2013

Combinando workflows e semântica para facilitar o reuso de software

Revista de Informática Teórica e Aplicada, Porto Alegre, v. 20, n. 2, p. 73-89, 2013
http://www.producao.usp.br/handle/BDPI/45843

Combinando Workflows e Semântica para Facilitar o Reuso de Software

Luciano A. Digiampietri[1], Jonatas C. Araújo[1], Éric H. Ostroski[1], Caio R. N. Santiago[1], José de Jésus Pérez-Alcázar[1]

Resumo: Este artigo apresenta um ambiente para o desenvolvimento de software baseado no reuso de componentes, que combina tecnologias de gerenciamento de workflows, semântica e componentes de software. O ambiente desenvolvido pode ser utilizado por desenvolvedores de software, projetistas de workflows e usuários finais. Um estudo de caso real é apresentado para demonstrar a utilização do ambiente.

Abstract: This paper presents an environment for component-based software development, which combines workflow management, semantics and software components technologies. The resulting environment can be used by different sorts of users: software developers, software designers, and end users. One real study case is presented to demonstrate the use of the environment.

1 Escola de Artes, Ciências e Humanidades da Universidade de São Paulo (EACH-USP),
Av. Arlindo Béttio, 1000 Ermelino Matarazzo - São Paulo SP, CEP: 03828-000
{digiampietri, jonatascaraujo, hainer, caio.santiago, jperez @usp.br}

1 Introdução

Nos últimos anos, diversos avanços foram feitos nas áreas de reuso de software, gerenciamento de fluxos de trabalho (*workflows*) e uso de semântica. Enquanto cada uma dessas áreas está continuamente evoluindo surgem oportunidades de combiná-las a fim de se obter soluções mais robustas, eficientes, inteligentes ou mesmo com maior usabilidade, facilitando sua adoção pelo usuário final.

Na área de reuso de software, vários esforços têm sido feitos para tornar o desenvolvimento de componentes de software mais efetivo, bem como para facilitar o armazenamento, a integração e o consequente reuso dos mesmos [1-4]. Estes esforços incluem a proposta de novos modelos para desenvolvimento e anotação de componentes até o desenvolvimento de infraestruturas específicas para facilitar a descoberta e composição dos componentes. Porém, há riscos associados ao reuso de software, relacionados principalmente ao custo inicial de adaptação dos softwares já desenvolvidos a componentes de software reutilizáveis de acordo com o modelo adotado e a manutenção do repositório de componentes [2,5]. Além disso, são necessários mecanismos sofisticados de descoberta de componentes para garantir que os componentes desenvolvidos sejam facilmente encontrados [6-7].

Estudos sobre gerenciamento de workflows têm ocorrido há décadas e ganharam destaque nos últimos anos no gerenciamento de experimentos na e-Science [8] e gerenciamento de workflows para ambientes de alto desempenho [9-10]. Dentre os principais desafios ligados aos sistemas para o gerenciamento de workflows estão a identificação automática das atividades que atendam às necessidades dos usuários e a execução paralela e tolerante a falhas de suas atividades [6,10-12].

O uso de semântica em sistemas computacionais vem crescendo gradativamente visando principalmente a facilitar o entendimento automatizado de conteúdos disponíveis na Web, contribuindo assim para o compartilhamento e reuso de dados e serviços [13]. A web semântica pode servir de base para possibilitar descoberta, composição e orquestração automáticas de serviços Web ou componentes digitais de software de forma a facilitar e ampliar o reuso de software e a criação e execução de workflows [14].

Este artigo combina essas três áreas a fim de criar um ambiente de desenvolvimento que possa ser utilizado por projetistas de software para a rápida criação de novos sistemas com base nos componentes existentes; porém também se pretende auxiliar no trabalho do desenvolvedor de software de forma a facilitar a implementação de novos componentes ou mesmo estender ou personalizar facilmente os sistemas produzidos automaticamente; e por fim, espera-se que até usuários finais sem formação em computação possam tirar vantagem do ambiente, utilizando-o como uma ferramenta para testar e calibrar sistemas já desenvolvidos.

O restante deste artigo está organizado da seguinte forma. A Seção 2 descreve alguns conceitos básicos que serão utilizados ao longo do artigo. A Seção 3 apresenta a solução proposta. Dando continuidade, a Seção 4 descreve aspectos de implementação. A Seção 5 apresenta dois estudos de caso. A Seção 6 apresenta uma discussão sobre o reuso de componente em trabalhos correlatos. Por fim, a Seção 7 contém as conclusões e trabalhos futuros.

2 Conceitos Básicos

Um workflow consiste de um conjunto de tarefas que precisam ser realizadas para se atingir um objetivo. Estas tarefas podem ser ligadas por fluxos de dados ou por restrições de ordem (também chamadas de fluxos de controle).

Neste trabalho, o termo *workflow* será utilizado para descrever novos softwares gerados a partir da composição de tarefas mais simples (componentes de software). Estes workflows podem ser executados dentro do ambiente que será apresentado na Seção 3, ou podem ser exportados para códigos executáveis, funcionando de maneira independente do ambiente apresentado neste artigo.

As tarefas de um workflow, também chamadas de atividades, são as unidades básicas do workflow. O sistema de workflows desenvolvido neste trabalho possibilita o uso de dois tipos de componentes de software como tarefas de workflows: métodos desenvolvidos na linguagem de programação Java[2] e serviços Web.Um componente de software nada mais é do que o encapsulamentos de um pedaço de software executável que possua uma interface bem definida [15].

