# Implementação de um modelo baseado em XML para suporte da dinâmica processual de negócio

Gilberto Rocha[1], Isidro Vila Verde[1], Rui Humberto Pereira[2]

[1] Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-465 Porto Portugal.
{mrs03015,jvv}@fe.up.pt
[2] Instituto Politécnico do Porto
Rua Dr. Roberto Frias, 712 4200-465 Porto Portugal.
rhp@iscap.ipp.pt

**Resumo.** Este artigo descreve um modelo para implementação de plataformas dinâmicas de negócio, exclusivamente baseadas em tecnologias XML. Apresenta um caso real que implementa as suas diversas camadas, apresentação, lógica e dados, em XML. A comunicação entre camadas é assegurada por serviços Web (WS), tornando esta arquitectura orientada aos serviços (SOA). O modelo proposto sustenta toda a programação do processo de negócio numa linguagem de alto nível, o WS-BPEL, proporcionando, desse modo, condições de adaptação ao dinamismo exigido pelo negócio da organização e à heterogeneidade dos sistemas. O caso real é de uma secretaria electrónica que surge no contexto dos portais Web universitários. O sistema desenvolvido visa oferecer um conjunto de serviços para acesso a informação e para o despoletar de acções computacionais e/ou humanas.

## 1 Introdução

Nestes últimos anos, a proliferação da Internet, quer a nível tecnológico, quer a nível social, permitiu democratizar o acesso à informação e melhorar a interacção com o mundo da informação.

Associado a este fenómeno, foram aparecendo inúmeros serviços on-line, que evoluem de acordo com os requisitos que lhe vão sendo impostos em função dos objectivos. A continuação desta evolução, está dependente das tecnologias que a suportam, as quais devem permitir uma grande flexibilidade para que seja possível satisfazer e adaptar-se aos requisitos que vão surgindo.

Neste contexto surgiram os portais Web universitários e, na sequência destes, as secretarias electrónicas. A aproximação clássica aos serviços do tipo secretaria electrónica, consiste em fazer um levantamento complexo à lógica do negócio e repercuti-la em código aplicacional. Este modelo de desenvolvimento de código para implementação de lógicas de negócio, tem obviamente desvantagens, a primeira das quais é a adaptabilidade às alterações processuais, isto é, alteração à lógica de negócio.

Por outro lado, hoje em dia as organizações têm uma herança no que respeita aos sistemas de informação já existentes (*legacy*), o que torna complicado desenvolver e acrescentar novas funcionalidades. Se disponibilizarmos estes sistemas como serviços e usarmos o modelo SOA [1], [2], [3] como elemento agregador, conseguimos construir facilmente novas aplicações, e/ou reestruturar aplicações já existentes.

Um serviço on-line, pode ter que interagir com várias aplicações e algumas delas utilizam tecnologias de certa forma obsoletas. Podemos envolver essas aplicações com uma camada aplicacional que interage com a API da aplicação e disponibiliza um serviço Web para atender pedidos de outras aplicações ou serviços Web.

Com base nisto, foi desenvolvido no âmbito da tese de mestrado um Modelo totalmente baseado em XML [4], que procura satisfazer a dinâmica dos requisitos inerentes à evolução do modelo de negócio. O Modelo desenvolvido é constituído por um conjunto de tecnologias de apresentação, lógica e armazenamento baseadas em XML. Pretendeu-se avaliar a exequibilidade destas tecnologias para a agilização dos serviços à mudança.

Este Modelo foi implementado numa secretaria electrónica, que pode conter vários processos, e esses processos podem ter que ser reconfigurados ou alterados ao longo do tempo. Nas aplicações Web actuais, isso implica alterar o código da aplicação. Neste artigo apresenta-se uma possível solução que envolve a utilização de um motor BPEL [5], onde é especificado o processo de negócio.

Na próxima secção, apresentamos a contextualização da necessidade deste modelo e fazemos uma breve descrição das arquitecturas e tecnologias existentes, que visam dar resposta a estes problemas. Na secção três, apresentamos e descrevemos um Modelo que assenta no uso estrito de tecnologias XML, mas permitindo a interface via serviços Web a aplicações convencionais. Na secção quatro, ilustramos a implementação de um módulo de uma secretaria electrónica de acordo com este modelo. Por fim apresentaremos as conclusões resultantes da implementação e das expectativas daí extraídas.

