WED-SQL (Rascunho)

Bruno Padilha, João E. Ferreira
Departamento de Ciência da Computação
IME-USP
Rua do Matão 1010, 05508-090
São Paulo, SP, Brasil
brunopadilha@usp.br, jef@ime.usp.br

Calton Pu
CERCS, Georgia Institute of Technology
266 Ferst Drive, 30332-0765
Atlanta, GA, USA
calton@cc.gatech.edu

***Resumo*—Desenvolver e implementar um modelo de processo de negócios utilizando os conceitos do WED-flow resulta em um sistema de informação dinâmico, simplificando o desenvolvimento incremental e adaptativo, além de reduzir significativamente a complexidade no tratamento de exceções quando comparado aos arcabouços tradicionais (BPEL, álgebra de processos, Redes de Petri e etc). No entanto, uma aplicação baseada no WED-flow precisa implementar uma camada de software em um nível abaixo para fazer o controle transacional baseado em estados de dados (WED-states), assim como fornecer suporte ao tratamento de exceções ou utilizar uma ferramenta como a WED-tool gerenciar esses mecanismos. A proposta da WED-SQL é fornecer um arcabouço WED-flow distribuído e de alto desempenho para simplificar tanto o desenvolvimento quanto a implementação de tais aplicações. Algumas vantagens da WED-SQL é que seu mecanismo de controle está integrado a um SGBD relacional e, com isso, utiliza a linguagem SQL para a especificação das definições do WED-flow (WED-triggers, WED-transitions, WED-conditions, etc).**

Keywords*—*WED-flow, PostgreSQL, transações longas, ...

I. Introducão

Sistemas computacionais contemporaneos para o gerenciamento de processos de negócio estão sujeitos a constantes modificaçoes estruturais, ocasionadas tanto por execessões não previstas na fase de modelagem quanto para atender a novos requisitos. Ao longo do tempo, essas modificaçoes tendem a deteriorar a qualidade do código de tais sistemas, aumentando exponencialmente seu custo de manutenção além de, eventualmente, comprometer seu desempenho.

Os modelos clássicos para especificação de processos de negócios, dentre eles o WS-BPEL, algebra de processos e redes de petri, procuram capturar interações, relações e o comportamento entre processos, muitas vezes negligenciando a importância dos dados no projeto. Nenhuma dessas ferramentas, no entanto, é exatamente adequada para modelar processos que necessitam ser modificados frequentemente, o que implica em aumento exponencial no custo e na complexidade do projeto.

O modelo WED-flow[1], como uma alternativa aos modelos clássicos, captura não só a dinâmica da interação entre processos como também, com igual importância, os dados gerados assim como os eventos que implicam em alterações desses dados. Com isso, o WED-flow é capaz de construir um workflow de estados de dados com suas respectivas condições de transição entre os mesmos. E assim, agregando propriedades de recuperação transacional a esse workflow, prover mecanismos para o tratamento de exceções em tempo real.

Ao utilizar a abordagem WED-flow para modelar um processo de negócio, modificações nos processos são traduzidas em novos estados de dados e novas regras, ou novos processos, se traduzem em condições para se atingir um determinado estado de dados. Essa versatilidade permite capturar a natureza de processos verdadeiramente dinâmicos, simplificando a evolução incremental tanto do modelo de negócio quanto do software que o implementa.

Para que um modelo de processo de negócio baseado no WED-flow seja implementado em software, é preciso também implementar as definições do WED-flow[2], que são a base para o tratamento de exceções e para o controle transacional. Ao invés de incorporar essas definições a cada software que implemente um modelo WED-flow, uma maneira mais eficiente de o fazer é construir um arcabouço para padronizar a implementação do WED-flow em software. Nesse contexto, esse trabalho apresenta o arcabouço WED-SQL.

O WED-SQL foi construido dentro de um SGBD relacional e com isso utiliza a linguagem SQL para definir propriedades e controles de fluxo de um modelo WED-flow, além de contar com um robusto aparato transacional e ser altamente tolerante a falhas. Por utilizar como base tecnologias consolidadas e amplamente difundidas no universo da computação, a WED-SQL visa disseminar a adoçao do paradigma WED-flow na modelagem de processos de negócio , fornecendo uma ferramenta confiável, facilmente escalável, simples de ser utilizada e que permita a flexibilidade exigida na especificação de controle de fluxo dos processos de negócio modernos.