Cada tarefa possui um conjunto de entradas e saídas. Tanto as entradas e saídas quanto as tarefas em si poderão ser descritas semanticamente. A descrição semântica é feita anotando-se as tarefas com termos das ontologias de domínio e de componentes. Uma ontologia é a descrição compartilhada de conceitos e das relações entre conceitos de um dado domínio. Elas permitem não apenas resolver questões de ambiguidade como também o processamento automatizado de recursos [16].

Há três papéis principais que os usuários do ambiente desenvolvido podem desempenhar: o desenvolvedor de software, o projetista de workflows e o usuário final. O desenvolvedor de software é a pessoa que desenvolverá os métodos ou serviços Web que são os componentes básicos de reuso do sistema. Além disso, o desenvolvedor também poderá estender o código exportado pelo sistema para adicionar funcionalidades ou adaptá-lo para satisfazer algum requisito. O projetista de workflows é a pessoa que conhece as funcionalidades do novo software que será desenvolvido e combina os componentes existentes a fim de criar este sistema. O usuário final é o usuário que apenas usará os sistemas ou *workflows* produzidos. Estes sistemas poderão ser executados dentro do

[2] http://java.sun.com/

ambiente proposto ou como ferramentas *stand-alone*. Obviamente, uma única pessoa poderá exercer mais do que um dos papéis apresentados.

3 Solução proposta

Este trabalho combina tecnologias utilizadas em componentes digitais, gerenciamento de experimentos científicos e Web Semântica a fim de facilitar o reuso de software.A solução desenvolvida considera que cada componente de software a ser reutilizado é uma tarefa de um workflow e cada novo software ou sistema desenvolvido será um novo workflow (que por si só também poderá ser considerado um novo componente de software).

Na solução desenvolvida, os componentes de software podem ser de dois tipos: métodos desenvolvidos na linguagem de programação Java ou serviços Web. Enquanto serviços Web apresentam diversas vantagens quanto ao compartilhamento e execução remota, eles também apresentam algumas desvantagens relacionadas ao desempenho na execução, limitações quanto à transmissão de grandes volumes de dados e ao fato dos serviços não armazenarem estados (*stateless*) [17-18]. Assim, optou-se por permitir a construção de softwares de maneira híbrida, mas neste artigo será dada ênfase no uso de métodos Java.

Para que um método escrito em Java seja considerado um componente, basta que a classe a qual este método pertença faça parte do *classpath* da infraestrutura desenvolvida. Desta forma, qualquer usuário pode adicionar suas próprias bibliotecas de métodos ao sistema. Adicionalmente, recomenda-se que estes métodos sejam anotados semanticamente a fim de permitir ao sistema buscas e validações mais complexas sobre os métodos disponíveis e workflows desenvolvidos. De qualquer forma, até os métodos não anotados ficam disponíveis como componentes no sistema. Para um serviço Web ser utilizado pelo sistema basta que a URL de sua descrição WSDL seja adicionada ao sistema.

As anotações semânticas são baseadas no modelo de anotação de serviços Web Semânticos utilizando *Web Ontology Language* (OWL) e *Web Ontology Language for Services* (OWL-S) [19]. Para estas anotações, duas ontologias são criadas: uma de domínio e outra dos componentes de software. A ontologia de domínio descreve semanticamente os dados que são utilizados pelos componentes. Qualquer dado não anotado é considerado pertencente ao conceito *Thing*, conceito mais abstrato na ontologia de domínio. A ontologia de componentes de software descreve semanticamente cada componente, não apenas ligando o componente a um conceito relacionado à sua função, mas também descrevendo suas entradas e saídas (que serão anotadas de acordo com conceitos da ontologia de domínio). Outras anotações também podem ser utilizadas, por exemplo, para determinar o valor padrão dos parâmetros de entrada.

Além das anotações semânticas, outras informações são utilizadas para a descoberta e validação dos componentes de software. São elas: os parâmetros de entrada de cada componente (nomes e tipos) e a saída do componente. As informações sobre os tipos dos

parâmetros são utilizadas para validar as conexões entre diferentes componentes dentro do workflow (verificação sintática). Além disso, quando disponíveis, as anotações semânticas são utilizadas para verificar a compatibilidade semântica entre as conexões. Esta funcionalidade foi implementada seguindo o modelo de verificação de compatibilidade sintática e semântica de componentes digitais proposto por Santanchè e Medeiros [20].

Conforme dito, workflows são especificados como conjuntos de tarefas (componentes de software). Duas tarefas podem ser conectadas de duas maneiras: por fluxos de dados (através de seus parâmetros, isto é, a saída de uma tarefa é utilizada com entrada de outra) ou através de um fluxo de controle, em ambos os casos, uma das tarefas só poderá ser executada após o encerramento da outra. Enquanto fluxos de dados ligam a saída de uma tarefa com a entrada de outra e representam a passagem de dados entre atividades, fluxos de controle ligam diretamente uma atividade a outra e indicam que uma atividade só poderá iniciar sua execução após a conclusão da execução da outra.

Em sistemas baseados apenas em fluxos de dados (como no caso dos chamados *pipelines* ou *pipes and filters*) não existe necessidade de fluxos de controle, pois o dado produzido por uma tarefa é consumido e transformado pela tarefa seguinte. Este tipo de sistema apresenta algumas limitações: não permite processamento paralelo e não permite o compartilhamento de dados que não sejam saídas das atividades [21]. Sistemas deste tipo não permitem, por exemplo, que diversas atividades compartilhem um dado produzido por uma certa atividade (e que esteja disponível em memória).