## 2 Estado de arte

O mundo dos negócios é um ambiente que está sempre em constante alteração. As empresas evoluem, querem expandir-se e melhorar a sua própria estrutura, quer por movimentos de aquisição e fusão, quer por adaptações ou reestruturações internas. Este dinamismo obriga os sistemas de informação a estarem preparados para rapidamente reflectirem as novas realidades e as prováveis alterações nas lógicas de negócio da empresa. Por outro lado, em qualquer sistema de informação, os requisitos são dinâmicos e constantemente se alteram.

Se a estrutura de suporte ao negócio da empresa for orientada ao serviço, torna muito mais fácil ultrapassar as exigências impostas pelas situações em cima explicadas e satisfazer rapidamente os novos objectivos das organizações.

Ao desenvolvermos uma aplicação para suporte de lógica de negócio, ou temos tecnologias que são capazes com eficiência e rapidez de adaptar-se aos processos, ou então estamos a construir uma aplicação que quando estiver totalmente pronta, estará já desactualizada.

As linguagens de alto nível de descrição de processos, permitem definir o núcleo do sistema de negócio, para que seja dinâmico e possibilite a convivência inter-aplicacional num ambiente complexo e heterogéneo. Em ambientes onde existe uma grande diversidade de aplicações, em diferentes tecnologias, a implementação de processos pode ser bastante simplificada com a utilização de linguagens de definição de processos de negócio.

O processo de negócio é uma sequência de interacções entre diversas entidades internas e/ou externas a uma organização, que tem como finalidade modelar o negócio da organização. Um bom processo de negócio deve maximizar e flexibilizar a reestruturação e/ou a integração de novas estruturas de negócio. Isto torna os processos complexos e dinâmicos, implicando constantes alterações com o objectivo da optimização. Mais uma vez, as linguagens de definição processual podem minimizar os inconvenientes do carácter dinâmico dos processos organizacionais.

Uma aplicação convencional, que implementa um processo de negócio, tem embebido no próprio código da aplicação a lógica de negócio. Isto implica que sempre que seja alterado o processo de negócio, é necessário alterar o código da aplicação, tornando complexa a gestão do processo de negócio.

## 2.1 SOA

A Service Oriented Architecture (SOA) tem como objectivo orientar as aplicações à disponibilização como serviços. Ao disponibilizarmos uma determinada aplicação como serviço, permitimos que essa aplicação consuma, troque e disponibilize informação. Estas aplicações já têm um determinado papel dentro de uma organização, permitindo a sua reutilização e minimizando o impacto da mudança.

A utilização de arquitecturas SOA permite uma maior facilidade de adaptação do sistema a novos requisitos e dá às organizações uma maior capacidade para poderem responder de uma forma rápida e eficiente às exigências do mercado.

As estratégias de desenvolvimento e implementação assentes em SOA's podem variar de acordo com o facto de se ter que alterar serviços existentes, desenvolver novos serviços ou integrar serviços que foram desenvolvidos fora da organização.

## 2.2 Linguagens de descrição de processos

Uma linguagem de descrição de processos suporta a lógica dos processos de negócio e deve permitir definir a interoperabilidade entre aplicações.

Um processo de negócio pode ser descrito como uma orquestração de interacções, onde existe um ponto central que vai ser encarregue de coordenar as aplicações, ou pode ser descrito como uma coreografia, onde a definição do processo de negócio vai estar distribuído entre as aplicações que estão envolvidas. Numa coreografia, as aplicações agem entre si de acordo com a definição do processo de negócio, mas a implementação é mais difícil, porque coloca problemas de sincronização.

### 2.3 Base de dados XML

Uma base de dados XML, tem como unidade fundamental de armazenamento, um documento XML, análogo às bases de dados relacionais que utilizam a linha de uma tabela como unidade de armazenamento.

Não precisa de nenhuma sub-camada física de armazenamento em especial, pode ser implementada em cima de uma base de dados relacional (ex: o Oracle disponibiliza isso), hierárquica, orientada ao objecto ou em cima de qualquer formato proprietário de armazenamento.

Como é utilizado XML, e devido à sua extensibilidade, é possível estruturar e adicionar informação de uma forma mais flexível do que a oferecida por um modelo relacional.

De modo a garantir que, a informação contida na base de dados, tem uma determinada estrutura e respeita determinadas condições, pode-se utilizar XML *Schemas* para fazer a validação.

A existência de linguagens de interrogação e interacção, como o XQuery [6],[7],[8] e o XUpdate [9],[10], permitem integrar em plataformas XML a interacção directa com as bases de dados, sem recurso a linguagens de programação convencional.

