eXPLoit

Um Sistema de Processamento em Pipeline de XML

Nuno Teixeira[1] e José Paulo Leal[2]

[1] FCUP, Universidade do Porto
[2] DCC-FC & LIACC, Universidade do Porto

Resumo Um *pipeline* de processamento *XML* é uma sequência de transformações de documentos em que a saída de uma transformação alimenta a entrada da transformação seguinte. A vantagem da utilização de *pipelines* é a flexibilização do uso das transformações de *XML*, permitindo a sua reutilização em diferentes *pipelines*.
As linguagens de *pipeline* de processamento *XML* permitem definir e controlar o encadeamento das transformações. Têm sido propostas várias linguagens deste tipo, entre as quais o *XPL - XML Pipeline Language* - submetido ao W3C para discussão pela *Orbeon Inc.*.
Este artigo descreve o *eXPLoit*, um sistema de processamento em *pipeline* de dados *XML*, que inclui um interpretador da linguagem *XPL* e um conjunto de adaptadores que facilitam a sua ligação a diferentes contextos, nomeadamente: a linha de comandos, aplicações *web* e *webservices*. Os adaptadores existentes no *eXPLoit* convertem os dados relativos a cada ambiente em fluxos de *XML* que são processados pelo interpretador. Exemplos desses dados são a informação contida num pedido *HTTP* para uma aplicação *web* ou a mensagem *SOAP* enviada para um *Web Service*. O interpretador do *eXPLoit* avalia um documento *XPL* que controla a invocação de um conjunto de processadores *XML*. O *eXPLoit* inclui um biblioteca básica de processadores, tais como transformações *XSL*, conversões de *CSV* em *XML*, consultas a *webservices*, entre outros. É também possível a adição de processadores ao *eXPLoit*. O interpretador possibilita o uso das estratégias de avaliação *lazy* e *greedy*.

1 Introdução

Com a crescente preponderância do *XML* na transferência de dados entre sistemas heterogéneos, são cada vez mais frequentes os componentes que usam este formalismo. De facto, o *XML* já extravasou há muito a formatação de páginas *web* para o qual foi desenhado. É a peça fundamental na implementação de *web services*, quer na formatação das mensagens, quer no protocolo que as envolve, quer na especificação dos próprios serviços. O *XML* é também cada vez mais usado na formatação de documentos de aplicações, garantindo um formato aberto, independente de plataformas e dos sistemas que os produziram.

O conceito de *pipeline XML* foi proposto para gerir o crescente número de componentes desenvolvidos para processar dados formatados em *XML*. Um *pipeline XML* liga um conjunto de componentes que processam *XML*, conectando

os fluxos de saída dum processador aos fluxos de entrada do seguinte. Esta abordagem permite a reutilização de processadores de *XML* pela sua participação em diferentes *pipelines*. Permite também uma nova forma de programação, baseada na articulação de um conjunto de componentes genéricos que processam *XML*.

A gestão de *pipelines XML* é feita por motores de *pipeline*, a partir de definições em linguagens *XML* especializadas. A generalidade dos motores de *pipelines* actualmente existentes estão associados a sistemas específicos, dos quais se destacam os gestores de conteúdos, como o *Cocoon*[1] ou o *Orbeon Forms* (anteriormente chamado *Orbeon Presentation Server*)[2]. No entanto, um motor de *pipeline* genérico, utilizável em múltiplos contextos, poderá aumentar a capacidade de reutilização dos processadores de *XML*.

Um motor de pipeline que possa ser usado indistintamente a partir de um contentor de *servlets*, de um motor *SOAP*, ou mesmo da linha de comandos permitirá a reutilzação dos mesmos *pipelines* como parte duma aplicação *web*, como operações de um *web service*, ou como comandos *shell*. O objectivo do trabalho apresentado neste artigo é, portanto, desenvolver um motor de *pipeline XML* com possibilidade de utilização em múltiplos contextos, e explorar as possibilidades deste tipo de abordagem.