O restante desse artigo segue a seguinte estrutura: ...

II. WED-SQL: Visão geral e arquitetura

/* Incluir um resumo das definições do WED-flow ? */
Com o objetivo de ser utilizada em um ambiente distribuido, por exemplo por meio de web-services, e também para contemplar o controle de transações longas [**?**], o WED-SQL foi construido utilizando a arquitetura cliente-servidor. A componente servidor, ou WED-server, é responsável por fazer o controle transacional de acordo com as WED-conditions definidas para um determinado WED-flow, disparando as WED-triggers conforme necessário. Já a componente cliente, ou WED-worker (quem sabe até WED-service ?), é quem efetivamente realiza as WED-transitions.

A. WED-worker e WED-server

O WED-server é, basicamente, uma extensão para o sgbd Postgresql composta de triggers, tabelas de controle e "stored

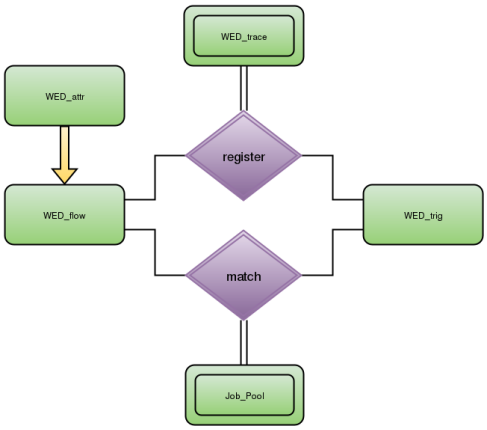

Figura 1. Diagrama ER do WED-server

procedures". A escolha do Postgresql como base do WED-server se deu por diversos motivos, tais quais, ser de código aberto, implemetação modular baseada em catálogs (system catalog driven), facilidade de extender suas funcionalidades por meio de código C e tambem em liguagens de alto nível como Python e Perl. Por ser de código aberto, a licensa do Postgresql garante que o software possa ser modificado e redistribuido, além de que, o acesso ao código fonte possibilita uma melhor compreensão dos mecanismos interno, garantido uma implementação mais robusta e eficiente do paradigma WED-flow. O postgresql armazena seus dados de controle em tabelas que são acessíveis aos usuários, que são chamadas de catálogos de sistema, viabilizando a criação de modulos para extender suas funcionalidades. Dados de transações podem ser facilmente obtidos por meio desses catálogos de sistema, o que é partircularmente útil no caso do WED-server. Dada a natureza dinâmica do modelo WED-flow, a utilização de uma linguagem de programação de alto nível, como Python, é fundamental para que a expressividade desse modelo seja implementada de forma plena. Do ponto de vista de implementação, o WED-server contém código Python, SQL, C e aproveita o arcabouço transacional clássico (propriedades ACID, controle de concorrência e etc) fornecido pelo Postgresql.

Já o WED-worker tem a função de conectar-se ao WED-server para efetuar as WED-transitions. Cada WED-transition é associada a um ou mais WED-workers, dependendo da demanda de trabalho gerada pelo WED-server, e cada WED-worker é especializado em realizar uma WED-transition específica. A quantidade máxima de WED-workers trabalhando simultaneamente é limitada apenas pela quantidade de conecções que um dado WED-server é capaz de manter abertas.

B. Estrutura de dados

O diagrama Entidade-Relacionamento na figura 1 representa o modelo de dados gerenciado internamente pelo WED-server. Vale notar que, devido a flexibilidade de modificaçao da estrutura de dados exigida para representar os WED-states, não é possível capturar todos os aspectos do paradigma WED-flow por meio de um diagrama Entidade-Relacionamento clássico. Por exemplo, o relacionamento entre as entidades WED_attr e WED_flow é gerenciado pelo WED-server, ou seja, não há chaves indicando a relação entre essas duas entidades. A principal razão disso é pertimir adicionar ou remover WED-attributes em tempo de execução, o que demanda modificações na estrutura de dados que, por sua vez, podem ser efetuadas de maneira ininterrupta.