## 3 Modelo desenvolvido

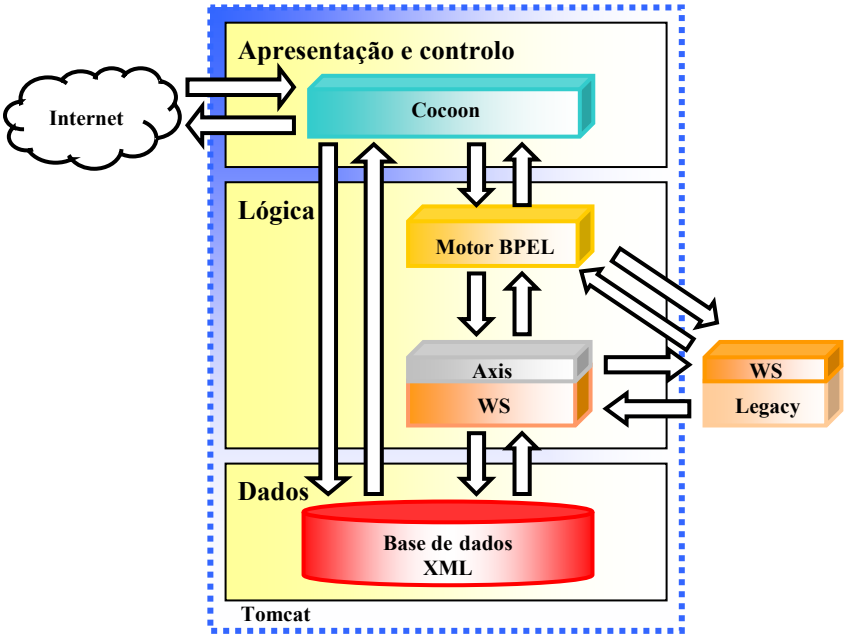


**Fig. 1.** Arquitectura do modelo

O modelo desenvolvido é constituído pelas camadas de, apresentação e controlo, lógica e dados. A camada de apresentação e controlo, é implementada pelo Cocoon[11],[12] e consiste em atender pedidos Web e interagir com o sistema de negócio. A camada de lógica está distribuída entre o motor BPEL e os serviços Web. É nesta camada de lógica que vai ser especificado e implementado o processo de negócio. A camada de dados é implementada numa base de dados XML.

O módulo do Cocoon é responsável pelas componentes de interacção com o utilizador, interface com os mecanismos de *backend* e pela formatação dos dados para apresentação. Quando o utilizador faz um pedido que envolve uma interacção processual assíncrona, o Cocoon selecciona uma *pipeline* que começa com um *generator* do tipo *Web services* que por sua vez interage com o motor BPEL. Quando o pedido do utilizador é para o acesso a informação já disponível, é usada uma *pipeline* do Cocoon cujo o *generator* é do tipo XQuery e é usado para interrogar directamente a base de dados XML e obter a resposta.

O motor BPEL implementa a lógica de alto nível, e fornece o suporte para as interacções assíncronas (computacionais ou humanas). A lógica de baixo nível, isto é, quando é necessário recorrer a operações ou procedimentos cujo nível de detalhe está muito associado à plataforma tecnológica, é implementado em classes Java disponibilizadas como serviços Web na plataforma Axis [13],[14]. Desta forma conseguimos colocar a lógica do negócio expressa apenas por uma linguagem de alto nível, o BPEL (usando editores gráficos). Remetemos para a programação convencional os pormenores de implementação, que devem ser transparentes para a lógica de negócio. Muitas vezes parte desta camada lógica já existe, em aplicações convencionais, nas plataformas tecnológicas mais diversas e, com este modelo, pode ser reutilizada.

A interacção com aplicações já existentes é efectuada recorrendo ao uso de serviços Web, que fazem a interface a essas aplicações.

Visto que todas as tecnologias usadas são baseadas em XML, e os dados a armazenar são também eles documentos XML, a opção por uma base de dados centrada ao documento, como é o caso das bases de dados XML, parece-nos ser a opção adequada, por já existirem diversas linguagens de interrogação para XML e diversas implementações.

## 4 Implementação do Modelo

Foi implementado um módulo de uma secretaria electrónica para disponibilização do serviço de emissão de certidões que, além de permitir efectuar os pedidos e acompanhar os estados dos mesmos, envolve ainda um processo de negócio composto pelas acções de pedir, pagar e emitir as certidões.

### 4.1 Apresentação e controlo

O Cocoon é uma ferramenta desenvolvida em Java que corre num contentor Web (ex: Tomcat) e permite definir *pipelines* compostas por sequências de acções de obtenção, transformação e serialização de documentos XML.

