EasyBPMS: Uma Abordagem para Integração de Sistemas de Informação e Sistemas de Gerenciamento de Processos de Negócio

EasyBPMS: An Approach to Information Systems and Business Process Management Systems Integration

Mariana Sousa Bernardes
Universidade Federal de Lavras
Caixa Postal 3037
CEP: 37200-000 Lavras/MG
+5537991134094
mari.bernardes93@gmail.com

André Vital Saúde
Universidade Federal de Lavras
Caixa Postal 3037
CEP: 37200-000 Lavras/MG
+5535991365353
saude@dcc.ufla.br

RESUMO

Gerenciamento de processos de negócio tem se tornado frequente no contexto de Sistemas de Informação (SI), permitindo um fluxo de negócio mais organizado e alinhado aos requisitos de software. Porém, a integração de Sistemas de Informação com os atuais Sistemas de Gerenciamento de Processos de Negócio (do inglês, *Business Process Management System*, BPMS) ainda possui algumas limitações, tais como alta taxa de trabalho manual, falta de padronização de código e complexidade de arquiteturas BPMSs. Com base nisso, este trabalho tem como proposta uma abordagem para integração de Sistemas de Informação e BPMSs. Mais especificamente, uma abordagem que permite abstrair a captura das atividades de usuário em um SI e sua execução em um BPMS. Para fins de avaliação, a abordagem proposta foi utilizada com uma aplicação exemplo. A mesma aplicação foi integrada diretamente com um BPMS de mercado e com um *framework* de mapeamento Objeto-Processo de Negócio. A partir das três implementações do sistema, foram realizadas análises qualitativa e quantitativa.

Palavras-Chave

Processos de Negócio, BPMS, Sistemas de Informação.

ABSTRACT

Business processes management has become frequent in the context of Information Systems (IS), allowing a more organized and aligned flow of business to the software requirements. However, the integration of Information Systems with the current Business Process Management System (BPMS) still has some limitations, such as increased manual work, lack of code standardization, and complexity of BPMS architectures. Based on this, this paper proposes an approach for the integration of Information Systems and BPMSs. More specifically, an approach that allows abstracting the capture of user activities in an IS and its execution in a BPMS. For evaluation purposes, we use our approach with an example application. In addition, we integrate the same application with a market BPMS directly and with a Business Object-Process mapping framework. From the three system implementations, we performed qualitative and quantitative analyzes.

SBSI 2017, June 5th–8th, 2017, Lavras, Minas Gerais, Brazil.

CCS Concepts

• Software and its engineering → Software creation and management → Designing software • Software and its engineering→ Software notations and tools → Development frameworks and environments.

Keywords

Business Processes; BPMS; Information Systems.

1. INTRODUÇÃO

Com as frequentes mudanças do mercado e exigências impostas às empresas em geral, Sistemas de Informação (SI) mais organizados e alinhados ao negócio são cada vez mais necessários. Nesse contexto, os BPMSs surgiram como uma nova categoria de software que suportam a definição, execução e acompanhamento de processos de negócio [13]. Um BPMS é considerado uma poderosa ferramenta de gestão e pode ser definido basicamente como um motor de execução, onde os processos são efetivamente executados conforme modelados.

Existem duas alternativas ao considerar o uso de BPMSs em sistemas de software atuais [8]. Na primeira alternativa, a implementação do fluxo de trabalho é gerada exclusivamente pelo BPMS. Uma plataforma web (*workbench*) pode ser utilizada para modelar o processo de negócio e gerar as interfaces de usuário correspondentes. No entanto, com o uso dessa alternativa, uma menor flexibilidade é obtida em relação ao *design* das interfaces de usuário, integração com outros sistemas, uso de outras tecnologias web tais como os *frameworks* MVC (*Model-View-Controller*), e uso dos recursos de uma IDE, como validação, formatação e mecanismos de navegação [5] [9]. Com base nisso, não é considerada a primeira alternativa neste trabalho.

Na segunda alternativa, o BPMS pode ser usado como um sistema que suporta apenas o fluxo de trabalho baseado na camada de negócio de um SI. Neste caso, a aplicação pode ser desenvolvida com sua própria arquitetura e linguagem de programação e uma interface é requerida para comunicação com o BPMS. Entretanto, a integração entre componentes do Sistema de Informação e atuais BPMSs possui algumas desvantagens. Por exemplo, a falta de padronização durante a integração. Caso seja alterado o motor de

processos, códigos não podem ser reutilizados [8]. Além disso, os desenvolvedores precisam manipular elementos de baixo nível, tais como atividades e eventos, que não fazem parte do domínio da aplicação [9][2]. A referência a esses elementos pode desencadear um maior esforço de implementação durante a integração, bem como aumentar a curva de aprendizado, gerando gastos de capacitação de desenvolvedores [4].

Geralmente, alterações no código do aplicativo são necessárias para capturar os eventos que executam as atividades do processo em um BPMS. Para execução das atividades de usuário especificamente, os desenvolvedores precisam monitorar e manipular os componentes do negócio que geram os eventos no SI relacionados ao fluxo do processo. A compreensão da semântica do negócio para execução das atividades de usuário é essencial para gerir eficazmente o fluxo e os recursos humanos [12].

Com base nos problemas anteriormente mencionados e foco voltado para as atividades de usuário, este trabalho tem como proposta uma abordagem de integração que permite abstrair a captura das atividades de usuário em um SI e sua execução em um BPMS. O objetivo é apoiar uma menor curva de aprendizagem de componentes de baixo nível provenientes de um BPMS, menor esforço de programação durante a integração, controle principal da aplicação a partir do SI e independência de BPMS.

Este artigo está organizado da seguinte forma: Seção 2 apresenta uma visão geral da abordagem proposta. Seção 3 apresenta a arquitetura projetada para a abordagem. Seção 4 apresenta a ferramenta desenvolvida a partir da arquitetura proposta. Seção 5 apresenta um exemplo de uso da ferramenta. Seção 6 descreve a avaliação da abordagem, onde um sistema exemplo é implementado utilizando a abordagem proposta, um BPMS diretamente, e uma outra ferramenta denominada NextFlow. Seção 7 cita algumas ameaças a validade. Seção 8 apresenta alguns trabalhos relacionados. Seção 9 conclui este trabalho.