O motor de *pipeline* desenvolvido -o *eXPLoit* - deve o seu nome à linguagem de *pipeline* em que se baseia, o *XPL*. O *eXPLoit* é constituído por um interpretador desta linguagem, com uma *API* neutral que permite a sua utilização em múltiplos contextos. O *eXPLoit* contém ainda um conjunto de adaptadores que facilitam a utilização do interpretador em alguns contextos, como sejam os pedidos e resposta *HTTP*, as mensagens *SOAP*, o sistema de ficheiros e o ambiente de execução.

Neste artigo começaremos por apresentar as linguagens de *pipeline* em geral e o *XPL* em particular, bem como alguns dos sistemas que integram motores de *pipeline*. Seguidamente descreveremos o sistema *eXPLoit* e terminaremos com um exemplo da sua utilização em diferentes contextos.

2 Estado de Arte

Nesta secção são apresentados alguns conceitos sobre o processamento *pipeline*, começando por definir o próprio conceito. São também apresentadas as principais linguagens de definição de *pipeline*. Dos vários motores de *pipeline* presentemente em utilização, como a plataforma *Orbeon Forms*, destacamos o gestor de conteúdos *Cocoon*, cuja linguagem de definição de *pipeline* tem um papel central na configuração da *framework*. Nesta secção é também dada uma ideia geral da linguagem XProc.

2.1 Definição de *XML Pipeline*

Um pipeline de processamento *XML* corresponde à interligação de um conjunto de processamentos *XML*, normalmente chamadas transformações *XML*.[3]

Dadas duas transformações T_1 e T_2, estas podem ser ligadas de forma a que um documento *XML* transformado por T_1 seja posteriormente usado por T_2. Estes *pipelines* são chamados lineares, pois um documento dado como *input* percorre uma série de transformações pré-fixadas para produzir um documento como *output*. *Pipelines* não lineares podem incluir:

- Múltiplos *inputs* e *outputs* – possibilidade de receber *input* de várias fontes e a sua aplicação resultar em diversos *output*s;
- Testes – onde uma *pipeline* é executada se se verificar uma dada condição;
- Iteração – a transformação é aplicada a um conjunto de fragmentos *XML* seleccionados de um documento;
- Agregações – múltiplos documentos (ou fragmentos *XML*) são agregados num único.

2.2 Linguagens *XML Pipeline*

As linguagens *XML Pipeline* são usadas para definir *pipelines*. Um programa escrito numa linguagem deste tipo é interpretado por um componente denominado *motor de XML pipeline*. Este é responsável por criar processamentos, ligá-los entre si e, por fim, processá-los. Existem várias linguagens deste tipo, tais como:

- *W3C XML Pipeline Language (XPL) Version 1.0 (Draft)*[4] especificada numa *W3C Submission*. A *Orbeon Inc.* efectuou uma implementação de uma versão anterior a esta linguagem. Esta é a linguagem que é interpretada pelo sistema que este artigo pretende ilustrar;
- *smallx XML Pipelines*[5] é usada pelo projecto *smallx*;
- *ServingXML*[6] define um vocabulário para exprimir transformações $flat \rightarrow XML$, $XML \rightarrow flat$, $flat \rightarrow flat$, e $XML \rightarrow XML$ em *pipelines*;
- *XProc: An XML Pipeline Language*[7] é a linguagem mais recente cuja especificação começou a ser formulada em finais de Dezembro 2005 e continua muito activa. Esta linguagem surge como uma tentativa de se criar uma linguagem *standard* para processamento *XML*.

XProc
Esta linguagem aborda um *pipeline* como sendo um conjunto de componentes interligados. Quer os componentes quer o *pipeline* aceitam como *input* um conjunto (vazio ou não) de documentos *XML* e produzem, ou não, um conjunto de documentos *XML* como *output*. O *input* de um componente pode ser um documento *web*, ser o *input* do *pipeline* ou o *output* de outro componente. A *XProc* classifica os componentes em dois tipos: *steps* e *constructs*. Um *step* consiste num conjunto de operações, nomeadamente processamentos *XML*. Por seu lado, um *construct* engloba um conjunto de componentes. Um exemplo de um *construct* é o próprio pipeline. A linguagem *XProc* encaixa-se no conjunto de linguagens de *pipelines* não lineares, pois suporta múltiplos inputs e outputs, testes, iterações.