A fim de superar estas limitações, a solução desenvolvida neste trabalho permite o compartilhamento de memória e execução concorrente de tarefas. A Figura 1 contém o diagrama da arquitetura desenvolvida. A arquitetura contém três módulos principais: Interface, Processamento e Dados.

A *Interface* contém a interface gráfica com a qual o usuário interage com o sistema. Esta interface foi desenvolvida com dois propósitos: (i) otimizar o trabalho de um desenvolvedor de software que deseje construir novos sistemas a partir dos componentes disponíveis e (ii) permitir que usuários sem grandes conhecimentos em computação possam criar ou ao menos manipular workflows a fim de adaptá-los para seus objetivos.

Há dois módulos principais de processamento, um deles é responsável pela edição e verificação do workflow e o outro pela execução dos workflows. O *Editor* contém todas as funcionalidades ligadas à criação e manipulação de um workflow. Através deste módulo, tarefas são adicionadas ao workflow, bem como os fluxos de dados e de controle que conectam essas tarefas. As tarefas podem ser selecionadas pelo nome ou por filtros considerando as anotações semânticas de sua funcionalidade, saída, ou dados de entrada. O Verificador é responsável por verificar se os dados relacionados a cada fluxo de dados são compatíveis sintática e semanticamente. Além disso, ele também é responsável por verificar se não existem dependências circulares entre as tarefas.

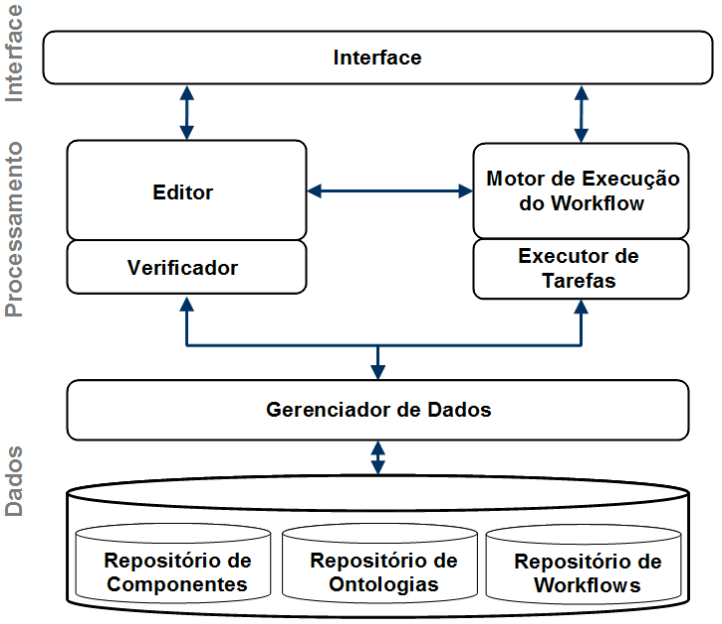

Figura 1. Arquitetura do sistema

O *Motor de Execução do Workflow* recebe e executa um workflow. Cada tarefa do workflow é executada de maneira concorrente em uma nova *thread*. O *Executor de Tarefas* é responsável pela execução individual de cada tarefa. Semáforos são utilizados para verificar as dependências de dados e do fluxo de controle, permitindo que todas as tarefas cujas dependências estejam resolvidas sejam executadas de maneira concorrente.

O módulo *Gerenciador de Dados* é responsável pela interface entre os módulos de processamento e os repositórios de dados. O sistema possui três repositórios principais: *Repositório de Componentes* que contém as referências e anotações dos métodos Java ou às referências e descrições dos serviços Web; o *Repositório de Ontologias*, que contém as ontologias de domínio e de componentes; e o *Repositório de Workflow* que contém os workflows já salvos pelo sistema.

As próximas subseções apresentam o sistema desenvolvido, detalhando os aspectos e avanços relacionados às principais funcionalidades do sistema.

3.1 Funcionalidades iniciais

A Figura 2 apresenta uma cópia da tela inicial do sistema desenvolvido.

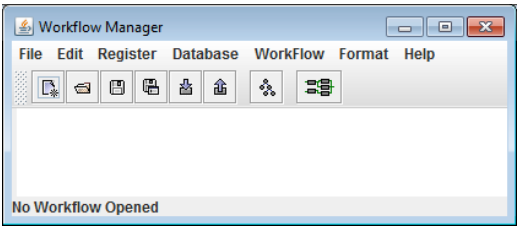

Figura 2. Cópia da tela inicial do sistema

As funções principais desta tela são: iniciar um novo workflow, salvar e abrir um workflow; importar um workflow salvo como um arquivo RDF[3], exportar um workflow para um código executável, abrir uma ontologia e abrir um plano.

Das funções iniciais há três que merecem maiores explicações, por distinguirem o trabalho desenvolvido neste artigo de trabalhos correlatos. São elas: exportar workflow para código executável, abrir uma ontologia e abrir um plano.

Exportação para código executável. Uma das funcionalidades mais importantes da solução proposta é a capacidade da geração de novos sistemas a partir dos componentes já desenvolvidos. Enquanto para muitos usuários, especialmente aqueles que não são desenvolvedores de software, o ambiente proposto possa ser usado para executar workflows ou testar configurações dos parâmetros dos mesmos, a exportação de código é uma funcionalidade pensada especificamente para aqueles que desejem manipular ou compartilhar o código gerado, sem ficarem restritos ao uso da infraestrutura que o gerou.

Esta funcionalidade possui dois modos de operação. No primeiro, os códigos são exportados para serem executados pelo *Motor de Execução do Workflow*. No caso, o motor é exportado juntamente com o código do workflow não sendo necessário que todo ambiente gráfico seja exportado. Detalhes sobre o motor de execução são dados na Seção 4 e exemplos de código exportado podem ser vistos na Seção 5.