2. VISÃO GERAL

A abordagem proposta foi denominada EasyBPMS e tem como objetivo facilitar a comunicação entre o SI e o BPMS. De modo geral, consiste em detectar eventos gerados em interfaces de usuário de um SI, que possuem relação com o processo de negócio, e delegar esses eventos a motores de processo, de forma que o fluxo possa ser executado por um BPMS. Por exemplo, o *status* de uma viagem foi atualizado de iniciado para finalizado. A alteração de *status* da entidade Viagem na aplicação por algum usuário pode corresponder à execução de alguma atividade do processo [7]. A captura dessa atividade a partir da abordagem EasyBPMS, em contraponto com a forma tradicional de uso do BPMS, permite um menor grau de acoplamento entre o SI e componentes do motor de processos, bem como uma menor curva de aprendizado de APIs BPMS. As Figura 1 e Figura 2 apresentam a sequência de passos para integração do BPMS e SI com uso do EasyBPMS.

No Passo 1, um usuário acessa a interface do sistema e realiza algum serviço de acesso a dados. O serviço de acesso a dados realizado na aplicação pode corresponder ao início do processo em um BPMS. Dessa forma, para que o motor de processos tenha conhecimento do fluxo iniciado, um observador de início de processo é notificado. Esse observador é um componente da abordagem EasyBPMS que é associado a uma determinada definição de processo.

No Passo 2, o observador de início de processo notificado acessa o banco de dados do EasyBPMS para buscar a definição de processo compatível com a entidade manipulada na aplicação. As definições do processo são registradas pelo analista ou desenvolvedor durante a modelagem, com informações relacionadas ao domínio da aplicação. No início da execução do software, tais informações são capturadas e mapeadas para o banco de dados do EasyBPMS. Assim, ao obter a definição do processo, o observador de início de processo delega ao BPMS a função de iniciar o fluxo de negócio.

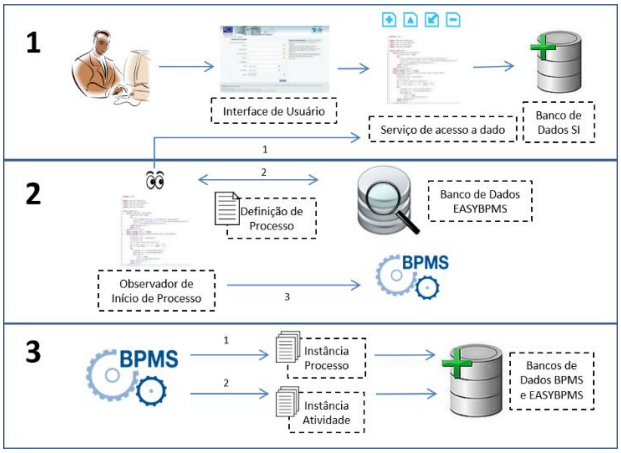

Figura 1. Fluxo de Execução Passos 1, 2 e 3

Para iniciar o fluxo, o BPMS cria uma instância do processo no Passo 3. A mesma instância processo é enviada e criada no EasyBPMS. Após iniciar o processo, o motor BPMS executa o fluxo até a primeira atividade de usuário. Isso ocorre porque a execução desse tipo de atividade depende de interação humana. Ao parar na atividade de usuário, uma instância dela é criada no BPMS e no EasyBPMS.

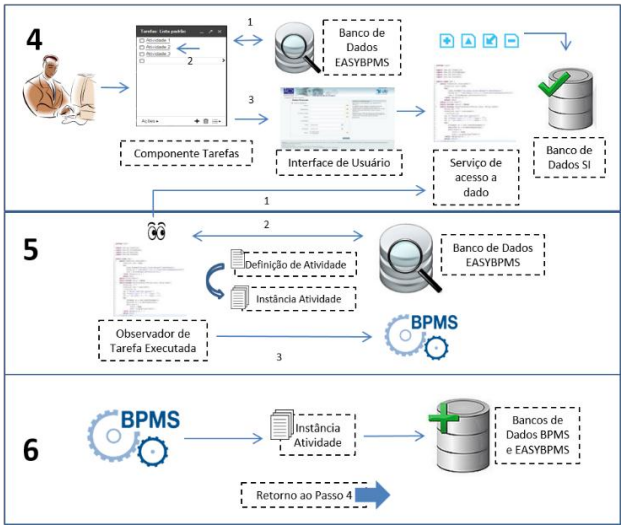

Figura 2. Fluxo de Execução Passos 4, 5 e 6

No Passo 4, o usuário da aplicação, ao acessar o sistema, interage com suas instâncias atividades pendentes no BPMS, listadas por meio de um componente chamado de componente tarefas. O componente tarefas é um componente da abordagem de integração, embutido na interface de usuário, que apenas apresenta uma lista de atividades pendentes, com links que direcionam o usuário para a tela de registro da execução dessa atividade do processo. Ou seja, para uma tela que permita executar um serviço de acesso a dados. Para que o motor de processos tenha conhecimento da atividade executada no SI e continue o fluxo, um observador da tarefa executada é notificado. Esse observador é um componente da

abordagem EasyBPMS que é associado a uma determinada definição de atividade.

No Passo 5, o observador de tarefa executada notificado acessa o banco de dados do EasyBPMS para buscar a definição de atividade compatível com a entidade manipulada na aplicação. A partir da definição de atividade e de informações da entidade de domínio, a instância atividade que está parada no BPMS é capturada. Os dados necessários para executar a tarefa, que foram preenchidos pelo usuário da aplicação, são inseridos para a instância atividade identificada e enviados ao BPMS.

A partir da atividade executada, o BPMS continua o fluxo do processo no Passo 6 e para na próxima atividade de usuário. Nesse ponto, uma instância da atividade é registrada tanto no banco de dados do BPMS quanto no banco de dados do EasyBPMS. Para executar essa atividade, precisa novamente da interação do usuário. Assim, retorna ao Passo 4 e esse fluxo continua até o término do processo.

3. ARQUITETURA EASYBPMS

O projeto da abordagem EasyBPMS proposta consiste de uma arquitetura de alto nível, que engloba os componentes apresentados nos passos da Seção 2. Tal arquitetura é composta de três módulos: *EasyBPMS Core*, *Pluggables e Application*, conforme pode ser visualizada na Figura 3.