Do relacionamento entre as entidades WED_flow e WED_trig resultam a entidade fraca Job_Pool, que contém as WED-transitions a serem executadas, e a entidade fraca WED_trace, que contém os registros de execução de cada instância dos WED-flows.

C. Tabelas de controle

O WED-server utiliza cinco tabelas para definir regras, propriedades e controlar o fluxo de execução dos WED-flows: WED_attr, WED_flow, WED_trace, WED_trig e Job_pool.

Os WED-attributes são definidos na tabela WED_attr por um nome e, opcionalmente, por um valor padrão representados pelas colunas "aname"e "adv"respectivamente. Cada WED-attribute é identificado univocamente por seu nome, que também é a chave primária da tabela. Ao criar-se um novo WED-attribute, ou seja, ao inserir-se uma nova linha na tabela WED_attr, uma coluna de mesmo nome será automaticamente criada na tabela WED_flow.

A tabela WED_trig contém as WED-triggers, ou seja, cada entrada representa a associação de uma WED-condition com uma WED-transition. Possui os seguintes atributos:

- $tgid$, é o identificador único de uma WED-trigger;
- $tgname$, atributo opcional utilizado para dar um nome à WED-trigger;
- $enabled$, permite que a WED-trigger seja desativada;
- $trname$, identificador único da WED-transition associada.
- $cname$, atributo opcional que pode ser utilizado para dar um nome à WED-condition associada;
- $cpred$, predicado da WED-condition associada. Aceita qualquer predicado valido na claúsula WHERE em SQL;
- $cfinal$, utilizada para indicar qual é a condição final. Embora apenas uma WED-condition possa ser marcada como final, o operado lógico OU pode ser utilizado em seu predicado para definir-se multiplos WED-states finais.
- $timeout$, tempo limite para que um WED-worker finalize a WED-transition representada por "trname".

O histórico de execução das WED-transition fica armazenado na tabela WED_trace, que possui os seguintes atributos:

- wid, referência ao identificador da instância do WED-flow;
- $state$, WED-state, em formato json, que disparou as WED-transitions listadas em "trf";
- trf, lista de WED-transitions disparadas pelo WED-state em "state";
- trw, WED-transition que gravou o WED-state representado por "state". Um valor nulo indica que esse registro representa o WED-state inicial;

```
BEGIN;

INSERT INTO wed_attr (aname, adv) VALUES ('a1','ready');
INSERT INTO wed_attr (aname) VALUES ('a2'),('a3');

INSERT INTO wed_trig (tgname,trname,cname,cpred,timeout)
  VALUES ('t1','tr_a2','c1', $$a1='ready' and (a2 is null)$$, '3d18h');
INSERT INTO wed_trig (tgname,trname,cname,cpred,timeout)
  VALUES ('t2','tr_a3','c2', $$a1='ready' and (a3 is null)$$, '00:00:30');
INSERT INTO wed_trig (tgname,trname,cname,cpred,timeout)
  VALUES ('tf','tr_final','cf', $$a1='ready'
          and (a2 is not null)
          and (a3 is not null)$$, '00:00:10');
INSERT INTO wed_trig (cpred,cfinal) VALUES ($$a1 <> 'ready'$$, True);

COMMIT;
```

Figura 2. Exemplo de definição de um novo WED-flow

- $status$, indica o tipo do WED-state representado em "state". Os possíveis valores são "F","E"ou "R"que indicam que o WED-state é final, excessão ou regular, respectivamente.
- $tstmp$, indica o momento em que ocorreu o registro. Pode se recuperar a história de execução de uma instância ordenando-se as entradas nessa tabela por essa coluna.

A tabela Job_Pool contém entradas referentes as WED-transitions pendentes que precisam ser executadas pelos WED-workers. Suas colunas são:

- wid, referência ao identificador da instancia do WED-flow;
- $tgid$, referência a WED-trigger que disparou a WED-transition "trname";
- $trname$, nome da WED-transition a ser executada;
- $lckid$, parâmetro opcional que pode ser utilizado pelos WED-workers para se identificarem. Futuramente, poderá ser utilizado para fins de autenticação e validação dos WED-workers;
- $timeout$, tempo limite para a execução da WED-transition (tempo limite da transação). É uma cópia da coluna de mesmo nome da tabela WED_trig;
- $payload$, WED-state, em formato json, que disparou a referida WED-transition. Pode ser utilizado, por exemplo, por WED-workers que executam WED-transitions associadas a WED-conditions que possuem o operador lógico "OR"em seu predicado.