O segundo modo de operação, exporta o workflow como um novo código Java tradicional sem utilizar o motor de execução. A vantagem deste modo é que o código gerado é mais simples, só importará as classes cujos métodos (componentes) serão efetivamente utilizados dentro do código. De um modo geral, o código produzido será mais eficiente em termos de tempo de execução e consumo de memória. Há duas restrições quanto à implementação atual desta funcionalidade: (i) atualmente ela não permite a utilização de serviços Web (apenas métodos Java); e (ii) o código gerado é executado de maneira sequencial, ao contrário do código gerado no primeiro modo de operação que utiliza a execução concorrente de cada atividade, respeitando-se as dependências de dados e controle.

Abrir uma ontologia. Esta função permite ao usuário navegar pela ontologia de domínio ou de componentes de maneira a facilitar a seleção dos componentes que irá utilizar.

[3] http://www.w3.org/RDF/

Abrir um plano. Esta função abre um plano (conjunto de ações que resolvem um dado problema). A descrição dos mecanismos de criação de planos (composição automática de workflows com base nos componentes existentes) pode ser encontrada em [22].

3.2 Funcionalidades específicas de edição de um workflow

A Figura 3 apresenta uma cópia da tela de edição de um workflow. Esta tela é apresentada ao usuário sempre que um novo workflow for criado ou um workflow for aberto ou importado.

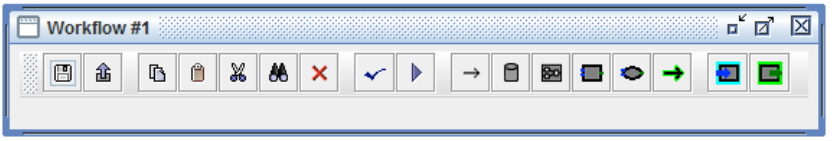

Figura 3. Cópia da tela de edição de workflows

Por meio desta interface o usuário pode gerenciar a edição de um workflow, inserindo e removendo componentes, fluxos de dados, fluxos de controle, dados iniciais e mesmo criando um novo componente que ainda não tenha sido implementado. Neste caso, o usuário criará uma tarefa especificando suas entradas e saídas e irá, posteriormente, implementá-la. Além delas, há uma função de verificação de um workflow, que é responsável por verificar três aspectos do workflow: (i) a compatibilidade sintática e semântica dos fluxos de dados; (ii) se existem dependências circulares entre as tarefas; (iii) se todas as entradas das tarefas estão preenchidas (com dados iniciais passados pelo usuário, ou recebendo a saída de outra atividade, ou com valores padrão anotados na descrição da tarefa). Caso a verificação não encontre nenhum problema, o usuário poderá solicitar a execução do workflow.

Conforme apresentado, a execução é feita de maneira concorrente, onde cada tarefa é uma *thread* que será executada quando todas as suas dependências (de dados e controle) forem satisfeitas. Detalhes sobre alguns aspectos de implementação serão dados na Seção 4.

4 Aspectos de Implementação

A implementação do sistema proposto pode ser dividida em três partes principais, quanto às tecnologias utilizadas: a interface gráfica, o verificador de um workflow e o motor de execução do workflow.

Para a interface gráfica foram utilizadas duas bibliotecas gráficas bem conhecidas da linguagem Java: Awt e Swing. A interface gráfica precisou suportar criação e manipulação dinâmica de elementos gráficos mantendo a consistência com o modelo de workflows subjacente.

A implementação do verificador de um workflow considerou dois aspectos diferentes. O primeiro é a verificação sintática e semântica dos fluxos de dados. Um fluxo é considerado

sintaticamente correto se os dados (saída de uma atividade e entrada da outra) forem do mesmo tipo ou de tipos compatíveis. A verificação semântica considera um fluxo compatível caso o conceito atribuído ao dado de saída seja igual ou um subconceito (ideia equivalente a uma subclasse em orientação a objetos) do conceito do dado de entrada. Para isso foram utilizadas ontologias e consultas ontológicas, conforme proposto em [20].

O motor de execução do workflow, bem como a modelagem computacional do workflow apresentaram alguns dos maiores desafios tecnológicos de implementação e por isso serão detalhados a seguir.

No modelo desenvolvido, o *Workflow* foi definido como um conjunto de *Tarefas* que serão executadas, tais *Tarefas* são conectadas umas as outras por *Conexões* que são responsáveis por manter o fluxo de dados entre *Tarefas*. Além disso, cada tarefa está ligada a uma *Operação* que, em suma, corresponde a o que será executado nesta tarefa. A abstração de *Operação* permite que ao invés de utilizar apenas métodos Java como tarefas, possa-se também utilizar, de maneira transparente, outros tipos de recursos, como serviços Web.

As tarefas podem possuir dois tipos de dependência: de dados, quando uma tarefa precisa dos resultados de outra, e a dependência de controle, quando uma tarefa só pode ser executada após o término de outra, ainda que não exista necessariamente um fluxo de dados entre elas. Dependências de controle são fundamentais para garantir o fluxo correto de execução de tarefas, especialmente quando existem tarefas que modificam o estado do mundo porém sem produzir explicitamente um dado de saída (por exemplo, uma tarefa ligada a autenticação em um banco de dados).

O motor de execução de workflows foi concebido pra permitir a execução concorrente das tarefas que compõem o workflow (respeitando-se as dependências de dados e controle). Desta forma, além da execução das atividades também é função do executor gerenciar as dependências entre elas, permitindo que apenas as atividades cujas dependências já estejam resolvidas possam ser executadas.