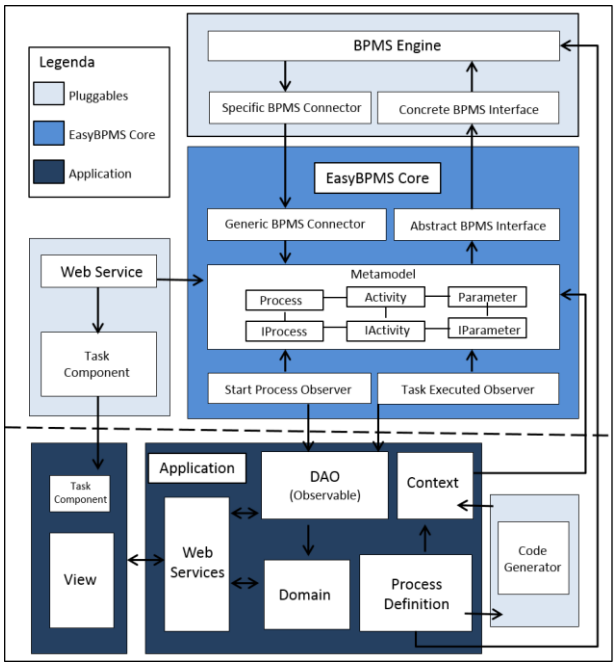

Figura 3. Arquitetura EasyBPMS

O *EasyBPMS Core* abrange componentes genéricos e abstratos, que podem ser utilizados na integração com diferentes BPMSs. Já os *Pluggables* são componentes específicos, necessários durante a comunicação com um determinado BPMS e com uma aplicação de software. Por fim, o *Application* engloba alguns componentes do SI que possibilitam a orientação a processos.

O módulo *EasyBPMS Core* é composto de um núcleo (*Metamodel*) e outros componentes que interagem com o BPMS e com o Sistema de Informação. O núcleo do *EasyBPMS Core* representa um modelo com os metadados do processo. Ou seja, com as definições do processo, das atividades, parâmetros e respectivas instâncias obtidas a partir da execução do processo. Além disso, contém o mapeamento de usuários e grupos de usuários para alocação das atividades. Esse metamodelo tem como base o modelo de metadados representado em [3] e o modelo de dados unificado representado em [5]. Os componentes que comunicam o núcleo com o BPMS e com o SI são:

- ***Start Process Observer*** – componente que é notificado quando entidades de domínio associadas ao início do processo são manipuladas no Sistema de Informação.
- ***Task Executed Observer*** – componente que é notificado quando entidades de domínio associadas à atividade de usuário do processo são manipuladas no Sistema de Informação.
- ***Generic BPMS Connector*** – conector genérico que armazena no *EasyBPMS Core* as instâncias criadas internamente no BPMS.
- ***Abstract BPMS Interface*** – interface que padroniza a comunicação do *EasyBPMS Core* com diferentes BPMSs.

Os *Pluggables* representam componentes específicos e substituíveis na integração. Tais componentes são:

- ***Specific BPMS Connector*** – conector específico, chamado pelo motor BPMS, que cria as instâncias no banco de dados do BPMS.
- ***Concrete BPMS Interface*** – recebe do *EasyBPMS Core* as informações necessárias para iniciar o processo e executar as tarefas de usuário no BPMS.
- ***Task Component*** – componente embutido na aplicação que lista as atividades pendentes de um determinado usuário por meio de *links*, no qual, cada *link* direciona para a tela de gerenciamento da entidade de domínio correspondente.
- ***Web Service*** – componente que busca no *EasyBPMS Core* as tarefas referentes ao usuário logado, que estão pendentes no BPMS, e lista-as no *Task Component*.
- ***Code Generator*** – responsável por capturar as definições do modelo de processo e gera-las automaticamente em um arquivo denominado *Context*.

Com o *EasyBPMS Core* e os *Pluggables*, não é mais necessário implementar a execução das atividades de usuário de um processo de negócio diretamente no SI. O esforço é voltado para o domínio da aplicação e para a modelagem do processo. Com base nisso, os seguintes componentes são definidos para o SI:

- ***Process Definition*** – modelagem do processo de negócio, incluindo as atividades de usuário e complementado com informações do domínio da aplicação, por exemplo, variáveis do processo e parâmetros de entrada e saída de cada atividade.
- ***Context*** – arquivo gerado automaticamente a partir do modelo de processo, com as definições de processo, atividades, parâmetros, grupos de usuário e mapeamento de observadores.
- ***Domain*** – contém as classes de domínio da aplicação.
- ***DAO*** – contém as classes que implementam os serviços de acesso a dados e que notificam os observadores.
- ***View*** – contém as classes que representam a interface do usuário bem como o componente tarefas embutido.
- ***Web Services*** – permite a comunicação da interface de usuário com o domínio da aplicação.

A estrutura do Sistema de Informação proposta na arquitetura EasyBPMS é baseada no padrão arquitetural MVC, pois muitos projetos de sistema seguem esse padrão de *design* [10]. Dentre os componentes, o *Domain* corresponde à camada de negócio, o *View* está relacionado à camada de apresentação, os *Web Services* representam nesse contexto o controlador, e o *DAO* está associado com as classes de persistência da aplicação.

4. APOIO COMPUTACIONAL

Para automatização da abordagem e uso na prática, foi implementado um apoio computacional em Java. Mais especificamente, uma API, denominada API EasyBPMS que segue a arquitetura descrita na Seção 3. Dentre os componentes propostos, foram implementados o *EasyBPMS Core*, os componentes necessários para geração do arquivo *Context* e os componentes específicos (*pluggables)* de um determinado BPMS.

Na implementação do *EasyBPMS Core*, o modelo de domínio corresponde ao componente *Metamodel*, conforme pode ser visualizado na Figura 4. A interface *Abstract BPMS Interface* foi definida com base no padrão de projeto *Adapter* [6], e contém os métodos para iniciar o BPMS, iniciar o processo e executar as tarefas de usuário. Já o *Generic BPMS Connector* é um componente genérico que recebe os dados referentes à execução do processo e instancia as classes *ProcessInstance*, *ActivityInstance* e *ParameterInstance* do metamodelo do *EasyBPMS Core*.