Finalmente, a tabela WED_flow é o ponto de entrada para inicializar-se uma nova instância. Essa tabela é criada dinâmicamente de acordo com os WED-attributes definidos na tabela WED_attr. Cada entrada em WED_attr corresponde a uma coluna em WED_flow. Suas entradas são o WED-state atual de cada instância, ou seja, são tuplas formadas por um identificador, representado na coluna "wid", e os WED-attributes.

III. FUNCIONAMENTO

Embora seja possível definir multiplos WED-flows em uma única base de dados, o WED-server permite que cada um deles esteja localizado em uma base de dados distinta, de acordo com um determinado significado semântico. Com isso também é possível isolar atributos que não devem ser compartilhados entre diferentes WED-flows.

```
from BaseWorker import BaseClass
import sys

class MyWorker(BaseClass):

    #  trname and dbs variables are static in order to conform
    #with the definition of wed_trans()

    trname = 'tr_aaa'
    dbs = 'user=aaa dbname=aaa application_name=ww-tr_aaa'
    wakeup_interval = 5

    def __init__(self):
        super().__init__(MyWorker.trname,MyWorker.dbs,MyWorker.wakeup_interval)

    # Compute the WED-transition and return a string as the new WED-state,
    #using the SQL SET clause syntax. Return None to abort transaction
    def wed_trans(self,payload):
        print (payload)

        return "a2='done', a3='ready', a4=(a4::integer+1)::text"
        #return None

w = MyWorker()

try:
    w.run()
except KeyboardInterrupt:
    print()
    sys.exit(0)
```

Figura 3. Exemplo de definição de um novo WED-flow

Para se criar um novo WED-flow é preciso definir um conjunto de WED-atributes e um conjunto de WED-triggers associando WED-transitions à WED-conditions. Essa definição, por ora, é expressa em SQL (futuramente em WSQL). Veja um exemplo na figura 4.

Note que os valores dos predicados para as WED-conditions, representados por meio da coluna "cpred", tem a mesma sintaxe utilizada para expressar as restrições de uma cláusula WHERE em SQL. De fato, a expressão em "cpred"será utilizada as-is pelo WED-server para disparar as WED-transitions. Ao utilizar-se duplo $ para delimitar essa expressão, elimina-se a necessidade de "escapar"as aspas simples ou outros caracteres especiais.

Note também que a condição final de um WED-flow é declarada nessa mesma tabela WED_trig, embora de modo simplificado. São necessários apenas o predicado em "cpred"e o valor Verdade em "cfinal". Caso não haja uma condição final na tabela WED_trig, todas as instâncias desse WED-flow terminarao em um WED-state de excessão.

É recomendado encapsular a definição de um WED-flow em uma única transação, uma vez que, no caso de um erro de sintaxe na definição de uma WED-trigger, por exemplo, seria necessário remover manualmente do WED-server as definições executadas até o momento do erro.

A. *Definindo os WED-workers*

Como mencionado anteriormente, quem executa as WED-transitions disparadas pelo WED-server são os WED-workers. Sendo assim, é necessário criar esses WED-workers e associá-los às respectivas WED-transitions. É recomendado ter ao menos um WED-worker associado a cada WED-transition.

Um WED-worker é uma aplicação cliente do WED-server e, por esse motivo, pode ser escrito em qualquer linguagem de progra- mação que tenha suporte para conectar-se ao sgbd PostgreSQL e que implemente o protocolo de comunicação do WED-server (veja o capitulo ???). No escopo deste trabalho os WED-workers são escritos na linguagem Python utilizando-se o pacote BaseWorker, que acompanha o WED-SQL. A figura 3 ilustra a definição de um novo WED-worker:

O primeiro passo é importar a classe abstrata *BaseClass* do pacote *BaseWorker*. Essa classe implementa a lógica do

protocolo de comunicação com o WED-server e também gerencia as conecções com o mesmo.