As *pipelines* são definidas num ficheiro de nome *sitemap.xmap* e são seleccionadas pelo Cocoon em função do tipo de pedido HTTP [15].

A selecção da *pipeline* que vai ser executada é feita por uma acção de *matching* entre o URL do pedido e um padrão definido no elemento `<map:match pattern="URL"/>`.

As *pipelines* contêm pelo menos o *generator* que acede à fonte de informação e gera uma sequência de eventos SAX [16],[17]. O *serializer* consome eventos SAX e produz o resultado. Opcionalmente, uma *pipeline* pode conter um ou vários *transformers*.

O Cocoon disponibiliza um conjunto de *generators* de vários tipos, entre os quais, o XQuery e o XSP [18]. O *generator* do tipo XQuery processa documentos definidos em XQuery, e o *generator* do tipo XSP permite gerar dinamicamente, de uma forma análoga aos JSP e aos PHP, documentos XML. Os *transformers* permitem, como o próprio nome indica, transformar o resultado do estágio anterior. Esta operação é normalmente conseguida pelo uso de um ou vários *scripts* escritos em XSLT [19].

#### 4.1.1 Interface

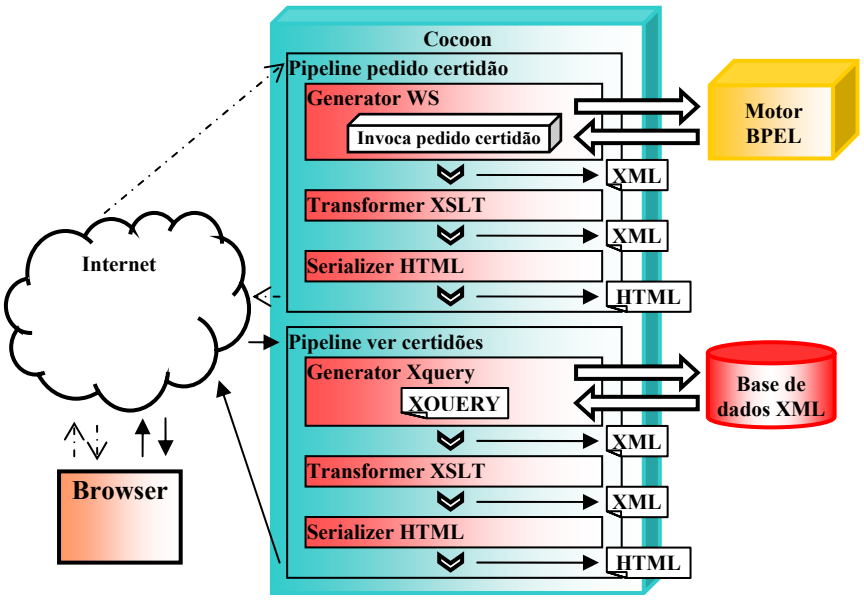


**Fig. 2.** Módulo de Interface ao utilizador

Quando um utilizador, por intermédio do browser, decide interagir no processo de pedido de certidão, é gerado um pedido HTTP que é recebido pelo Cocoon. Em seguida a *pipeline* referente à interacção começa a ser executada. Primeiro é solicitado o *generator* que invoca um serviço Web, disponibilizado pelo motor BPEL, para se proceder ao desencadear da acção. A resposta será um documento XML que irá ser transformado num documento HTML [20] pelo(s) *transformer(s)* XSLT. Finalmente o *serializer* envia para o browser a página HTML.
Por outro lado quando há um pedido para consulta do estado do pedido de certidão, é desencadeada uma outra *pipeline* cujo o *generator* acede e executa uma XQuery directamente sobre a base de dados. Os resultados desta *query* são, à semelhança do caso anterior, transformados, serializados e devolvidos ao cliente.

Em seguida, vão ser mostrados dois exemplos de pipelines, a primeira utiliza um *generator* que acede directamente à base de dados XML, utilizando XQuery, a segunda, utiliza um *generator* do tipo *serverpages* (XSP), que retorna um documento XML resultante da invocação de um serviço Web, disponibilizado pelo motor BPEL.