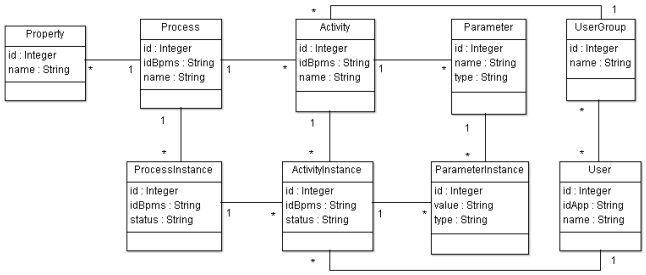

Figura 4. Diagrama de classes do componente Metamodel

Para obtenção dos eventos da aplicação que permitem a execução do fluxo do processo, os componentes *Start Process Observer* e *Task Executed Observer* foram implementados utilizando o padrão de projeto *Observer* [6]. Quando o *Start Process Observer* é notificado, a instância de processo correspondente à entidade de domínio manipulada na aplicação é recuperada ou criada no *EasyBPMS Core*. Após isso, o observador comunica com o BPMS, por meio da interface *Abstract BPMS Interface*, para iniciar o processo caso ele não tenha sido iniciado ainda. Já quando o *Task Executed Observer* é notificado, a instância atividade pendente e correspondente à entidade de domínio é capturada e enviada ao BPMS por meio da interface *Abstract BPMS Interface* para execução da tarefa. Na implementação proposta, os parâmetros de saída da atividade correspondem aos atributos da entidade de domínio atualizados na aplicação.

A implementação proposta depende de um BPMS para ser totalmente funcional. Dessa forma, foram implementados os componentes plugáveis (*Specific BPMS Connector* e *Concrete BPMS Interface*) específicos do BPMS jBPM versão 6.3.0. Final. O motor de processos jBPM foi escolhido por ser próximo do desenvolvedor, ter disponibilidade de acesso e contar com uma documentação e comunidade abrangente [1][14]. O componente *Specific BPMS Connector* é chamado quando o motor de processo inicia o processo ou alcança uma atividade de usuário. Ele é responsável por instanciar as classes relacionadas a instância do processo, atividades e parâmetros do BPMS e envia-las ao *Generic BPMS Connector*. Já o *Concrete BPMS Interface* consiste de uma implementação da interface *Abstract BPMS Interface*, em que os métodos para iniciar o BPMS, iniciar o processo e executar as tarefas de usuário são específicos do BPMS jBPM.

Finalmente, para capturar as informações modeladas nos processos de negócio, foi desenvolvido o plugável *Code Generator* (arquivo executável em Java). Esse arquivo lê os processos de negócio modelados pela ferramenta BPMN2 *Modeler* versão 1.2.4 (*Mars*) e gera o arquivo Java *Context* no pacote `com.easybpms.codegen` do SI. A ferramenta BPMN2 *Modeler* é um *plugin* para o Eclipse e foi selecionada neste trabalho por ser baseada na notação BPMN e ter vínculo com o jBPM, que é o BPMS escolhido para integração. A notação BPMN é um padrão de modelagem de processos e é utilizada por diferentes motores de processo [11].

A entrada do plugável *Code Generator* é um arquivo XML contendo as informações modeladas no fluxo de negócio. A partir do modelo XML, os elementos do processo são estruturados em uma árvore DOM (*Document Object Model*) e mapeados para uma estrutura de dados em Java. Tal estrutura é submetida a um gerador (componente *Velocity* do Java) para geração do arquivo *Context*. Nesse arquivo, as classes *Process*, *Property*, *Activity*, *Parameter e UserGroup* do metamodelo do *EasyBPMS Core* são instanciadas.

5. EXEMPLO DE USO

Para exemplificar o uso da API EasyBPMS, considere o processo de solicitação de pagamento representado na Figura 5. Nesse processo, um funcionário solicita um pagamento e o setor financeiro aprova ou não o pagamento requisitado. Embora a tarefa Solicitar Pagamento seja executada por um usuário, ela foi modelada no processo como uma tarefa de *script*, porque é considerada uma atividade que inicia o processo.

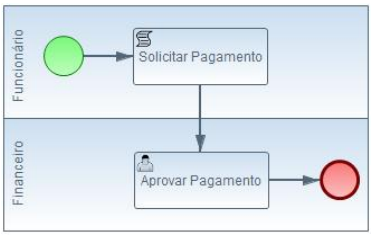

Figura 5. Processo de Negócio Solicitar Pagamento

Para representar os dados registrados durante o fluxo do processo, foi criado na aplicação a classe de domínio `Pagamento`, como pode ser visualizada na Listagem 1.

```
1  public class Pagamento {
2     public long id;
3     public String funcionario;
4     public float pagamento;
5     public boolean aprovar;
6     //métodos getters and setters
7  }
```

Listagem 1. Classe `Pagamento` do Sistema de Informação

Na modelagem, a classe `Pagamento` é definida como id do processo, como pode ser visualizado na Figura 6. Com isso, um observador de início de processo pode ser mapeado para ela no *Context* e ser notificado quando um pagamento for solicitado na aplicação. Os atributos da classe Pagamento que armazenam valores durante o fluxo foram definidos como variáveis do processo, como pode ser visualizado na Figura 7. Tanto o id do processo quanto o nome das variáveis e parâmetros devem ser seguidos do nome do pacote. Essa nomenclatura é necessária para que os observadores do *EasyBPMS Core* detectem pela API de reflexão os valores do processo obtidos em tempo de execução.

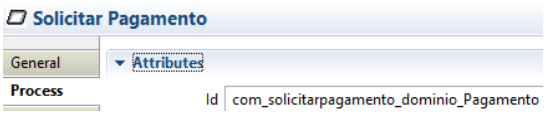

Figura 6. Definição da classe `Pagamento` na modelagem do processo

Figura 7. Definição das variáveis do processo Solicitar Pagamento

Na atividade Aprovar Pagamento, o usuário responsável pelo setor financeiro deve aprovar ou não o pagamento solicitado. A execução dessa atividade gera uma alteração no atributo `aprovar` da classe `Pagamento`. Portanto, esse atributo é modelado como parâmetro de saída da atividade, conforme apresentado na Figura 8. Além disso, a classe `Pagamento` é definida no parâmetro de entrada da atividade Aprovar Pagamento. Com isso, um observador de tarefa executada pode ser mapeado para ela no *Context* e ser notificado quando um pagamento for aprovado na aplicação. O parâmetro de entrada `id` é necessário para que o observador *Task Executed Observer* detecte a instância da classe `Pagamento` responsável pela execução da tarefa, bem como a instância atividade associada à atividade Aprovar Pagamento que está pendente no BPMS.