Além disso, inclui um conjunto de funcionalidades tais como: execução de transformações usando o sistema *try/catch*, cópias de documentos *XML* em que determinadas áreas desse documento são transformadas, importação e execução de *sub-pipelines*.

2.3 Motores de linguagens *XML Pipeline*

Neste secção são enunciados alguns dos motores de linguagens *XML* Pipeline existentes no momento. É indicada a forma como estes motores interagem com os programas que estão escritos na linguagem *XML* que é usada na sua especificação.

Orbeon Forms
A *Orbeon* desenvolveu uma plataforma *open source* que permite criar aplicações *web* tendo como base o uso de *XForms* e documentos *XML*. Esta plataforma é composta pelos seguintes componentes: *XForms* para a criação de formulários *web* complexos; *PFC (Page Flow Controller)* é o centro de qualquer aplicação nesta plataforma, pois define as páginas da aplicação e como é feita a navegação entre elas; *XPL* é a linguagem usada para definir os *pipelines* de processamento *XML* que podem ser usados na aplicação.

O motor que manipula os formulários é baseado em *Ajax*, usando a biblioteca fornecida pela *Yahoo!*: *Yahoo! User Interface library*. O *PFC* é o componente responsável para controlar o fluxo de execução de uma aplicação implementada na plataforma. Este componente direcciona os pedidos que vêm por parte do cliente para as páginas que fazem parte da aplicação. Estas páginas devem obedecer a uma arquitectura *MVC* (*Model-view-controller*). A analogia é a seguinte:

- *Model* - a sua função é obter informação para ser mostrada pelo *View*. Normalmente o *Model* está associado ao um pipeline *XPL*;
- *View* - produz documentos *XHTML* e *XForms*, mas pode produzir outros formatos tais como: *XSL-FO* ou *RSS*. Um exemplo de um *View* é documento no formato *XSL* que efectua *XSLT* sobre o resultado fornecido pelo pipeline;
- *Controller* - é precisamente esta a função ligada ao *PFC*. Envia o pedido que chega de um cliente, como um *browser*, aos respectivos *Model* e *View*. Por isso indica a ligação entre o *Model* e o *View*.

Esta plataforma possui um conjunto de processadores, entre os quais se destacam: processadores que efectuam transformações *XSL*, acesso a base de dados *SQL*, serialização de documentos *XML* para o disco, comunicação com *webservices*. É possível uma integração com a linha de comandos e com webservices.

Cocoon
O *Cocoon* é uma plataforma de publicação *XML* na web. Permite definir documentos *XML* e aplicar transformações para outros formatos, como o *HTML*, *PDF*, ou texto. O *Cocoon* permite controlo das transformações, possibilitando, desta forma, *pipelines* não lineares.

Os processadores de *XML* do *Cocoon*, controlados pela sua linguagem de definição de *pipelines* são os seguintes: *Matchers*, *Generators*, *Transformers*, *Serializer*, *Selectors*, *Views*, *Readers* e *Actions*.

Uma *pipeline* no *Cocoon* deve ser formada por uma sequência dos seguintes componentes: um *generator* que fornece os dados a serem trabalhados (como por exemplo, o *FileGenerator* que lê um ficheiro e o converte em eventos *SAX*); um *transformer* que recebe eventos *SAX* e gera outro *XML* como eventos *SAX* (como por exemplo o *XalanTransformer* que aplica *XSL*); e finalmente um *serializer* que recebe eventos *SAX* e converte-os para outro formato (como o *XMLSerializer* - transforma eventos *SAX* num ficheiro *XML*).

3 XML Pipeline Language (XPL)

O *XPL* é linguagem de definição de *pipelines* proposta à W3C pela *Orbeon Inc.*. Esta linguagem é utilizada no *Orbeon Forms*, uma plataforma desenvolvida pela *Orbeon*, e é também implementada pela *Oracle* no *Oracle XML Developer's Kit*[8]. Esta linguagem de *pipeline* foi a escolhida para ser incluída porque quando se iniciou o estudo para a implementação do *eXPLoit* era a que possuia uma definição mais estável. Chegou-se a colocar a linguagem *XProc* como hipótese mas a sua definição encontrava-se numa fase embrionária. Contudo, é encarado como uma tarefa a ser executada a inclusão de um interpretador de *XProc* no *eXPLoit*.