Um dos requisitos do executor de workflows é que ele possa descobrir, em tempo de execução, os nomes das classes e métodos que serão executados, bem como os tipos de retorno e parâmetros de cada método. Para satisfazer esses requisitos utilizou-se Reflexão[4] (*Reflection*), um recurso existente na linguagem Java e em outras linguagens de programação para descobrir informações de uma classe e invocar seus métodos. Este recurso foi utilizado tanto para permitir o uso de métodos Java dentro dos workflows quanto para permitir o uso de serviços Web.

[4] java.sun.com/developer/technicalArticles/ALT/Reflection

5 Estudos de caso

Nesta seção serão apresentados dois estudos de caso, um foi construído especificamente para exemplificar o ambiente proposto enquanto o outro é um estudo de caso real, correspondendo a um workflow construído para permitir a análise morfológica dos segmentos de embriões de moscas do gênero *Drosophila* [23].

5.1 Primeiro estudo de caso

O primeiro estudo de caso envolve a utilização de quatro componentes de software diferentes relacionados a operações matemáticas simples, entrada e saída de dados. O objetivo do workflow (sistema resultante) deste exemplo é executar as seguintes operações matemáticas: (*entrada1* + *entrada2*) * 4, dadas duas entradas numéricas passadas pelo usuário, chamadas aqui de *entrada1* e *entrada2*.

Para tanto, um workflow com cinco tarefas foi criado. As tarefas, todas componentes já existentes nos repositórios de componentes, são: duas tarefas de entrada de dados (que recebem os valores *entrada1* e *entrada2*, passados pelo usuário em tempo de execução), uma tarefa para somar os valores de entrada, uma tarefa para multiplicar por 4 o resultado da tarefa anterior e, por fim, uma tarefa para exibir o resultado da execução.

A Figura 4 apresenta o workflow correspondente a este estudo de caso.

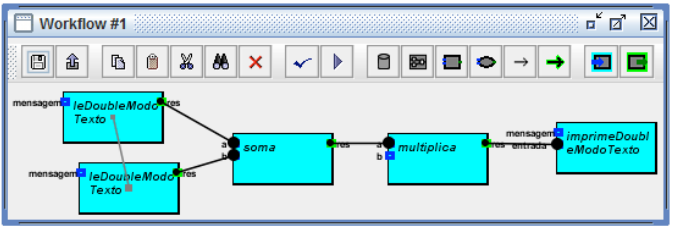

Figura 4. Cópia da tela do primeiro estudo de caso

As duas tarefas mais a esquerda da Figura 4 representam as tarefas de entrada de dados. É possível notar que elas apresentam um parâmetro de entrada, este parâmetro é a mensagem que será exibida ao usuário final quando lhe for requisitado que entre com um valor de entrada via teclado. Neste caso, a mensagem usada na primeira tarefa foi "Digite o primeiro valor:", preenchida pelo projetista do workflow conforme apresentado na Figura 5. O campo tipo do componente (*type*) não está preenchido porque esta tarefa não foi anotada semanticamente.

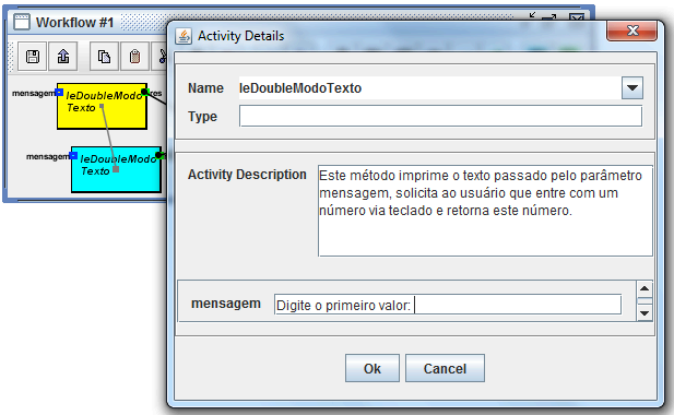

Figura 5. Cópia da tela da edição dos parâmetros de uma tarefa

Os valores inseridos pelo usuário em tempo de execução serão enviados para a tarefa de soma; o resultado da soma irá para a tarefa de multiplicação, que multiplicará por 4 o valor recebido (4 é o valor que foi inserido pelo projetista do workflow, da mesma forma que as mensagens utilizadas nas tarefas de entrada). O resultado da multiplicação irá para a última tarefa que imprimirá esse valor e a mensagem definida pelo projetista.

Além dos fluxos de dados entre as tarefas há também um fluxo de controle entre a primeira e a segunda tarefas de entrada de dados. Sem este fluxo de controle estas duas tarefas executariam de maneira concorrente e, embora isso não afete o resultado do sistema, as mensagens para o usuário poderiam aparecer fora de ordem. Isto é, primeiro a solicitação para digitar o segundo valor e depois para digitar o primeiro. Assim, este estudo de caso demonstra um uso simples, porém interessante do fluxo de controle. A Figura 6 contém as mensagens que serão apresentadas ao usuário final durante a execução do workflow. Neste exemplo, o usuário inseriu os valores 2 e 3 como entrada e o resultado produzido foi 20, correspondendo ao resultado das operações matemáticas: (2+3)*4.

```
Digite o primeiro valor: 2
Digite o segundo valor: 3
Resultado: 20.0
```

Figura 6. Cópia da tela da execução do primeiro estudo de caso

A Figura 7 contém o código correspondente ao workflow apresentado na Figura 5. Este código foi gerado pelo exportador de código operando no segundo modo, ou seja, de forma a produzir um código executável que não use o motor de execução.