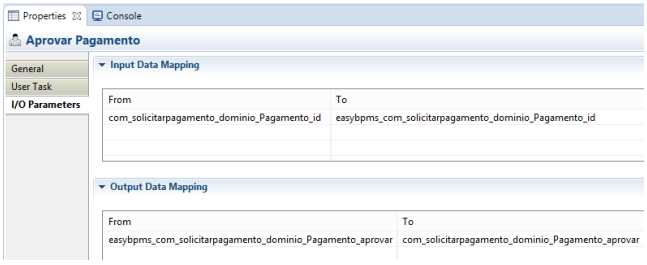

Figura 8. Mapeamento dos parâmetros de entrada e saída da atividade Aprovar Pagamento

Após a modelagem, os dados do processo bem como o mapeamento dos observadores são gerados automaticamente no arquivo *Context*. Na Listagem 2, é apresentada parte desse arquivo, onde o mapeamento de observável e observadores pode ser visualizado. Nesse exemplo, tanto o observador de início de processo quanto o observador de tarefa executada observam a classe `CRUDPagamento` (linhas 5 e 10). Isso ocorre porque a entidade Pagamento é a única gerenciada durante todo o fluxo do processo. Além disso, como a atividade Aprovar Pagamento (com idBpms igual a `UserTask_1`) é a única atividade de usuário do processo, somente um observador de tarefa executada é mapeado no arquivo *Context* (linha 8). Caso houvesse outras atividades de usuário, seria criado um observador para cada tarefa.

```
1  listObservers = new ArrayList<Observer>();
2  //Observador de início de processo
3  StartProcessObserver spo = new StartProcessObserver
      ("com_solicitarpagamento_dominio_Pagamento");
4  listObservers.add(spo);
5  addMapping("CRUDPagamento", listObservers);
6  listObservers = new ArrayList<Observer>();
7  //Observadores de tarefas
8  TaskExecutedObserver teo = new
          TaskExecutedObserver("UserTask_1");
9  listObservers.add(teo);
10 addMapping("CRUDPagamento", listObservers);
```

Listagem 2. Parte do *Context* gerado automaticamente para o processo Solicitar Pagamento

Na implementação do Sistema de Informação, é necessário somente que o desenvolvedor notifique os observadores quando a manipulação das entidades de domínio gera alguma alteração no processo. Na Listagem 3, por exemplo, quando um pagamento é criado ou atualizado a partir da classe `CRUDPagamento`, um processo pode ser iniciado ou uma tarefa executada no BPMS. Portanto, basta somente notificar os observadores (`notifyObservers`) (linhas 3 e 6), enviando a instância do pagamento gerenciada. O método `notifyObservers` pertence à classe `CRUDObservable` da API EasyBPMS.

```
1  class CRUDPagamento extends CRUDObservable{
2     void create (Pagamento p){
3         notifyObservers(p);
4     }
5     void update (Pagamento p){
6         notifyObservers(p);
7     }
8     //Outros métodos
9  }
```

Listagem 3. Classe `CRUDPagamento` do Sistema de Informação

Em suma, com a utilização da API EasyBPMS, não foi necessário implementar no Sistema de Informação a comunicação com o BPMS. Mais especificamente, não foi necessário implementar o código para criar e buscar as instâncias de processo e atividades, bem como para enviar requisições ao motor para que ele inicie o processo ou execute alguma tarefa de usuário. Além disso, o gerenciamento da entidade Pagamento faz parte do domínio da aplicação. Dessa forma, essa entidade precisaria ser manipulada mesmo sem o uso do BPMS.

6. AVALIAÇÃO

Para avaliar a abordagem EasyBPMS proposta, foram comparadas três implementações de um mesmo sistema. A primeira se baseia na API nativa fornecida pelo BPMS jBPM. A segunda se baseia em NextFlow. O NextFlow é um *framework* de mapeamento de processos de negócio e objetos cujo objetivo também é facilitar a integração de BPMSs e Sistemas de Informação, mas com uma diferença importante na abordagem: enquanto o EasyBPMS tem foco no mapeamento de elementos do domínio no modelo de processo, o NextFlow realiza a mesma solução por meio do mapeamento de forma facilitada em código da aplicação. Por fim, a terceira implementação se baseia em EasyBPMS. Nessa comparação, são destacadas as principais diferenças de implementação utilizando cada uma das abordagens e discutido como o uso direto de um BPMS pode ser custoso.

A avaliação proposta é baseada no modelo de avaliação de Oliveira e Valente [9] e foi dividida em duas partes. Primeiro, é apresentada uma análise qualitativa, onde o código de integração requerido pelas implementações é comparado e discutido. Posteriormente, é apresentada uma análise quantitativa que revela o quanto de linhas de código, classes e bibliotecas foram gastas em cada implementação.

6.1 Sistema Proposto

O sistema proposto na avaliação é um sistema de registro de ocorrências, denominado Fixwo, que permite o reporte de problemas. Em suma, um usuário registra por meio de geolocalização uma determinada ocorrência, por exemplo, buraco na rua. O sistema verifica se a área da ocorrência pertence a algum cliente registrado, por exemplo, o buraco se encontra em um munícipio cuja Prefeitura é um cliente. Se houver cliente registrado, a ocorrência se torna uma ordem de serviço e é enviada para análise. A Figura 9 mostra o processo de negócio Fixwo, modelado na ferramenta BPMN2 *Modeler*.

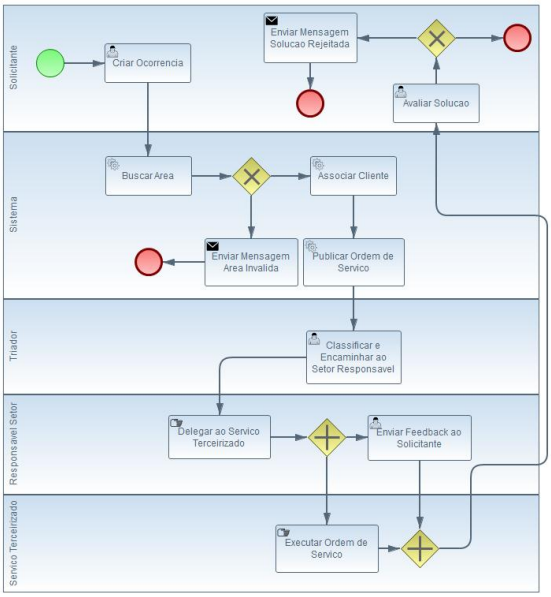

Figura 9. Processo de Negócio Fixwo

6.2 Comparação API Nativa jBPM, NextFlow e EasyBPMS

Nesta seção, são comparadas as implementações das três abordagens propostas. Basicamente, são apresentados o código utilizado para conectar com um BPMS, iniciar um processo e executar tarefas de usuário.