3.1 Definição do XPL

Um ***Infoset*** é um fragmento de *XML*. Cada um desses fragmentos tem uma identificação a qual permite que seja possível o acesso ao seu valor. Exemplos de *Infoset* são os parâmetros de *input* e *output*.

Um **programa *XPL*** é um documento *XML* que indica uma sequência de operações efectuadas sobre um conjunto de dados: *Infosets*. Cada operação é definida como um comando da linguagem. As operações sobre um *Infoset* incluem a produção, o consumo ou a transformação de *Infosets*. Um exemplo de transformações é o *XSL* [9]. O *XPL* inclui ainda comandos de controlo de fluxo de execução, como testes e iterações e, controlo de I/O, à semelhança das linguagens de programação. Os programas *XPL* são validados pelo interpretador antes de serem processados. O interpretador assegura a validade sintáctica dos programas consultando um ficheiro *XSD* (*XML Schema Definition*), incluído no *eXPLoit*.

A cada **comando** está associado um elemento específico no espaço de nomes *http://www.orbeon.com/oxf/xpl*. A especificação do XPL indica a criação de um prefixo *p* associado a este espaço de nomes. Um programa *XPL* suporta os seguintes elementos:

- `p:input, p:output`
  identifica os parâmetros de *input* e *output*, respectivamente. Estes elementos poderão existir dentro dos elementos seguintes, dependendo do seu tipo;

– `p:pipeline`
  engloba todo o programa *XPL*. Este comando tem como argumentos *p:input* e *p:output* que indicam os parâmetros de entrada e saída do programa. Estes argumentos têm uma identificação associada para permitir o seu uso dentro do programa;
– `p:choose`
  teste de condições para controle de fluxos de execução. Este comando contém um conjunto de ramos, cada um deles com um teste. É a verificação destes testes que indica se o ramo deverá ser processado pelo interpretador;
– `p:for-each`
  itera sobre um *Infoset*, executando os comandos embebidos no elemento, por cada iteração. O conjunto de fragmentos *XML* a iterar é dado por uma expressão na linguagem *XPath*[10] aplicado a um *Infoset*. Existe uma referência que é associada a cada fragmento que é iterado. Esta referência permite aceder ao valor do fragmento quando os comandos embebidos no elemento são executados;
– `p:procesor`
  este elemento define um processamento sobre fragmentos *XML*. Os processadores são o ponto de extensão do *motor pipeline*, podendo ser adicionados processamentos para novas funcionalidades. O funcionamento do comando p:processor é análogo a uma chamada a uma função numa linguagem declarativa, onde são passados os parâmetros de *input* e têm como retorno determinados *output*s.

O acesso a um *Infoset* faz-se recorrendo a **referências**. As referências são indicadas em atributos de um comando. Estas podem:

– ser referências a documentos externos: ficheiros alojados no sistema ou páginas *web*;
– identificar dados existentes na tabela de variáveis no decorrer da execução do interpretador. Como por exemplo, *#data*, que corresponde à variável denominada *data*;
– agregar documentos usando a função *aggregate()*;
– seleccionar parte de fragmentos *XML*, usando expressões *XPointer*[11];
– corresponder a um fragmento *XML* que é iterado num *p:for-each*.

3.2 Hello World

O seguinte exemplo de um programa em *XPL* ilustra a estrutura base da linguagem assim como o uso do processador *xpl:hello*, um dos processadores da biblioteca de processadores do *eXPLoit*:

```
<p:pipeline version="1.0"
          xmlns:p="http://www.orbeon.com/oxf/xpl"
          xmlns:xpl="http://www.orbeon.com/oxf/xpl/standard">
    <p:input name="data" infoset="http://www.helloworld.com" />
    <p:output name="data" infosetref="#data" />
```

```
    <p:processor name="xpl:hello" />
</p:pipeline>
```

Este programa vai receber a página alojada em *http://www.helloworld.com* e essa página será o seu *output*. O *p:input* e o *p:output* estão interligados, pois usam o mesmo identificador (*data* e *#data*) como consta nos atributos destas *tags*. O processador *xpl:hello* escreve no *stdout* do sistema o texto "Hello world"quando é executado.