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
<map:pipelines>
  <map:pipeline>
    <map:match pattern="dadosPessoais.x">
        <map:generate src="dadospessoais\dadospessoais.xq"
                      type="xquery"/>
        <map:transform src="dadospessoais\dadospessoais.xsl"/>
        <map:serialize type="html"/>
    </map:match>
  </map:pipeline>
  <map:pipeline>
    <map:match pattern="emitir.x">
        <map:generate src="emitir\emitir.xsp"
                      type="serverpages"/>
        <map:transform src="emitir\emitir.xsl"/>
        <map:serialize type="html"/>
    </map:match>
  </map:pipeline>
</map:pipelines>
</map:sitemap>
```

**Fig. 3.** Definição das *Pipelines* (Sitemap.xmap)

Na Fig. 3 a primeira *pipeline* é executada sempre que o URL termina em `dadosPessoais.x`. Esta pipeline executa a XQuery definida no ficheiro `dadospessoais.xq`. O resultado desta query é transformado pelo XSLT presente no ficheiro `dadospessoais.xsl`.

A segunda *pipeline* invoca um *generator* do tipo XSP. Como o Cocoon não disponibiliza nenhum *generator* do tipo *Web services* utiliza-se um XSP que invoca um serviço Web. Na Fig. 4 apresenta-se o XSP para invocar o serviço Web disponível em `http://localhost/EmitirService`, e constrói-se a mensagem SOAP [21],[22] para a operação Emitir.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp:page language="java"
  xmlns:xsp="http://apache.org/xsp"
  xmlns:xsp-request="http://apache.org/xsp/request/2.0"
  xmlns:soap="http://apache.org/xsp/soap/3.0">
  <search-results>
    <soap:call
  url="http://localhost/EmitirService"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <Emitir>
        <soap:enc/>
        <numeroUtilizador xsi:type="xsd:int">
          <xsp-request:get-parameter name="numero"/>
        </numeroUtilizador>
        <ref xsi:type="xsd:string">
          <xsp-request:get-parameter name="ref"/>
        </ref>
      </Emitir>
   </soap:call>
 </search-results>
</xsp:page>
```

**Fig. 4.** XSP para invocar o serviço WEB "EmitirService"

Neste XSP faz-se uso da *SOAP Logicsheet* [24], para definir a mensagem SOAP e invocar o serviço WEB disponibilizado pelo BPEL.

### 4.2 Lógica

A lógica na nossa implementação está dividida pela lógica do processo de negócio implementada em BPEL e pela lógica de mais baixo nível implementada em serviços Web desenvolvidos em Java.

#### 4.2.1 A lógica do processo de negócio

O BPEL é uma linguagem XML para especificar o comportamento de um processo de negócio. É baseado em serviços Web, suporta interacções assíncronas e permite definir o controlo de fluxo. Um processo BPEL é definido em termos das suas interacções com outros serviços Web, através da definição de parcerias (*partners*). Um parceiro pode disponibilizar serviços Web para o processo BPEL, pode requerer serviços do processo BPEL e pode participar numa interacção bi-direccional (pedido/resposta) com o processo BPEL.

O BPEL orquestra serviços Web, especificando a ordem em que vai chamar uma colecção de serviços e atribui responsabilidades (papel) para cada um dos serviços aos parceiros. A definição de um processo BPEL consiste em definir as relações de parceria e a descrição do processo executável.

Na Fig. 5 apresentamos a lógica do processo de negócio de pedido de certidão que está implementada em BPEL.