Criando uma conexão: Para a interação com um mecanismo de processos, uma conexão deve ser criada. Na implementação jBPM (Listagem 4), é necessário a configuração de uma sessão (Linha 4) e o envio de uma base de conhecimento contendo as definições dos processos (Linha 1). Uma vez configurada a sessão, ela pode ser usada para iniciar e obter processos em execução.

```
1  RuntimeEnvironment builder =
       RuntimeEnvironmentBuilder.Factory.get()
       .newDefaultInMemoryBuilder()
       .addAsset(ResourceFactory
       .newClassPathResource("fixwoProcess.bpmn2"),
       ResourceType.BPMN2);
2  RuntimeManager manager =
       RuntimeManagerFactory.Factory.get()
       .newSingletonRuntimeManager(builder.get());
3  RuntimeEngine engine =
       manager.getRuntimeEngine(EmptyContext.get());
4  KieSession ksession = engine.getKieSession();
5  TaskService taskService = engine.getTaskService();
```

Listagem 4. Conectando ao motor jBPM

Na implementação NextFlow, a conexão é representada pela interface `WorkflowObjectFactory`, conforme apresentada na Listagem 5. Nessa conexão, são enviadas as definições de processo (Linha 2) e classes de *callback* (Linha 3). As classes de *callback* são componentes que devem ser criados para utilização do NextFlow. Nessas classes, para cada tarefa do processo, um método contendo o seu comportamento é implementado. Exemplos podem ser visualizados em Oliveira e Valente [9].

```
1  WorkflowObjectFactory factory;
2  Configuration configuration = new Configuration
       ("jwfc:jbpm:fixwoProcess.bpmn2");
3  configuration.addCallbackClass(FixwoCallback.class);
4  factory = configuration.createFactory();
```

Listagem 5. Conectando ao NextFlow

Em EasyBPMS, a conexão é obtida por meio da classe `AbstractContext`, conforme mostra a Listagem 6. O método `getContext` é responsável por inicializar o *Context* e o método `connect` é responsável pela conexão. Durante a inicialização do *Context*, os arquivos de processo são capturados, os observadores são mapeados e o metamodelo *EasyBPMS Core* apresentado na Seção 3 é preenchido com as definições do processo.

```
1  AbstractContext.getContext().connect();
```

Listagem 6. Conectando ao EasyBPMS

Uma desvantagem da implementação jBPM em relação às implementações NextFlow e EasyBPMS é a especificidade de integração. A conexão é única para cada BPMS, não existe uma interface padrão de comunicação. Assim, caso o BPMS mude, a Listagem 4 deve ser alterada. Tanto em NextFlow e EasyBPMS, existe uma interface padrão de comunicação com um BPMS. A implementação dessa interface pode ser desenvolvida fora do contexto do Sistema de Informação.

Iniciando um processo: No sistema Fixwo, o início do processo ocorre quando um usuário solicitante registra uma ocorrência. As Listagem 7 e Listagem 8 apresentam o método `startNewFixwoProcess` nas implementações jBPM e NextFlow respectivamente. Na implementação jBPM, a instância de processo é armazenada em um objeto genérico `ProcessInstance` (Linha 2). Já na implementação Nextflow, a instância de processo retornada está associada a uma interface específica do processo, denominada `FixwoProcess`. Essa interface é outro componente que deve ser implementado para uso do NextFlow. Nessa interface, cada método está associado a uma tarefa de usuário. Exemplos podem ser visualizados em Oliveira e Valente [9].

```
1  ProcessInstance startNewFixwoProcess() {
2      ProcessInstance processInstance = ksession
           .startProcess("org_fixwo_domain_Ocorrencia");
3      return processInstance;
4  }
```

Listagem 7. Iniciando um processo em jBPM

```
1  FixwoProcess startNewFixwoProcess() {
2      return factory.start(FixwoProcess.class);
3  }
```

Listagem 8. Iniciando um processo em NextFlow

Na API EasyBPMS, não é necessária a implementação do método `startNewFixwoProcess`. Ao registrar uma ocorrência, o observador de início de processo é notificado. Ele, por sua vez, invoca o motor BPMS para que o processo possa ser iniciado. O desenvolvedor precisa somente definir a classe `Ocorrencia` como id do processo na modelagem e notificar os observadores quando uma ocorrência for criada, seguindo o mesmo exemplo da Seção 5.

Executando tarefas de usuário: A tarefa Avaliar Solução é uma tarefa de usuário, onde o usuário Solicitante avalia a solução da ocorrência registrada. Ao executar essa atividade no SI, o atributo `avaliacao` da entidade Ocorrência é alterado. A Listagem 9 mostra o código que executa essa tarefa na implementação jBPM.

```
1  void executeUserTask(Ocorrencia ocorrencia,TaskService
   taskService, Task task) {
2      Map <String,Object> results =
               new HashMap <String,Object>();
3      results.put("out_avaliacao",
               ocorrencia.getAvaliacao());
4      taskService.start(task.getId(), "Administrator");
5      taskService.complete(task.getId(),
               "Administrator", results);
6  }
```

Listagem 9. Executando uma tarefa de usuário em jBPM

Primeiramente, o valor do atributo `avaliacao` é inserido em um mapa de resultados (Linhas 2 e 3). A chave do mapa (`out_avaliacao`) corresponde ao parâmetro de saída da atividade Avaliar Solução. Após o mapeamento de resultados, a tarefa é iniciada e completada (Linhas 4 e 5). Na implementação NextFlow, o método `executeUserTask`, responsável pela execução da tarefa Avaliar Solução, pode ser visualizado na Listagem 10. O comportamento da tarefa é implementado na classe de *callback* `FixwoCallback`, conforme apresentado na Listagem 11.

```
1  void executeUserTask(Ocorrencia ocorrencia,
   FixwoProcess fixwoProcess) {
2      fixwoProcess.avaliarSolucao
           (ocorrencia.getAvaliacao());
3  }
```

Listagem 10. Executando uma tarefa de usuário em NextFlow

```
1  void avaliarSolucao(Boolean avaliacao){
2     ocorrencia.setAvaliacao(avaliacao);
3  }
```

Listagem 11. *Callback* que fornece o comportamento da tarefa de usuário Avaliar Solução em NextFlow

Na implementação EasyBPMS, não é necessário implementar o método `executeUserTask` como nas outras implementações, uma vez que a execução das tarefas de usuário é capturada por observadores. O desenvolvedor precisa somente definir o parâmetro de entrada `id` e o parâmetro de saída `avaliacao` na modelagem e notificar os observadores quando a ocorrência for atualizada, seguindo o mesmo exemplo da Seção 5.