3.3 Processadores da biblioteca básica de processadores

A especificação do *XPL* apresenta um conjunto de processadores que considera serem *standard*. Seguem-se esses processadores e uma breve descrição da funcionalidade de cada um:

- **identity** Usando este processador torna-se possível atribuir a um dado identificador um determinado *Infoset*. Este processador é muito útil para definir *Infoset* associado a um fragmento *XML* no contentor do processador ou até num documento externo. Esta utilização do *identity* pode ser equiparada às variáveis de uma linguagem de programação.
- **pipeline** Com este processador é possível executar sub-programas em *XPL*. Como argumentos, são indicados um conjunto de *input*s e *output*s. Como *input* indicam-se novos *Infoset*s ou *Infoset*s que existam no ambiente de variáveis do programa que usa este processador. Estes *Infoset*s serão guardados no ambiente de variáveis do sub-programa. Como um sub-programa pode gerar vários *output*s, deve-se indicar nos argumentos p:output quais os *output*s em que se está interessado. Esses são passados para o ambiente do programa principal. Este processador permite modular a programação por permitir que se possa chamar o mesmo sub-programa em programas distintos.

4 eXPLoit

O *eXPLoit* é composto por vários componentes conforme esquematizado na figura 1. O interpretador é responsável pela interpretação dos programas *XPL* e os restantes acedem a este interpretador e denominam-se adaptadores. Os adaptadores comunicam com o interpretador usando a *API* que este disponibiliza, e estão associados a diferentes ambientes onde a execução de *XPL* ocorre: numa *shell*, na *web* ou num *webservice*.

4.1 Estrutura base

Cada adaptador é responsável por fornecer ao interpretador um *Infoset* que possua dados relacionados com o ambiente que o adaptador suporta. O *Infoset* tem o identificador *#request*. Segue uma breve descrição dos adaptadores que são incluídos no *eXPLoit* e qual o valor de *#request*:

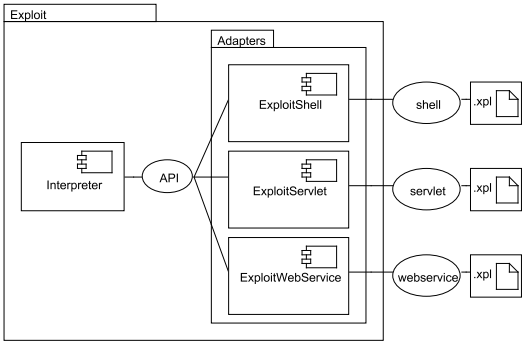

Figura 1. Estrutura base do eXPLoit

- *ExploitWebService* - implementa um *webservice* que comunica com o interpretador. É responsável por enviar os dados que servirão de *input* ao programa *XPL* recebidos no pedido *SOAP* e produz uma mensagem de resposta *SOAP* com o *output* do mesmo. Este módulo usa o *AXIS* para implementar o *webservice*. O *#request* é a mensagem *SOAP* que é enviada para o *webservice* convertida para um fragmento *XML*;
- *ExploitServlet* - uma classe que faz a ligação entre um pedido *HTTP* (*POST* ou *GET*) e a interpretação de um programa *XPL*. Esta classe poderá ser usada com qualquer contentor de *Servlets*, como o *Tomcat*, que recorra a *Servlets* para disponibilizar serviços. O fragmento XML que é identificado por #request contém informações presentes no *HTTP Request*. São indicados valores que existem no *header* como: o *browser* usado, o tipo de conteúdo e o *charset encoding* usados pelo *browser*, entre outros. São também indicados dados referentes ao pedido *HTTP* efectuado, ou seja, o nome das variáveis e respectivos valores;
- *ExploitShell* - é o que indica ao interpretador quais os argumentos colocados na linha de comandos quando a sua execução é feita a partir de uma *shell*. Além dos argumentos também é fornecido ao interpretador o conjunto das variáveis ambiente do sistema operativo e o que é enviado para o *stdin*. Esta informação é construída num fragmento *XML* e associada à variável *#request*.

