 representa uma modelação UML de um diagrama de actividades muito simples. Todos os exemplos de código presentes neste artigo são referentes a esta modelação concreta. Este exemplo apenas exemplifica os padrões possíveis neste momento, de onde resulta a sua simplicidade.

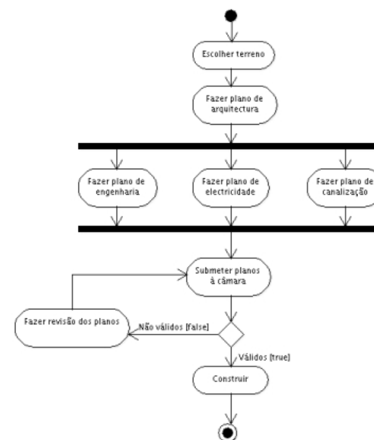

Fig. 3. Exemplo de diagrama de actividades

3.2 Relações e estados

Como foi referido na secção 2, toda a aplicação acenta sobre ISCO, que vai mantendo a persistência dos modelos estáticos dos *workflows* e principalmente da sua respectiva informação dinâmica.

Seguidamente são apresentados os dados estáticos e respectivos pormenores.

Actividades São suportadas as actividades habituais, nomeadamente *"action state"* (apelidada de "simples"), *"join"*, *"fork"*, "inicial", "final", "decisão"/"junção" e "composta". Esta última representa uma actividade que está ela própria definida usando outro diagrama de actividades, correspondendo portanto a outro *workflow*. Suporte para a actividade *"flow final"*, será adicionado posteriormente, uma vez esta fazer parte apenas de padrões de controlo mais avançados. De momento, e para restringir a quantidade de casos iniciais a cobrir, existem algumas "obrigatoriedades" no que toca ao uso das actividades. Destas destacam-se : só poder existir uma actividade final; um *fork* será sempre acompanhado de um *join*; não se processam *forks*/*joins* dentro dos mesmos.

Eventos ou Acções Cada actividade "simples" possui um evento, que é despoletado quando o controlo de fluxo atinge a actividade em questão. Dos tipos existentes de eventos, actualmente apenas é suportado o *"call event"*, responsável pela chamada de uma "função". Posteriormente será introduzido o *"time event"*, que introduzirá uma restrição temporal na execução de determinada actividade. Apenas depois será pensada a inclusão dos restantes tipos de acção (*"signal event"* e *"change event"*), não sendo relevantes de momento.

Transições Cada transição necessita de um nó de partida e um nó de chegada. Como apenas se deseja que as transições ligadas a nós decisores tenham condições de guarda (ver descrição das "condições"), definem-se dois tipos de transição: transição "condicional", possuindo a sua condição de guarda, e a simples transição "incondicional".

Condições Em teoria, cada actividade de decisão possui uma condição, responsável pela escolha do caminho a tomar pelo controlo de fluxo. No entanto, isto implicaria uma estrutura dinâmica para manter as diversas hipóteses de fluxo no nó decisor. Como tal, opta por ter a decisão um passo à frente, isto é, nas transições que partem do nó decisor. Desta forma, aquando da chegada do controlo de fluxo a uma actividade de decisão, todas as condições das transições que dele partem são avaliadas, sendo escolhida a primeira condição avaliada positivamente. Com esta abordagem consegue-se esperar sempre um resultado booleano (verdadeiro ou falso) da avaliação das condições de cada transição, o que facilita o percurso do *workflow*. No presente, esta condição apenas pode ser uma comparação simples, como se vê no exemplo da figura 3, onde se testa apenas o resultado anterior como *true* ou *false*, mas será brevemente incluída a capacidade de realizar pequenas expressões, embora sempre com resultados booleanos.

Workflows Identificadores e nomes de todos os modelos de *workflow* carregados no sistema.

Blocos Um "bloco" é um percurso dentro de um par *fork/join*. Desta forma todas as actividades dentro de um par *fork/join* têm obrigatoriamente que fazer parte de um bloco, e um apenas. Este mecanismo serve para garantir que há isolamento de dados entre os diversos blocos de uma execução paralela, como é o caso da execução a partir de um *fork*. Cada bloco possui um identificador único e a referencia às actividades *fork* e *join* que o englobam. A ligação entre as actividades e os blocos é feita através de outra relação, contendo a referência ao bloco, à actividade e ao *workflow*.

Ao contrário de toda a outra informação estática, os blocos são calculados após a análise do XMI, já sobre a informação mantida relativa do modelo.

São apresentados em seguida os tipos de informação dinâmica.

Execução Uma "execução" é uma instância de um *workflow*. Cada instância possui o identificador do *workflow* a que se refere, uma *flag* de estado de execução (informando se a instância está activa ou já terminou), uma lista de actividades a serem executadas (para auxiliar na execução paralela) e o "traço" da execução, que permitirá saber exactamente o percurso de execução efectuado.