A captura das tarefas de serviço, manuais e de envio não está no escopo da abordagem EasyBPMS, devido às diferentes possibilidades de acesso a esses tipos de tarefas. Em outras palavras, a definição e execução das tarefas de serviço, manuais e de envio podem variar de acordo com BPMS utilizado. Portanto, a busca por uma generalização e padronização se torna exaustiva. Com base nisso, para permitir a execução dessas tarefas no processo Fixwo, foi utilizado a mesma abordagem definida na implementação jBPM. Nessa implementação utiliza-se caixas de propriedade do processo. Dessa forma, não foi necessário adicionar código na aplicação para execução dessas tarefas no BPMS.

6.3 Análise Quantitativa

O sistema Fixwo foi submetido a uma análise quantitativa, a fim de comparar o número de linhas de código (LOC), classes e dependências e/ou bibliotecas gastos em cada implementação. Para contabilizar a medida LOC e o número de classes, foi utilizado o *plugin* do Eclipse, denominado *Eclipse Metrics Plugin*. A Tabela 1 resume os resultados obtidos.

Tabela 1. Métricas contabilizadas nas implementações do sistema Fixwo

Métricas	EasyBPMS	NextFlow	Delta	jBPM	Delta
LOC	107	291	-63,2%	296	-63,8%
Classes	8	16	-50%	13	-38,4%
Dependências	12	16	-25%	10	+16,6%

Na contagem de LOC, a implementação EasyBPMS requer 63,2% menos linhas em relação ao NextFlow e menos 63,8% em relação ao jBPM. Na contagem do número de classes, a implementação NextFlow requer 8 classes a mais e o jBPM requer 5. Essa diferença ocorre devido à adição das classes para execução das tarefas de usuário e as classes de conexão com o BPMS. Por fim, para a implementação jBPM foram adicionadas 10 dependências. Já nas implementações EasyBPMS e NextFlow, foi necessário a adição das suas respectivas bibliotecas e as dependências jBPM.

A partir da análise realizada, pode-se concluir que a abordagem EasyBPMS permitiu uma integração com menos linhas de código em comparação com jBPM e NextFlow e, consequentemente, um menor esforço de programação durante a integração. Além disso, nos cenários onde o BPMS é um dos componentes do SI, quanto menos código, classes e bibliotecas utilizadas na integração, menos gastos na manutenção do sistema orientado a processos.

Em suma, as análises propostas nesta avaliação tiveram como foco principal mostrar como e quanto o trabalho de um desenvolvedor pode diminuir, de forma que a maior parte da integração seja realizada no modelo de processo e não durante o desenvolvimento.

7. AMEAÇAS A VALIDADE

A avaliação proposta foi realizada com apenas um sistema exemplo. Com isso, os resultados obtidos não podem ser generalizados para todos os sistemas integrados a BPMSs. No entanto, o processo de negócio utilizado na avaliação foi modelado com os componentes base da notação BPMN (definidos na Tabela 7.1 do guia BPMN, versão 2.0.2) e executado com êxito no jBPM.

A comparação com apenas um BPMS constitui uma ameaça de validade externa, uma vez que outros BPMSs podem ter características diferentes. No entanto, mesmo considerando que a avaliação do EasyBPMS utilizou uma implementação para jBPM, o trabalho mostra que é possível diminuir a quantidade de linhas de código gastas durante a integração com o BPMS.

A execução do sistema exemplo não foi realizada em um ambiente de produção. Em vez disso, a execução foi simulada a fim de recolher os eventos do processo. No entanto, se o sistema Fixwo fosse executado em um ambiente real, as classes de integração com o jBPM em todas as abordagens seriam praticamente as mesmas.

O desenvolvimento do sistema Fixwo para os três projetos é subjetivo, ou seja, existem diferentes formas de implementação. Com isso, a quantidade de classes e linhas de código geradas podem ter influenciado a análise quantitativa. No entanto, em defesa, o sistema foi desenvolvido de forma modular, a fim de obter um menor nível de acoplamento.

8. TRABALHOS RELACIONADOS

Oliveira e Valente [9] propõem um *framework*, chamado NextFlow, para mapeamento de elementos de processos de negócios em elementos orientados a objetos, a fim de facilitar a integração entre BPMSs e SI. Já na abordagem EasyBPMS proposta, não é necessário o mapeamento de elementos do processo de negócio diretamente no código-fonte do SI. Esse mapeamento é realizado no modelo do processo e gerado automaticamente. Dessa forma, as chamadas ao BPMS para início do processo e execução das atividades de usuário são detectadas no SI de forma implícita, abstraindo mais a integração no código do sistema.

Brambilla et al. [3] propuseram uma metodologia dirigida a modelos para especificação, modelagem e implementação de processos de negócio. Em uma das etapas de transformação, o modelo BPMN é detalhado com informações do domínio da aplicação, assim como proposto em EasyBPMS. No entanto, enquanto que no EasyBPMS é utilizada a notação BPMN pura, na metodologia de Brambilla et al. [3], a notação BPMN precisou ser estendida para aumentar a semântica do modelo de processo. Além disso, a metodologia proposta por Brambilla et al. [3] foi implementada como uma extensão do BPMS WebRatio. Já a abordagem EasyBPMS foi implementada como uma API genérica e reutilizável para captura e execução das atividades de usuário em diferentes BPMSs.