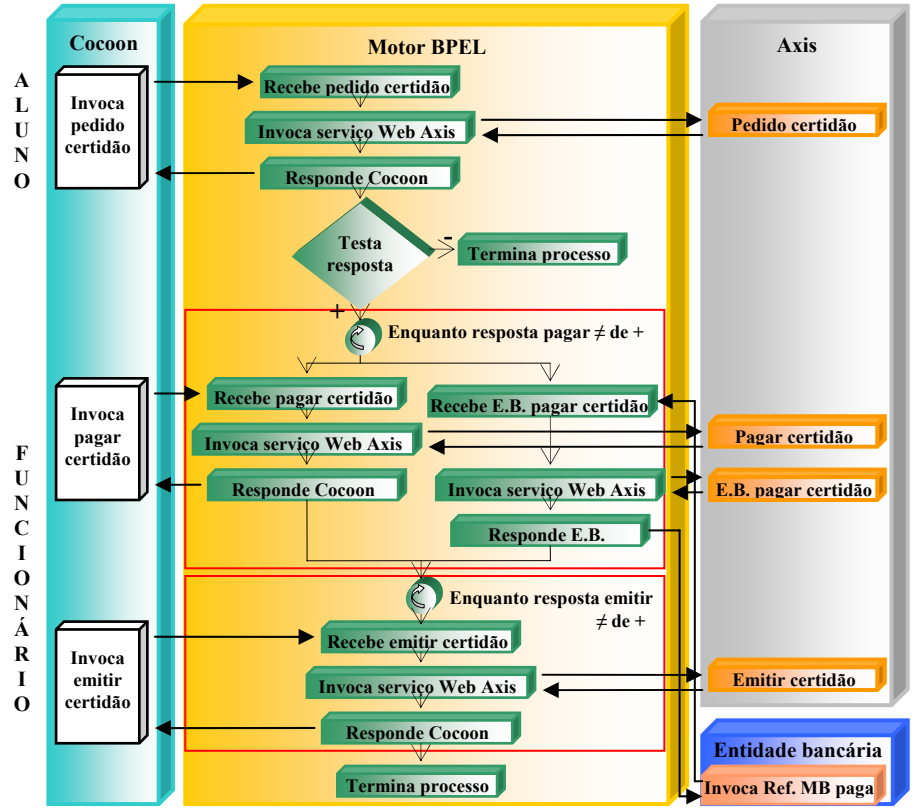


**Fig. 5.** Diagrama do processo de negócio (pedido de certidão)

Este processo de negócio está descrito num documento XML em linguagem BPEL. Este documento é utilizado pelo motor BPEL para pôr em prática o processo de negócio. Nesta implementação optou-se por utilizar o ActiveBPEL.

Na implementação efectuada, há várias etapas. Começa pelo pedido de certidão, que se traduz na criação de uma nova instância e de seguida invoca o serviço Web, que é responsável por verificar se a certidão tem condições para ser pedida. Qualquer que seja a resposta do serviço Web, é gerado um resultado para o Cocoon. Se a resposta foi negativa o processo BPEL termina de imediato. Como a certidão necessita ser paga, surgem duas soluções de pagamento, ou o utilizador paga a certidão ao balcão da secretaria e, nesse caso, o funcionário ao interagir com a plataforma vai invocar o serviço Web disponibilizado pelo motor BPEL, ou o utilizador paga por Multibanco e, por sua vez, a entidade bancária por intermédio de uma aplicação invoca outro serviço Web disponibilizado pelo motor BPEL. A terceira e última etapa do processo consiste na emissão da certidão. Esta interacção, depois de executar as acções correspondentes, termina o processo BPEL.

Note-se que estas 3 fases, acima descritas (pedido, pagamento e emissão), ocorrem em instantes de tempo distantes uns dos outros. Podem passar-se alguns dias entre cada uma delas.

Na Fig. 6 está apresentado um excerto do código BPEL do processo de negócio acima exposto. Este começa por declarar os *partnersLinks* envolvidos no processo, relativos ao pedido de uma certidão, quer do lado do Cocoon quer do lado do Axis. São definidas duas `correlationSet` para associar cada instância às operações respectivas. A sequência de operações inicia-se com a criação de uma instância do processo, quando o motor BPEL recebe uma mensagem do *partnerLink* `cocoonPedido` invocando a operação `pedirCertidao`. De seguida, o motor BPEL invoca a operação `efectuarPedido` através do *partnerLink* `axisPedido` que é responsável pelos pedidos de certidão, o qual responde com uma mensagem SOAP que é depois encaminhada para o *partnerLink* `cocoonPedido` através de.um `reply` à mensagem recebida.

```
<process name="certidoes"
   targetNamespace="http://gilberto.com/certidoes"
   xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
   xmlns:cocoonPedido="http://gilberto.com/cocoonPedido"
   xmlns:axisPedido="http://gilberto.com/AxisPedirCertidao">
<!—- Definição dos correlationSets -->
    <correlationSets>
        <correlationSet properties="cocoonPedido:ref" name="corrRef"/>
        <correlationSet properties="cocoonPedido:mb" name="corrMb"/>
    </correlationSets>
<!—- Definição dos partnerLinks envolvidos -->
    <partnerLinks>
       <partnerLink name="cocoonPedido"
           partnerLinkType="cocoonPedido:cocoonPedirCertidaoLT"
           myRole="pedirCertidaoProvider"/>
       <partnerLink name="axisPedido"
           partnerLinkType="axisPedido:axisPedirLT"
           partnerRole="pedirCertidaoProvider"/>
                .......
    </partnerLinks>
<!—- Definição das variáveis para guardar mensagens -->
    <variables>
       <variable name="cocoonPedInVariable"
           messageType="cocoonPedido:cocoonPedidoIN"/>
       <variable name="cocoonPedOutVariable"
           messageType="cocoonPedido:cocoonPedidoOUT"/>
       <variable name="axisPedInVariable"
           messageType="axisPedido:efectuarPedidoRequest"/>
       <variable name="axisPedOutVariable"
           messageType="axisPedido:efectuarPedidoResponse"/>
              .......
    </variables>
<!—- Conjunto de actividades executadas sequencialmente -->
    <sequence name="main">
       <receive name="Recebe_cocoon_pedido" partnerLink="cocoonPedido"
           portType="cocoonPedido:cocoonPedirCertidaoPT"
           operation="pedirCertidao" variable="cocoonPedInVariable"
           createInstance="yes"/>
                 .......
       <invoke name="Invoca_axis_pedir_certidao"
           partnerLink="axisPedido"
           portType="axisPedido:AxisPedirCertidao"
           operation="efectuarPedido"
           inputVariable="axisPedInVariable"
           outputVariable="axisPedOutVariable">
              <correlations>
                <correlation set="corrRef" initiate="yes" pattern="in"/>
                <correlation set="corrMb" initiate="yes" pattern="in"/>
              </correlations>
       </invoke>
                 .......
```

```
       <reply name="Responde_cocoon_pedido" operation="pedirCertidao"
           partnerLink="cocoonPedido"
           portType="cocoonPedido:cocoonPedirCertidaoPT"
           variable="cocoonPedOutVariable"/>
                .................
    </sequence>
</process>
```