4.2 Estratégias de avaliação

O interpretador de *XPL* do *eXPLoit* suporta duas estratégias de avaliação - *lazy* e *greedy* - sendo esta última usada por omissão. No caso de uma avaliação *greedy*, o interpretador percorre todos os comandos do programa e executa-os na ordem em que estão definidos. Na avaliação *lazy*, são executados os comandos necessários para o *output* do programa *XPL*. A avaliação *lazy* permite que sejam avaliados apenas os comandos necessários para obter um determinado identificador. Desta

forma, comandos que demorem imenso tempo a serem processados, só o são processados se for necessário. Outra vantagem é a possibilidade de se colocar num programa *XPL* vários blocos de comandos, cada um deles responsável por um determinado *output*. Usando a avaliação *lazy*, a execução desses blocos seria controlado pela indicação do *output* desejado desse programa.

4.3 API

A classe *Interpreter*, responsável pela interpretação de um programa *XPL* comunica pela *API* por forma a inicializar e controlar o processo da interpretação. Os seguintes métodos são usados pelos adaptadores:

- *void setRequest(Document requestDoc)* - inicializa a variável *#request* no interpretador, sendo necessário fornecer um objecto da classe *Document*. Esta variável pode ser usada no resto do programa podendo aceder, por isso, a toda a informação proveniente do contexto de execução;
- *void setLazyEvaluation(boolean lazyEvaluation)* - define qual o método de avaliação que é usado na interpretação. Por defeito, é usada a avaliação *greedy*;
- *void eval(File xplProgramFile)* - indica ao interpretador um apontador para o ficheiro que contém o programa *XPL* a ser interpretado;
- *void eval(Document xplDocument)* - passa ao interpretador um documento cujo conteúdo é um programa *XPL*;
- *String getOutputAsString(String outputId)* - solicita ao interpretador o resultado final de um determinado output nomeado *outputId*. Este resultado é serializado pelo interpretador e retornado para o adaptador;
- *Document getOutputAsDocument(String outputId)* - efectua o mesmo pedido que o método anterior só que o resultado retornado será da forma de objecto *Document*;
- *Source getOutputAsSource(String outputId)* - retorna o mesmo que os métodos anteriores só que o resultado retornado será um objecto *Source*.

4.4 Processadores adicionais

Foram acrescentados à biblioteca básica do *XPL* novos processadores pelo *eXPLoit*. Outros podiam ser criados como, por exemplo, um processador que envie e receba *emails*, um processador que aceda a *RDBMS* ou um processador que guarde resultados de um programa num ficheiro.

A extensão de novos processadores ao *eXPLoit* efectua-se configurando o ficheiro *exploit.xml*. Este ficheiro contém a relação entre o nome do processador (que é indicado no elemento *p:processor*) e o caminho completo para a classe que o implementa. De seguida, são dados os processadores incluídos no *eXPLoit* para linguagem *XPL*:

- **xslt** O processador *xslt* necessita como *input* os atributos *config* e *data*. O *config* refere-se a um ficheiro *.xsl*. O atributo *data* corresponde aos dados

que se pretendem a que seja aplicada a transformação. O resultado será um *Infoset* que é obtido após o uso da transformação indicada em *config* sobre os dados *data*.