O código equivalente, utilizando o motor de execução de workflows é apresentado na Figura 8. Os comandos deste código são gerados na seguinte ordem: a) todas as importações de bibliotecas são feitas; b) a classe principal é declarada, bem como o método *main*; c) um novo workflow é criado; d) todas as operações usadas dentro do workflow são criadas; e) as

tarefas (que são como instâncias das operações) são criadas; f) todas as tarefas são inseridas no workflow; g) para cada tarefa, todas as suas dependências são criadas (dados iniciais passados pelo projetista, dependências de dados entre tarefas e dependências de controle); h) por fim, a execução do workflow é iniciada.

```
public class Workflow1_CodeExported {
    public static void main(String[] args) {
        double out_double1 = Utilitarios.leDoubleModoTexto("Digite o primeiro valor: ");
        double out_double2 = Utilitarios.leDoubleModoTexto("Digite o segundo valor: ");
        double out_double3 = Matematica.Fundamentais.soma(out_double1,out_double2);
        double out_double4 = Matematica.Fundamentais.multiplica(out_double3,4);
        Utilitarios.imprimeDoubleModoTexto("Resultado: ",out_double4);
    }
}
```

Figura 7. Código exportado automaticamente correspondendo ao Estudo de Caso 1

```
import Reflection.OperacaoReflection;
public class Workflow1_WorkflowExported {
    public static void main(String[] args) throws ClassNotFoundException {
        Workflow w= new Workflow();
        Operacao op_leDoubleModoTexto = new OperacaoReflection("Utilitarios", "leDoubleModoTexto");
        Operacao op_soma = new OperacaoReflection("Matematica.Fundamentais", "soma");
        Operacao op_multiplica = new OperacaoReflection("Matematica.Fundamentais", "multiplica");
        Operacao op_imprimeDoubleModoTexto = new OperacaoReflection("Utilitarios", "imprimeDoubleModoTexto");
        Tarefa t1 = new Tarefa(op_leDoubleModoTexto);
        Tarefa t2 = new Tarefa(op_leDoubleModoTexto);
        Tarefa t3 = new Tarefa(op_soma);
        Tarefa t4 = new Tarefa(op_multiplica);
        Tarefa t5 = new Tarefa(op_imprimeDoubleModoTexto);
        w.addExecutavel(t1);
        w.addExecutavel(t2);
        w.addExecutavel(t3);
        w.addExecutavel(t4);
        w.addExecutavel(t5);
        w.addDadosIniciais("mensagem", t1, "Digite o primeiro valor: ");
        w.addDadosIniciais("mensagem", t2, "Digite o segundo valor: ");
        w.addDependencia(t2,t1);
        w.addConexao(t1, t3, "a");
        w.addConexao(t2, t3, "b");
        w.addConexao(t3, t4, "a");
        w.addDadosIniciais("b", t4, 4);
        w.addDadosIniciais("mensagem", t5, "Resultado: ");
        w.addConexao(t4, t5, "entrada");
        w.iniciaExecucao();
    }
}
```

Figura 8. Código que utiliza Motor de Execução de Workflows, exportado automaticamente

5.2 Segundo estudo de caso

Enquanto o primeiro estudo de caso visava apenas a ilustrar algumas características do sistema desenvolvido, o segundo estudo de caso apresenta uma solução real aplicada a um

problema de processamento de imagens. O resumo do problema, do ponto de vista biológico, é o seguinte. Existem alguns genes cuja expressão ou repressão aparentemente modificam a estrutura física das faixas transversais encontradas nos embriões de *Drosophilas* durante uma dada fase de seu desenvolvimento. Como medir a posição dessas faixas para verificar se realmente existe uma correlação entre a expressão desses genes e as faixas dos embriões [8]?

A resolução desse problema foi dividida em duas partes: calcular, usando técnicas de processamento de imagens, as posições relativas das faixas de cada embrião (que sofreu algum tipo de mutação ou não), em seguida analisar estatisticamente os dados para averiguar a existência de correlações entre as mutações e as posições das faixas.

O estudo de caso apresentado nesta subseção corresponde à resolução da primeira parte do problema: dada a imagem de um embrião, identificar a posição relativa de suas faixas. A implementação deste estudo de caso apresentou uma característica peculiar pois, a partir de uma imagem de entrada (contendo a foto de um embrião tirada por um microscópio), um objeto é criado e este objeto será manipulado por todas as demais tarefas do workflow. Em outras palavras, exceto a primeira tarefa, as demais não produzirão dados de saída e sim manipularão o dado produzido pela primeira tarefa. Desta forma, além dos fluxos de dados convencionais serão necessários diversos fluxos de controle para garantir a execução do experimento de maneira correta. Este tipo de preocupação muitas vezes não é comum para diversos desenvolvedores de software, pois, ao programarem um software não concorrente, a ordem dos comandos determinará o fluxo de controle. Porém, a infraestrutura desenvolvida visa a aproveitar os computadores multi-núcleos (*multicores*) comuns atualmente e por isso é baseada no processamento concorrente.

De maneira sucinta, o workflow da Figura 9 é composto das seguintes tarefas. A primeira (mais a esquerda na figura) recebe o nome de um arquivo que contenha a imagem de um embrião e converte-a para um formato específico utilizado pelas outras tarefas. As cinco tarefas que estão na parte mais baixa da figura são tarefas para exibir graficamente o estágio atual de processamento da imagem de entrada. Os resultados dessas tarefas podem ser vistos na Figura 10. O fluxo de processamento principal deste workflow encontra-se nas tarefas encontradas na parte superior do workflow. Após a conversão da imagem para o formato entendido pelas demais tarefas há seis etapas de processamento: a) limpeza do fundo da imagem; b) destaque da imagem através de mudanças no contraste; c) suavização da imagem para a remoção de ruídos; d) encontrar as extremidades do embrião; e) encontrar o histograma da região central do embrião; f) encontrar pontos de inversão no histograma (picos e vales) e, com eles, identificar os inícios e fins de cada faixa do embrião.