**Fig. 6.** Excerto de código BPEL que processa pedidos de certidões

Na figura 7 apresenta-se uma mensagem SOAP pedido entre o Cocoon e o motor BPEL da operação WSDL `pedirCertidao` no código BPEL anteriormente apresentado.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <pedirCertidao
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:xsd="http://www.w3.org/1999/XMLSchema"
                xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
                  <numeroUtilizador xsi:type="xsd:string">
                      1050001
                  </numeroUtilizador>
                  <tipo xsi:type="xsd:string">
                      Matricula
                  </tipo>
         </pedirCertidao>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Fig. 7.** Mensagem SOAP enviada pelo Cocoon para o motor BPEL

A figura 8 mostra um excerto do documento WSDL do serviço Web `cocoonPedido`, onde é definida a estrutura das mensagens SOAP de entrada e saída.

```
<wsdl:message name="cocoonPedidoIN">
   <wsdl:part name="numeroUtilizador" type="xsd:string"/>
   <wsdl:part name="tipo" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="cocoonPedidoOUT">
   <wsdl:part name="resposta" type="xsd:string"/>
   <wsdl:part name="respdesc" type="xsd:string"/>
   <wsdl:part name="ref" type="xsd:string"/>
   <wsdl:part name="mb" type="xsd:string"/>
</wsdl:message>
```

**Fig. 8.** Estrutura das mensagens SOAP de entrada e saída do serviço Web `cocoonPedido`

Por último refira-se que no contexto do sistema implementado, em que os produtores e consumidores de mensagens são únicos e estão muito bem definidos dentro da organização, não foi utilizada a descoberta de serviços Web recorrendo ao UDDI [25]. Estes poderão ser utilizados no contexto organizacional, para introduzir um outro nível de suporte à dinâmica de negócio.

### 4.2.2 Serviços Web

O Axis é uma plataforma, que possibilita o desenvolvimento de serviços Web. Na nossa implementação, os serviços Web são componentes do sistema de negócio. São eles que representam um determinado papel dentro da lógica de negócio e cujo objectivo é efectuar acções, tarefas, etc.

Na arquitectura da secretaria electrónica, o módulo Axis, disponibiliza estes quatro serviços:

- Pedido de certidão. Recebe o pedido, verifica se há condições e regista o pedido na base de dados XML.
- Pagar certidão. Altera o estado do pedido para pago. É usado apenas para certidões pagas ao balcão.
- Receber pagamento da entidade bancária (E.B.). Altera o estado do pedido para pago. É usado apenas para certidões que foram pagas através de entidades bancárias ou em caixas automáticas (ex: Multibanco).
- Emitir certidão. Altera o estado do pedido para emitido.

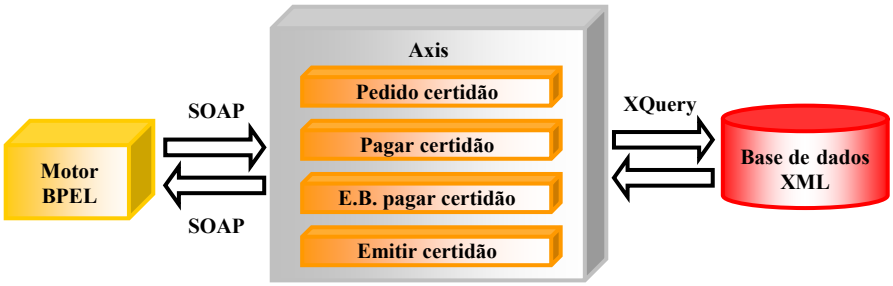


**Fig. 9.** Serviços Web

Na Fig. 9 estão representados os serviços Web implementados, bem como as interfaces ao motor BPEL e à base de dados XML. Note-se que as mensagens trocadas com motor BPEL, são obviamente mensagens SOAP, mas a interface à base de dados é feita pela execução de *querys* usando a linguagem XQuery.