- **csv2xml** Este processador, mediante um conjunto de dados em formato *CSV*, cria um documento *XML* que os representa. Esses dados deverão ter como primeiro registo o nome dos campos dos restantes registos. Como parâmetros poderão ser passados separadores de registos e de campos. Por omissão, é usado '$\backslash n$' (*newline*) e '*;*'. O resultado final é um documento que, além dos registos, contém a indicação do nome dos campos que cada registo possui.
- **webserviceRPC** Com este processador acede-se *webservices* usando *RPC*. Para se usar este processador, é necessário conhecer-se o ficheiro *WSDL* associado ao serviço que se pretende aceder.
- **webserviceMessage** A funcionalidade deste processador é idêntico ao anterior, só que ao invés de usar *RPC*, usa um outro método que consiste na troca de *Messages*.
- **googlewebservice** Este processador usa a *API*[12] da *Google* que permite aceder a um conjunto de serviços disponibilizados pela Google. São eles: a pesquisa de páginas, o acesso a páginas em cache e o uso de um *spell checker*. O retorno é dependente do tipo de serviço que for pedido.
- **xsdvalidator** Possibilita a validação de documentos *XML* mediante o uso de *XSD* (*XML Schema Definition*). É necessário indicar como input dois parâmetros: *config* e *data*. O primeiro é um ficheiro *.xsd* e o último os dados a serem validados. O retorno será *<validator result="valid"/>* se a validação suceder. Caso contrário o retorno é *<validator result="invalid"/>*.

5 Avaliação

Para ilustrar a flexibilidade do *eXPLoit* e o modo como se obtém dados de várias fontes e de vários ambientes, é dado como exemplo um programa que transforma dados do ficheiro */etc/passwd* existente no sistema *UNIX* num documento *HTML*.

O ficheiro */etc/passwd* é um ficheiro no formato *CSV* cujo delimitador de registos é uma *newline* e o separador de campos é o caracter **':'**. Cada registo deve conter os campos: nome do utilizador, palavra passe, *UID*, *GID*, nome completo do utilizador, directório da área pessoal e a shell a ser usada. De notar que o primeiro registo do ficheiro deve conter o nome dos campos dos registos seguintes. Estes campos são os elementos que figuram na conversão deste ficheiro para *XML*. Este ficheiro *XML* passa por um processo de validação. Para ser considerado válido deve possuir no mínimo três campos não vazios: nome de utilizador, *UID* e *GID*. Caso a validação suceda este é transformado num documento HTML.

A figura 2 mostra um diagrama do modo de funcionamento dos programas. Esta figura ilustra a forma como se pode reutilizar o mesmo programa (*passwd2html.xpl*) através do uso de *sub-pipelines*. De notar que o *sub-pipeline* é executado em dois ambientes de execução distintos: *shell* e *web*. Como vimos neste

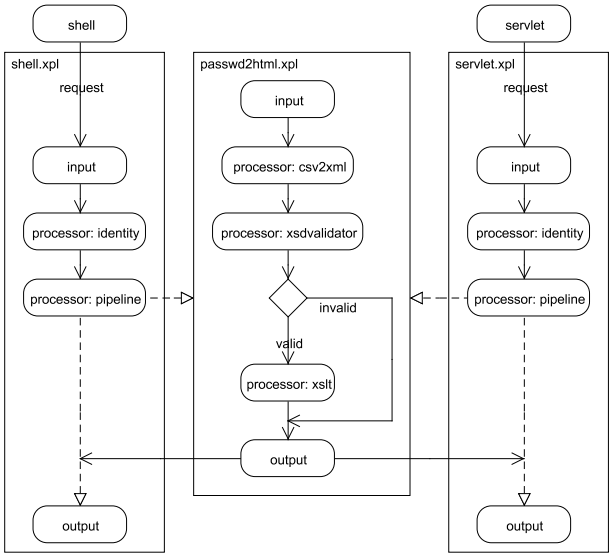

Figura 2. Execução do passwd2html em vários ambientes

artigo, os adaptadores do *eXPLoit* são responsáveis por enviar para o interpretador um fragmento *XML* que possua informação relativa ao pedido (*request*). No caso da *shell* o ficheiro é passado pelo *stdin* enquanto que no ambiente *web* este é enviado por *upload* através de um pedido *POST*. Cada um dos programas (*shell.xpl* e *servlet.xpl*) faz a ponte para a execução do sub-pipeline que efectua a transformação desejada. Esta ponte verifica-se inicializado variáveis através do processador *identity* e com expressões *XPointer*. Estas variáveis têm como conteúdo fragmentos *XML* que são passados como *input* para o sub-pipeline.