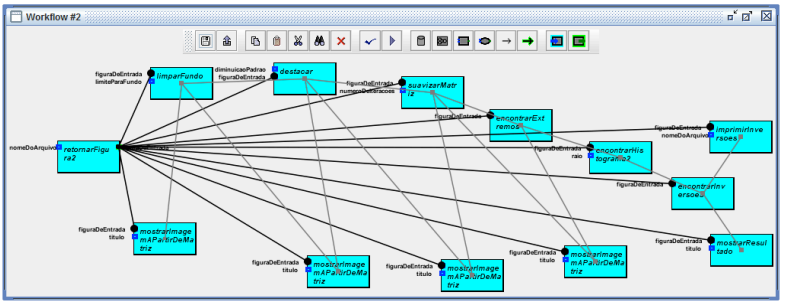

Figura 9. Workflow correspondente ao Estudo de Caso 2

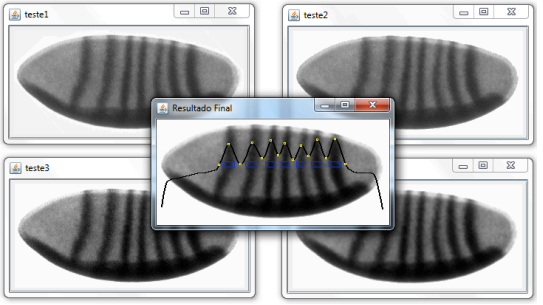

Figura 10. Resultados produzidos pela execução do workflow da Figura 9

Devido aos fluxos de dados e controle existentes neste estudo de caso, apenas duas tarefas podem ser executadas efetivamente de maneira paralela. São elas as duas tarefas mais a direita na Figura 10. Uma delas é responsável por salvar em arquivo os resultados obtidos pela execução do workflow e a outra pela exibição gráfica desses resultados.

A aplicação real deste estudo de caso utilizou as principais características do ambiente desenvolvido. Inicialmente o workflow (novo sistema baseado nos componentes de software) foi desenvolvido de maneira gráfica e diversas execuções foram feitas para se calibrar os parâmetros de entrada de cada uma das tarefas. Após se encontrar parâmetros satisfatórios, o workflow foi exportado para um código executável em Java. Este código foi estendido para que ao invés de tratar uma imagem de cada vez processasse todas as imagens de uma estrutura de diretórios.

Desta forma foi possível que diferentes usuários utilizassem o sistema: o projetista do workflow que produziu o workflow apresentado na Figura 9 e o passou para um biólogo calibrar os parâmetros; o biólogo que executou os testes; e, por fim, um desenvolvedor de software (no caso, o próprio projetista) que adaptou o código gerado para estender suas funcionalidades. Os resultados produzidos pela execução deste estudo de caso foram utilizados para confirmar hipóteses sobre a influência de um conjunto de genes na posição das faixas em artigo publicado na revista *Developmental Biology* [24].

6 Discussão

Na última década o reuso de componentes de software recebeu especial atenção devido ao desenvolvimento de novas tecnologias para facilitar o reuso e a existência de repositórios cada vez maiores de componentes disponíveis.

Existem diversas vantagens potenciais no projeto de software baseado em componentes, por exemplo, redução do tempo de produção de um sistema; redução dos custos de desenvolvimento; aumento da qualidade do produto final (já que os componentes são testados em diferentes tipos de aplicação) e o fato de ser relativamente mais fácil dar manutenção a um sistema baseado em componentes [2,5]. Porém, alguns estudos mostram que uma análise cuidadosa anterior à escolha pelo desenvolvimento baseado em componentes precisa ser feita, de forma que todas as vantagens potenciais e riscos sejam conhecidos pelos desenvolvedores [2,5,25].

O sistema desenvolvido neste trabalho enfrenta alguns dos desafios na área de reuso, especialmente aqueles relacionados à facilidade do uso do ambiente para construção de novos softwares (para os diversos tipos de usuário) e o uso de métodos de uma linguagem de programação e serviços Web sem a necessidade de adaptações para transformá-los em componentes. Além disso, a solução proposta se beneficia das arquiteturas multi-núcleos para permitir o processamento paralelo dos componentes dos sistemas produzidos.

Outra ideia utilizada foi a descrição opcional de cada componente de maneira semântica de forma a facilitar a busca e a verificação da compatibilidade entre componentes. Esta tipo de solução foi baseada no trabalho descrito em [20] e já havia sido utilizada para montar hierarquias de componentes com o objetivo de facilitar a recuperação dos mesmos [26].

7 Conclusões

Este artigo apresentou um ambiente desenvolvido para facilitar o reuso de componentes de software combinando tecnologias ligadas a gerenciamento de workflows, Semântica e Componentes Digitais.

O ambiente foi implementado de forma a permitir que diferentes recursos de software possam ser utilizados como componentes de software. Na implementação atual dois tipos de componentes foram considerados: métodos escritos na linguagem de programação Java e serviços Web. O ambiente está sendo utilizado tanto por pessoas com conhecimento em computação como por usuários de outras áreas e, conforme apresentado na Seção 5, resultados reais da utilização do ambiente já estão sendo produzidos.