Keeratichayakorn e Maneeroj [8] propõem um *framework* para integração de aplicações com diferentes BPMS, permitindo maior reuso, flexibilidade e facilidade de manutenção. O *framework* foi desenvolvido utilizando os padrões de projeto *Bridge*, *Decorator* e *Factory*. No entanto, apesar desses padrões, desenvolvedores ainda precisam manipular elementos do processo, como o envio de dados para completar as tarefas no BPMS. Já na abordagem EasyBPMS, os eventos de interação com o processo são manipulados pela API EasyBPMS. Dessa forma, o acesso a interfaces de processo por parte dos desenvolvedores não é necessário.

9. CONCLUSÃO

Neste artigo, foi apresentado EasyBPMS, uma abordagem de integração entre SI e BPMS, que tem como foco a abstração da execução das atividades de usuário. Como projeto e implementação dessa abordagem foram propostas respectivamente uma arquitetura de referência e uma API. Além disso, foram comparadas três implementações de um SI, usando a API do jBPM, NextFlow e EasyBPMS, e realizadas análises qualitativa e quantitativa.

Com a API EasyBPMS, o foco está na anotação de modelos de processo. Assim, a manipulação direta de elementos de baixo nível provenientes de um dado BPMS é diminuída, permitindo uma menor quantidade de código durante a integração. Além disso, a arquitetura proposta é uma contribuição para o âmbito acadêmico, uma vez que diferentes implementações, baseadas em tecnologias diversas, podem ser geradas utilizando os componentes definidos.

Atualmente, a abordagem EasyBPMS tem foco somente nas atividades de usuário e o protótipo de implementação está disponível apenas para sistemas Java. Além disso, a comparação somente por LOC é uma limitação. Componentes com poucas linhas podem ter, por exemplo, alta complexidade ciclomática. Como trabalho futuro, pretende-se fornecer suporte a outras linguagens e BPMSs, bem como avaliar o uso da abordagem em um ambiente real. Além do mais, coletar outras métricas em análises futuras e realizar a avaliação com pessoas, a fim de verificar o tempo gasto de aprendizagem para cada abordagem.

A API EasyBPMS e as três implementações do sistema Fixwo estão disponíveis publicamente e podem ser acessadas por meio do link github.com/easybpms.

10. AGRADECIMENTOS

Os autores agradecem pelo suporte obtido da Capes, CNPq e Fapemig.

11. REFERÊNCIAS

[1] Baïna, K., and Baïna, S. 2013. User experience-based evaluation of open source workflow systems: The cases of Bonita, Activiti, jBPM, and Intalio. In *Proceedings of the 3th International Symposium ISKO-Maghreb* (Marrakech, Morocco, Nov. 08 - 09, 2013), 1-8. DOI = 10.1109/ISKO-Maghreb.2013.6728122.

[2] Bouchelligua, W., Mahfoudhi, A., Mezhoudi, N., Daassi, O., and Abed, M. 2010. User interfaces modelling of workflow information systems. In *Proceedings of the 6th International Workshop on Enterprise and Organizational Modeling and Simulation* (Hammamet, Tunisia, Jun. 07-08, 2010), 143-163. DOI = 10.1007/978-3-642-15723-3_10.

[3] Brambilla, M., Dosmi, M., and Fraternali, P. 2009. Model-driven engineering of service orchestrations. In *Proceedings of the 2009 Congress on Services-I* (Los Alamitos, USA, Jul. 06-10, 2009), 562-569. DOI = 10.1109/SERVICES-I.2009.94.

[4] Cardoso, J., Bostrom, R. P., and Sheth, A. 2004. Workflow management systems and ERP systems: Differences, commonalities, and applications. *Information Technology and Management*. 5, 3-4 (Jul. 2004), 319–338. DOI = 10.1023/B:ITEM.0000031584.14039.99.

[5] Delgado, A., Calegari, D., and Arrigoni, A. 2016. Towards a Generic BPMS User Portal Definition for the Execution of Business Processes. *Electronic Notes in Theoretical Computer Science*. 329 (Dec. 2016), 39-59. DOI = 10.1016/j.entcs.2016.12.004

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. *Design patterns: elements of reusable object-oriented software*. Pearson Education, Upper Saddle River, New Jersey, USA, 395p.

[7] Ferreira, D. R., and Thom, L. H. 2012. A semantic approach to the discovery of workflow activity patterns in event logs. *International Journal of Business Process Integration and Management*. 6, 1 (Jul. 2012), 4-17. DOI = 10.1504/IJBPIM.2012.047909

[8] Keeratichayakorn, W., and Maneeroj, S. 2014. Design patterns for integration between enterprise application with any business process management systems. In *Proceedings of the 4th International Conference on Digital Information and Communication Technology and it's Applications* (Bangkok, Thailand, May 06-08, 2014), 7-12. DOI = 10.1109/DICTAP.2014.6821647

[9] Oliveira, R. G., and Valente, M. T. 2014. Object-Business Process Mapping Frameworks: Abstractions, Architecture, and Implementation. In *Proceedings of the 18th International Enterprise Distributed Object Computing Conference* (Ulm, Germany, Sep. 01-05, 2014), 160–169. DOI = 10.1109/EDOC.2014.30.

[10] Pérez-Castillo, R., Weber, B., Guzmán, I. G. R., Piattini, M., and Pinggera, J. 2014. Assessing event correlation in non-process-aware information systems. *Software & Systems Modeling*. 13, 3 (Jul. 2014), 1117-1139. DOI = 10.1007/s10270-012-0285-5

[11] Ruiz, M., Costal, D., España, S., Franch, X., and Pastor, Ó. 2015.GoBIS: An integrated framework to analyse the goal and business process perspectives in information systems. *Information Systems*.53 (Apr. 2015), 330-345. DOI = 10.1016/j.is.2015.03.007

[12] Suri, K., and Mos, A. C. 2015. Human Task Monitoring and Contextual Analysis for Domain-Specific Business Processes. In *Proceedings of the 26th International Conference on Software & Systems Engineering and their Applications* (Paris, France, May 27-29, 2015), 27-29.

[13] Vukšić, V. B., Brkić, L., and Baranović, M. 2016. Business Process Management Systems Selection Guidelines: Theory and Practice. In *Proceedings of the 39th International Convention Information and Communication Technology, Electronics and Microelectronics* (Opatija, Croatia, May 30 – June 3, 2016), 1476-1481.

[14] Wohed, P., Russell, N., Ter Hofstede, A. H., Andersson, B., and van der Aalst, W. M. 2009. Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark. *Information and Software Technology*. 51, 8 (Aug. 2009), 1187–1216. DOI = 10.1016/j.infsof.2009.02.002