Instâncias de eventos Um evento é despoletado quando o controlo de fluxo chega a uma actividade que o contém. O evento é instanciado para conter informação específica sobre a sua invocação, nomeadamente a execução activa, o bloco que contém a actividade (caso exista) e o resultado da chamada (ver 3.2). Isto é particularmente útil para manter históricos de resultados.

Buffer de entrada Esta relação será responsável por armazenar temporariamente os resultados das chamadas. Este processo é explicado a seguir. É criada uma instância de evento, com o resultado da chamada a *"null"* e é realizada a chamada. Nesta chamada é passada a informação necessária para que a "função" chamada saiba onde colocar a resposta, ou seja, toda a informação necessária para criar um registo no "buffer de entrada".

Um *daemon* externo ao motor de execução de *workflows* vai ser responsável por ver quando a resposta da chamada chega ao buffer e colocá-la no local adequado na relação de instâncias de eventos. Só depois de ter um resultado pode um evento ser considerado acabado, e a actividade que o inclui sinalizada como completa.

3.3 Regras de modelação necessárias

Ao deixar um utilizador usar uma ferramenta completa de modelação UML como o ArgoUML, existe uma boa probabilidade de este vir a utilizar erroneamente alguma das opções de modelação. Como tal existe a necessidade de estabelecer algumas regras no que toca à modelação dos diagramas de actividade, para que a sua "tradução" ocorra sem perdas ou falhas. Essas regras são[3]:

- um nó do tipo *action state* terá exactamente uma acção, que produzirá exactamente um resultado;
- nós do tipo *join* e decisão/junção têm **sempre** um resultado;
- ambos os resultado mencionados anteriormente têm uma duração de apenas uma transição, isto é, apenas o próximo nó (seja ele qual for) pode usar esse resultado;
- o resultado pode ser utilizado fazendo referência ao nome estático "Result" dentro do campo "expression" de uma acção (evento);

[3] Estas regras são as estabelecidas no momento da redacção deste documento, pelo que é natural que o seu número venha a ser aumentado com o suporte de outros padrões de controlo

– é excepção à regra anterior um resultado que seja definido como não-volátil, utilizando o estereotipo "non-ephemeral", podendo ser acedido a qualquer altura dentro do *workflow*;
– as acções dos nós do tipo *action state* são chamadas sob a forma de predicados comuns ISCO, que serão necessariamente codificadas externamente[4] ao ArgoUML;

3.4 Organização em "Units"

Dado que havia a noção clara de estar a trabalhar com partes de código com funções muito especificas, optou-se por organizar o mesmo utilizando os mecanismos de estruturação de programas oferecidos pela programação em lógica contextual [5]. Cada módulo é assim uma *unit* com funções específicas. A organização do código é a seguinte:

Main Esta *unit* é o ponto de união entre todas as outras. Este módulo serve de *router* de comunicação entre os outros, sendo que não há troca directa de informação. Isto ajuda à modularidade e mantém as comunicações mais simples.

PreProcess Este é o módulo responsável pela computação dos "blocos" dentro dos pares *fork-join*.

Process É nesta *unit* que são programadas as acções chamadas dentro do *workflow*. São-lhe passados os seguintes argumentos: identificador da instância de evento; identificador da instância do *workflow*; identificador do bloco (caso exista).

Model Nesta *unit* estão contidos todos os dados referentes a um modelo de *workflow*. Esta *unit* será o resultado directo do XMI processado pelo XSLT.

InterfacePHP Este módulo recebe todos os pedidos provenientes do *browser* e é responsável também pelas respostas.

DataReadWrite Esta *unit* permite ler e escrever na base de dados.

Engine Este módulo é responsável pela execução das instâncias dos *workflows*.

[4] A necessidade da especificação externa das acções prende-se com o facto de ser extremamente difícil garantir que quem faz a modelação consiga programar de facto as acções. Por outro lado, seria necessário estabelecer uma gramática rígida e específica para que as acções pudessem ser inseridas durante a modelação. Desta forma, tendo a codificação fora da modelação permite-se que ambas sejam definidas por utilizadores diferentes, e em tempos diferentes.

3.5 Arquitectura global

A figura4 apresenta o esquema da arquitectura global da aplicação contendo todas as partes que vieram a ser descritas.

Esta arquitectura permite ao utilizador inserir um modelo realizado num editor externo, desde que este possua uma forma de exportar os modelos sob a forma de XMI, bem como construir a modelação completamente a partir de um *browser*, directamente na aplicação. Essa mesma interface permite-lhe ter o controlo de gestão de fluxo do *workflow*.