O proximo passo é de fato criar o WED-worker, definindo uma nova classe concreta que implemente o método wed_trans() e defina os seguintes atributos utilizados para inicializar a *BaseClass*:

- *trname*: nome da WED-transition que será executada por esse WED-worker. Precisa ser o mesmo utilizado na definicão do WED-flow;
- *dbs*: parametros da conecção com o WED-server no formato aceito pelo driver *psycopg2*. No mínimo devem ser especificados o usuário, o nome da base de dados onde o WED-flow foi carregado e o nome do WED-worker por meio do campo *"application_name"*;
- *wakeup_interval*: é o intervalo de tempo no qual o WED-worker fica suspenso esperando por uma notificação do WED-server (veja capitulo ??);

O método wed_trans() recebe como parametro o WED-state da instância do WED-flow que disparou a transação, e retorna uma string com os novos valores dos WED-attributes dessa mesma instância. A sintaxe utilizada para esse valor de retorno deve ser a mesma utilizada em uma cláusula SET de uma cláusula UPDATE em SQL. O valor "None"pode ser retornado para abortar a transação. Essa classe concreta que implementa o WED-worker deve então ser instanciada e executada, invocando-se o método run().

IV. Algoritmos

Após definir-se e carregar-se um WED-flow em uma base de dados do WED-server e inicializar-se seus respectivos WED-workers, uma nova instância é inicializada inserindo-se um WED-state na tabela WED_flow. Essa nova instância receberá um identificador único que será utilizado tanto para o registro de sua história de execução, na tabela WED_trace, quanto para o controle transacional das WED-transitions. Se os valores padrao definidos para os WED-attributes, por meio da coluna "adv", na tabela WED_attr forem os valores de um WED-state inicial, basta executar o comando abaixo para se inicializar uma nova instância:

```
INSERT INTO wed_flow DEFAULT VALUES;
```

Para cada nova instância inserida na tabela wed_flow, o WED-server irá comparar os predicados das WED-conditions definidos na tabela wed_trig com o WED-state representado na nova instância. Para cada condição satisfeita a respectiva WED-transition será disparada na forma de uma entrada na tabela job_pool e uma notificação será enviada ao WED-worker responsavel por executá-la. Além disso, serão adicionadas entradas na tabela wed_trace registrando os eventos ocorridos. O WED-server ficará então aguardando até que uma nova instância ou uma WED-transition seja iniciada. Vale notar que WED-states iniciais que não sejam finais e que não disparem ao menos uma WED-transition serão rejeitados pelo WED-server.

Na sequencia, os WED-workers podem atuar de dois modos distintos: imediatamente ao receber uma notificação

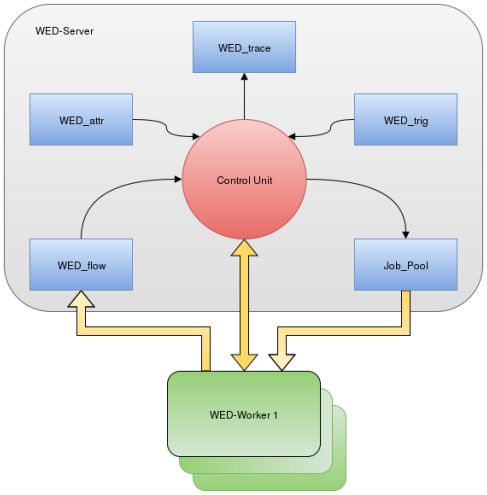

Figura 4. Flow-chart

do WED-server, ou quando o limite de tempo de espera por notificações termina (atributo wakeup_interval) e, nesse caso, é preciso consultar a tabela job_pool em busca de WED-transitions pendentes. Independentemente do modo de atuação e antes de inicializar a transação, cada WED-worker precisa solicitar ao WED-server uma trava (advisory_lock) na WED-transition e em qual instância do WED-flow será executada a transação. Essa trava é necessária tanto para avisar a outros WED-workers que estejam trabalhando na mesma WED-transition que não a executem para a mesma instância, quanto para que o WED-server possa controlar o tempo limite da execução de cada WED-transition. Não é possivel executar uma WED-transition sem que o respectivo WED-worker consiga obter essa trava previamente.