Dentre os trabalhos futuros, destacam-se: extensão do sistema de exportação de código executável de forma a gerar código cuja execução poderá ser concorrente, mesmo no modo de exportação que não utiliza o motor de execução de workflow; criação de um

sistema de recomendação de tarefas a ser utilizado durante a criação de um novo workflow; adaptação do ambiente proposto para operar sobre grades computacionais (*grids*), permitindo a distribuição da execução das tarefas na grade.

Agradecimentos

O trabalho apresentado neste artigo foi parcialmente financiado pela FAPESP (projetos 2009/10413-5 e 2011/07968-5), CNPq (Bolsa Produtividade em Pesquisa 304937/2010-0) e pela Pró-Reitoria de Graduação da Universidade de São Paulo.

Referências

[1] H. John, H., and G. Spiros-Theodoros. "Reuse concepts and a reuse support repository". Proceedings of the IEEE Symposium and Workshop on Engineering of Computer-Based Systems, pp. 27–34, 1996.

[2] B. Barns and T. Bollinger. "Making reuse cost-effective". IEEE Software, v. 8, pp. 13-24, 1991.

[3] S. Henninger. "An evolutionary approach to constructing effective software reuse repositories". ACM Transactions on Software Engineering and Methodology, v. 6, pp. 111-140, 1997.

[4] S. Sadaoui and P. Yin. "Generalization for component reuse". Proceedings of the ACM-SE 42nd Annual Southeast Regional Conference, pp. 134-139, 2004.

[5] P. Vitharana, "Risks and challenges of component-based software development". Communications of the ACM, v. 46, n. 8, 2003.

[6] M. A. M. Abed. "Automatic selection and invocation of a WSMO: based semantic web service". Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications, pp. 1-8, 2011.

[7] D. Celik and A. Elgi. "A semantic search agent approach: finding appropriate semantic Web services based on user request term(s)". International Conference on Enabling Technologies for the New Knowledge Society, pp. 675-687, 2005.

[8] E. Deelman, D. Gannon, M. Shields and I. Taylor. "Workflows and e-Science: An overview of workflow system features and capabilities". Future Generation Computer Systems, v. 25, n. 5, pp. 528-540, 2009.

[9] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. "GridFlow: workflow management for grid computing". 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 198-205, 2003

[10] Y. Zhao, I. Raicu, X. Fei and S. Lu, "Opportunities and Challenges in Running Scientific Workflows on the Cloud". International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2011.

[11] M. Strohmaier and E. Yu. 2006. “Towards autonomic workflow management systems”. Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '06), pp. 1-4, 2006.

[12] W. M. P.van der Aalst. “Generic workflow models: how to handle dynamic change and capture management information?”. Proceedings of IFCIS International Conference on Cooperative Information Systems (CoopIS '99), pp. 115-126, 1999.

[13] G. Antoniou and F. van Harmelen. “A semantic web primer”. MIT Press, pp. 1–238, 2004.

[14] M. Hepp. “Semantic Web and semantic Web services: father and son or indivisible twins?". IEEE Internet Computing, v. 10, n. 2, pp. 85-88, 2006.

[15] J. Hopkins. “Component primer”. Communications of ACM, v. 43, n. 10, 27-30, 2000.

[16] T. R. Gruber, “A translation approach to portable ontologies”. Knowledge Acquisition, vol. 5, pp. 199–220, 1993.

[17] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web services: concepts, architectures and applications". Springer Verlag Berlin, 2004.

[18] J. C. Zuñiga-Torres, J. J. Pérez-Alcázar, and L. A. Digiampietri, “Implementation Issues for Automatic Composition of Web Services”. Proceedings of the Twenty-First International Workshop on Database and Expert Systems Applications, Bilbao, Spain. IEEE Computer Society. pp. 201–205, 2010.

[19] The OWL Services Coalition, “OWL-S: Semantic Markup for Web Services”, 2004. http://www.daml.org/services/owl-s/1.1B/owl-s.pdf

[20] A. Santanchè, C. B. Medeiros, “Self describing components: Searching for digital artifacts on the web”. In Proceedings of Brazilian Symposium on Databases (SBBD), pp. 10–24, 2005.

[21] J. D. Eckart, and B. W. Sobral, “A life scientist's gateway to distributed data management and computing: the PathPort/ToolBus framework”. Spring, vol. 7, pp. 79-88, 2003.

[22] L. A. Digiampietri, J. J. Pérez-Alcázar, R. S. Freitas, J. C. Araújo, E. H. Ostroski, C. R. N. Santigo. “Uso de Planejamento em Inteligência Artificial para o Desenvolvimento Automático de Software”. Proceedings of Autonomous Software Systems (AutoSoft 2011), 2011, São Paulo, SP - Brazil.

[23] U. Grossniklaus, and R. K. Pearson, W. Gehring, “The Drosophila sloppy paired locus encodes two proteins involved in segmentation that show homology to mammalian transcription factors”. Genes & Development. Vol. 6, pp.1030–1051, 1992.

[24] L. P. Andrioli, L. A. Digiampietri, L. P. de Barros, A. Machado-Lima, “Huckebein is part of a combinatorial repression code in the anterior blastoderm”. Developmental Biology, v. 361, p. 177-185, 2012.

[25] I. Osbri, S. Newell, and S. L. Pan, “Implementing component reuse strategy in complex products environments”. Communications of the ACM, December 2007, vol. 50, no.2.

[26] M. Rosenmüller, N. Siegmund, and M. Kuhlemann, “Improving reuse of component families by generating component hierarchies”. Proceedings of the second FOSD’10, pp. 57-64, 2010.