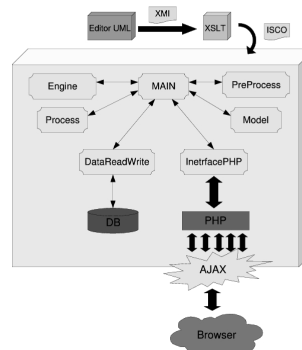

Fig. 4. Esquema da arquitectura global da aplicação

4 Resultados

A seguir são apresentadas algumas métricas caracterizando o que acontece ao longo das diversas fases desde o editor ArgoUML até se ter a representação do modelo em ISCO.

Todo o XMI do ArgoUML é construido fazendo encapsulamento dos componentes em blocos com identificadores, que por sua vez são referenciados noutros blocos. Estas referências cruzadas de blocos ao longo do XMI faz com que sejam necessários mais *templates* no XSLT de modo a conseguir realizar os percursos necessários ao longo XMI. Estes *templates* podem conter ciclos que pesquisam todos os nós de determinado nível da árvore de XMI de modo a conseguir encontrar as informações que procuram, o que aumenta a complexidade de pesquisa para quadrática, sobre o número nós.

A tabela seguinte relaciona o tipo de componente com as médias da quantidade de linhas geradas, linhas de XSLT usado para tratar cada um deles, chamadas a *templates* e ciclos que entram na pesquisa.

Componente	Linhas XMI	Linhas XSLT	N Templates	Ciclos
Join/Fork/Split/Merge	7+In+Out[5]	35	2	1
Act. inicial/final	6	35	2	1
Act. simples	17	63	5	3
Transição simples	8	19	2	1
Transição condicional	20	69	7	3

Para o código ISCO gerado, pegando no exemplo 3, temos aproximadamente 60 linhas de código geradas, com o seguinte aspecto:

```
fact(workflow(w1)).
fact(event_call(ev2,
            fazer_plano_de_arquitectura,
            process(fazer_plano_de_arq,_,_,_))).
...
```

```
fact(activity_start(start, w1, 'Actividade inicial')).
fact(activity_simple(act1, w1, 'Escolher o terreno', ev1)).
...
fact(activity_join(join1, w1, 'Join de todos os planos', 3, 3)).
fact(activity_decision(decision1, w1, 'Decisao sobre os planos')).
...
fact(transition_unconditional(trans1, act1, start)).
...
fact(condition(cond1, (XPTO == true), variable(XPTO))).
fact(condition(cond2, (XPTO == false), variable(XPTO)))
```

Com este código temos modelo pronto a ser interpretado pela aplicação.

5 Comparação com trabalho relacionado

É seguidamente apresentada uma pequena comparação de alguns pontos entre a nossa aplicação (APP) e o sistema de gestão de *workflows OpenFlow* (OF).

OF - A definição das condições das transições é realizada na linguagem TAL (Template Attribute Language), como por exemplo python:instance.some_property=='value'. Os parâmetros da condição são normalmente variáveis de instância.

APP - A definição das condições é realizada sob a forma de OCL. Por omissão qualquer condição é avaliada contra o resultado da actividade anterior. Pode-se no entanto explicitar que o resultado de uma actividade é persistente, podendo ser avaliado em qualquer local do *workflow*, dentro do âmbito da execução.

OF - A definição de eventos (aplications) é feita separadamente, sendo a sua chamada feita a partir da referência a um URL dentro do Zope. Desse URL constará uma qualquer função aplicacional desejada.

APP - A definição dos eventos é feita no sistema, num menu especificamente designado para a função e é implementada em Isco (goals). Os eventos são invocados através da chamada do seu goal.

OF - Falta de capacidade de visualização do modelo de workflow, quer em tempo tempo de definição quer em tempo de execução. Pode-se no entanto usar o OpenFlowEditor para obter visualização e alguma capacidade limitada de interação com o mesmo (criação/remoção de actividades/transições e atribuição de tarefas a utilizadores);

APP - Tudo é feito graficamente sobre o modelo do workflow. A sua construção é feita arrastando e ligando os diversos componentes, e preenchendo os menus de propriedades contextuais. A execução assinala em tempo real o que se passa numa instância do utilizador.

OF - Interface complexa (comandada pelo estilo "industrial" do ZOPE).

APP - Interface simples e intuitiva, altamente reactiva às acções do utilizador. Este artigo não apresenta nenhum *screenshot* da interface porque esta ainda se encontra numa fase de testes, em que se estão a avaliar quais as bibliotecas AJAX a usar.

OF - Exportação dos *workflows* em formato ZEXP.

APP - Exportação dos *workflows* em formato XMI, com formato semelhante ao que pode ser importado.

OF - O *OpenFlow* possui vários *roles*, correspondendo os mesmos aos vários utilizadores intervenientes no desenrolar do *workflow*. As actividades podem ser atribuidas a um utilizador, sendo que este, ao fazer *login*, pode visualizar quais as actividades que tem pendentes.

APP - A noção de utilizadores e grupos de utilização será implementada brevemente.