O sub-pipeline define uma sucessão de transformações. Em primeiro lugar, é executado o processador *csv2xml* que cria um fragmento *XML* com o conteúdo do ficheiro que faz parte do *request* e com a indicação dos delimitadores de registo e de campo. De seguida, este fragmento é validado conforme a as regras definidas no *XSD*. O *XSD* indica a estrutura que o fragmento *XML* deve possuir assim como a obrigatoriedade dos três campos falados anteriormente. Caso seja um fragmento válido este passa por uma transformação *XSL* que origina um fragmento *HTML* e que é o retorno final. Caso seja um *XML* inválido é indicado como retorno um documento *HTML* com uma mensagem de erro. Neste exemplo, o *output* do *sub-pipeline* é o resultado final.

O programa executado na *shell*, por omissão, imprime o conteúdo de todas as variáveis definidas como output do pipeline. A *servlet* retorna um resultado idêntico assim como o *header HTTP* respectivo. Quer o *header*, quer a variável de *output* desejada são configuráveis nos ficheiros de configuração do contentor de *Servlets*, como por exemplo, o *Tomcat*.

6 Conclusão

O exemplo da secção anterior mostra como o *eXPLoit* é modular na forma como permite interligar vários ambientes que executem o mesmo programa *XPL*. A acrescentar a isso junta-se a capacidade de se acrescentar, de uma forma simples, novos processadores para serem usados nos programas *XPL*. O sistema pode facilmente ser extendido com novos adaptadores para permitir a sua utilização noutros contextos. Um exemplo de um adaptador seria o uso do *eXPLoit* num sistema *GUI*.

O *eXPLoit* é uma aplicação que mostra as vantagens do uso de *pipelines* de *XML* em diferentes contextos. O facto de os fluxos de transformação serem baseados num conjunto de componentes, simplifica a adaptação destes componentes com o programa que se pretende obter. O uso desta abordagem permite que haja menos erros de programação por estes componentes funcionarem como funções pré-definidas de uma linguagem de alto nível.

Utilizando o *eXPLoit*, pretende-se que haja ganhos na implementação de aplicações *XML* pois a ocorrência de erros será menor e o tempo de implementação mais rápido. Dado que o uso de *pipelines* indicia uma abordagem usando-se componentes, o código torna-se modular e, por isso, reutilizável.

Como nota final e como projecto alternativo, sugere-se a criação de uma aplicação *GUI* para criar código de programas em *XPL*, usando-se gráficos que representem os vários componentes: parâmetros *input*/*output*, os comandos *standard* e chamadas a processadores. Traria vantagens porque em aplicações mais complexas, usar um editor *XML* torna-se impraticável.

Referências

1. Apache http://cocoon.apache.org/: (The Apache Cocoon Project)
2. Orbeon, Inc. http://www.orbeon.com/ops/: (Orbeon Forms)
3. World Wide Web http://en.wikipedia.org/wiki/XML_pipeline: (XML pipeline)
4. Bruchez, E., Vernet, A.: XML Pipeline Language (XPL) Version 1.0 (Draft). World Wide Web, http://www.w3.org/Submission/xpl/. (2005)
5. World Wide Web https://smallx.dev.java.net/: (smallx: Project Home Page)
6. World Wide Web http://servingxml.sourceforge.net/: (ServingXML)
7. Walsh, N., Milowski, A.: XProc: An XML Pipeline Language. World Wide Web, http://www.w3.org/TR/xproc/. (W3C Working Draft 17 November 2006)
8. Oracle http://www.oracle.com/technology/tech/xml/xdkhome.html: (Oracle XML Developer's Kit 10g)
9. Clark, J.: (XSL Transformations (XSLT) Version 1.0)
10. Clark, J., DeRose, S.: (XML Path Language (XPath) Version 1.0)
11. Steve DeRose, Ron Daniel Jr., P.G.E.M.J.M.N.W.: (XML Pointer Language (XPointer) Version 1.0)
12. Google http://code.google.com/apis/soapsearch/: (Google SOAP Search API)