De posse da referida trava, um WED-worker terá que finalizar a transação dentro do tempo limite de execução da WED-transition. Essa transação é finalizada atualizando-se o WED-state atual da instância em que está sendo executada a WED-transition por meio de um UPDATE na tabela wed_flow.

Quando uma instância é atualizada, o WED-server novamente irá verificar se esse novo WED-state dispara novas WED-transitions e, em caso afirmativo, registrá-las na tabela job_pool, além que registrar os novos eventos em wed_trace. Caso esse novo WED-state satifaça a condição final e não haja nenhuma WED-transition pendente, essa instancia é marcada como finalizada e não poderá ser modificada, a menos que sejam inseridos novos WED-attributes ou que sejam modificadas ou adicionadas novas WED-triggers. Em caso de não houver WED-transitions pendentes e esse WED-state não for final, a instância será marcada como excessão e uma entrada especial será adicionada a tabela job_pool.

V. Detalhes de funcionamento

VI. Trabalhos Futuros

VII. Conclusão

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc,

Algoritmo 1 WED-server: disparo de WED-transitions

Input: Instância i de um WED-flow
Output: **true** se a transação foi bem sucedida,
false em caso contrário

1: $L \leftarrow$ lista vazia de WED-transitions
2: $s \leftarrow$ WED-state atual de i
3: **for** tg in WED-triggers **do**
4: $c, tr \leftarrow tg$(WED-condition),tg(WED-transition)
5: **if** s satisfaz c **then**
6: **if** c é a condição final **then**
7: insira $\_FINAL$ em L
8: **else**
9: insira tr em L
10: **end if**
11: **end if**
12: **end for**
13: **if** L está vazia **then**
14: **if** i é uma nova instância **then**
15: Rejeite i e aborte a transação
16: **return false**
17: **else if** i **NÃO** possui WED-transitions pendentes **then**
18: Maque i como WED-state de exceção
19: Insira _EXCPT em Job_Pool
20: **end if**
21: Atualize i em WED_trace
22: **return true**
23: **else if** L contém $\_FINAL$ **then**
24: **if** i possui WED-transitions pendentes **then**
25: Rejeite i e aborte a transação
26: **return false**
27: **else**
28: Marque i como WED-state final
29: Atualize i em WED_trace
30: **return true**
31: **end if**
32: **else**
33: **for** tr em L **do**
34: Insira tr na tabela Job_Pool
35: Envie notificação para o WED-worker que executa tr
36: **return true**
37: **end for**
38: **end if**

Algoritmo 2 WED-worker

Input: Nome da WED-transition: *trname*
Output: **true** se a transação foi bem sucedida,
false em caso contrário

1: **loop**
2: $n \leftarrow$ Valor nulo
3: **while** n é nula **do**
4: $n \leftarrow espera\_notificacao(trname, wkup)$
5: **if** n é nula **then** {nenhuma notificação recebida após esperar $wkup$ segundos}
6: $n \leftarrow$ busque por WED-transitions $trname$ pendentes no WED-server
7: **end if**
8: **end while**
9: Inicia a transação
10: **if** obter_trava(n) **then**
11: Execute a WED-transition $trname$ para uma determinada instância i do WED-flow
12: Finalize a transação
13: **return true**
14: **else**
15: Aborte a transação
16: **return false**
17: **end if**
18: **end loop**

molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

APÊNDICE A
APD

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ACKNOWLEDGMENT

The authors would like to thank...

REFERÊNCIAS

[1] João E. Ferreira, Osvaldo K. Takai, Simon Malkowski e Calton Pu. *Reducing exception handling complexity in business process modeling and implementation: the WED-flow approach.*, Em Proceedings of the 2010 international conference on On the move to meaningful internet systems - Volume Part I, OTM'10, páginas 150–167. Springer-Verlag, 2010.

[2] João Eduardo Ferreira, Kelly Rosa Braghetto, Osvaldo Kotaro Takai e Calton Pu. *Transactional recovery support for robust exception handling in business process services.* Em Proceedings of the 19th International Conference on Web Services (ICWS), páginas 303–310, 2012