6 Conclusões e Trabalho Futuro

Para se poder usar o editor integrado em vez do editor externo de UML, é necessário chegar a algumas conclusões sobre o que se deseja do próprio editor. Antes de mais é necessário decidir se nos mantemos puristas, usando unicamente UML nas modelações, ou se disponibilizamos uma ferramenta mais específica e funcionalmente adaptada aos *workflows*. A conclusão é que uma mistura dos dois é o mais adequado. Temos assim uma base sólida de UML, juntamente com flexibilidade suficiente para chegar a padrões de fluxo para os quais a UML não oferece solução. Do ponto de vista de utilização, faz muito mais sentido ter o controlo completo de gestão e edição dentro do mesmo ambiente. Do ponto de vista técnico as vantagens de ter este controlo são bastantes, especialmente porque temos interacção directa com o motor de execução, o que permite realizar alterações em tempo real sobre modelos que já possuam instâncias em execução (em semelhança do que acontece com o *OpenFlow*).

O facto desta aplicação estar baseada em ISCO permite tirar partido das capacidades declarativas para implementar usos "inteligentes" usando pouco mais do que uma chamada de um *goal* ISCO. Exemplo disso são as contas de utilizadores e a respectiva construção de *task lists* relativa a tarefas em instâncias de *workflows*.

São seguidamente apresentados mais dois dos usos que podem ser dados às propriedades declarativas do ISCO para chegar a resultados benéficos ao utilizador.

Construção assistida de workflows Esta ideia baseia-se no pressuposto de colocar a aplicação a ajudar em tempo real o utilizador enquanto este modela o seu *workflow*, através da apresentação de pista visuais e mesmo técnicas sobre as opções a seguir na modelação. Poder-se-à ter inserção directa de padrões de controlo ou mesmo avaliação sobre o modelo de modo a sugerir reorganizações do mesmo. Outro modo de construção assistida poderá ser a simulação automática de execução com base, por exemplo, em resultados pré-estabelecidos das condições ou conjuntos de variação de resultados. Isto permitirá avaliar os diferentes percursos de fluxo e detectar erros de execução.

Google de workflows Baseada num motor de análise de "analogias" de modelos, esta ideia permitiria elaborar buscas sobre "sinónimos" de um modelo ou porç porções do mesmo, apresentando como resultados os modelos que fossem funcional e/ou modularmente idênticos.

References

1. Openflow homepage. http://www.openflow.it/EN/index_html. Homepage.
2. Xml metadata interchange (xmi), v2.1. http://www.omg.org/cgi-bin/doc?formal/2005-09-01. XMI specification.
3. Xslt elements reference. http://www.w3schools.com/xsl/xsl_w3celementref.asp.
4. Xslt reference. http://www.zvon.org/xxl/XSLTreference/Output/index.html.
5. Salvador Abreu and Daniel Diaz. Objective: in Minimum Context. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, pages 128–147. Springer-Verlag, 2003. ISBN 3-540-20642-6.
6. Salvador Abreu and Vítor Nogueira. Using a Logic Programming Language with Persistence and Contexts. In Masanobu Umeda and Armin Wolf, editors, *Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2005)*, Fukuoka, Japan, October 2005. Waseda University.
7. Wendy Boggs and Michael Boggs. *Mastering UML with Rational Rose 2002*. Sybex, 2002.
8. Kevin Breit, Henry House, and Judith Samson. Dia. http://www.gnome.org/projects/dia/doc/dia-manual.pdf. User's Manual.
9. Marlon Dumas and Arthur H.M. ter Hofstede. Uml activity diagrams as a workflow specification language. In *Proceedings of the UML'2001 Conference*, 2001.
10. Ricardo Pereira e Silva. Automatic code generation from uml models.
11. Veronica Fredriksen. Wide gap amongst developers' perception of the importance of uml tools, developereye study reveals. *Express Press Release*, 2005.
12. Jesse James Garrett. Ajax: A new approach to web applications. February 2005. Seminal.
13. Paul Hensgen. Umbrello uml modeller handbook. http://docs.kde.org/stable/en_GB/kdesdk/umbrello/index.html.
14. Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp. Argouml user manual - a tutorial and reference description. http://argouml-stats.tigris.org/documentation/manual-0.22/.
15. Will M.P. van der Aalst, Alistair H.M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns site. http://is.tm.tue.nl/research/patterns/flash_animations.htm, 2001.
16. Bartek Kiepuszewski Will M.P. van der Aalst, Alistair H.M. ter Hofstede and Alistair P. Barros. Workflow patterns. Technical report, Queensland University of Technology, 2002.
17. van der Aalst W. Dumas M. ter Hofstede A. Wohed, P. and N Russell. Pattern-based analysis of uml activity diagrams. In *Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005)*, Klagenfurt, Austria. Springer.