## 4.3 Armazenamento de dados

A camada de dados foi implementada recorrendo à base de dados eXist [23]. O eXist é uma base de dados XML, que permite ser acedida por intermédio de interfaces como XML:DB, XML-RPC, SOAP e WebDAV.

As *querys* à base de dados são feitas na linguagem XQuery, quer pelo *generator* do Cocoon, quer pelas classes Java dos serviços Web. As actualizações e inserções são feitas, pelas classes Java, usando a linguagem XUpdate.

## 5 Conclusões

Foi proposto um modelo de uma plataforma de negócio baseada em XML, que permite integrar tecnologias orientadas ao serviço.

Foi feita uma implementação de um módulo de uma secretaria electrónica seguindo este modelo. A implementação permitiu-nos demonstrar a exequibilidade do mesmo e ser capaz de resolver um problema exemplificativo, o caso do pedido de certidões. Durante o processo de análise, em que a visão do problema foi amadurecendo, foi fácil introduzir alterações à lógica de processo de negócio, actuando na generalidade das situações na lógica de alto nível.

Este tipo de abordagem traz grandes vantagens a nível da reestruturação dos processos de negócio, quer a nível da flexibilidade em adaptar-se as variações dos requisitos, quer pela integração com os sistemas informáticos já existentes.

Por último, resta-nos referir que não foi objectivo deste trabalho averiguar a validade do modelo do ponto de vista da segurança e desempenho. Estes aspectos serão objectivos do trabalho futuro.

## Referências


1. Web Services, Service Oriented Architecture (SOA). http://webservices.xml.com.
2. Mark Colan, Service-Oriented Architecture expands the vision of Web services, 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-soaintro.html.
3. IBM, Service-Oriented Architecture (SOA). http://www-306.ibm.com/software/info/openenvironment/soa/.
4. World Wide Web Consortium, eXtensible Markup Language (XML) 1.0, 2004. http://www.w3c.org/xml.
5. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, Business Process Execution Language for Web Services Version 1.1, 2003. ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf.
6. Dr. Michael Kay, Learn XQuery in 10 Minutes,2005. http://www.stylusstudio.com/xquery_primer.html.
7. World Wide Web Consortium, XQuery 1.0: An XML Query Language, 2005. http://www.w3.org/TR/xquery/.
8. Bob DuCharme, Getting Started with XQuery, Part 2, 2005. http://www.xml.com/lpt/a/2005/03/23/xquery-2.html.
9. Chimezie Ogbuji, Editing XML Data Using XUpdate and HTML Forms (XUpdate), 2002. http://www.xml.com/pub/a/2002/06/12/xupdate.html.
10. Andreas Laux, Lars Martin, XUpdate Syntax and Document Type Definition. http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html.
11. Apache Cocoon Project, Cocoon. http://cocoon.apache.org/.
12. Generating Web content with Cocoon. http://www.webreference.com/xml/column52/.
13. Apache Web Services, Axis. http://ws.apache.org/axis/.

14. Dennis Sosnoski, Apache Axis SOAP for Java, 2002.
http://www.sosnoski.com/presents/java-xml/axis/.
15. World Wide Web Consortium, HTTP - Hypertext Transfer Protocol.
http://www.w3.org/Protocols/.
16. Simple API for XML (SAX).
http://www.saxproject.org/.
17. Uche Ogbuji, Using SAX for Proper XML Output, 2003.
http://www.xml.com/pub/a/2003/03/12/py-xml.html.
18. Apache Cocoon Project, eXtensible Server Pages, XSP.
http://cocoon.apache.org/2.1/userdocs/xsp/logicsheet.html.
19. World Wide Web Consortium, XSL Transformations (XSLT) version 1.0, 1999.
http://www.w3.org/TR/xslt.
20. World Wide Web Consortium, HyperText Markup Language (HTML) Home Page.
http://www.w3.org/MarkUp/.
21. W3Schools, XML, SOAP, Web Services.
http://www.w3schools.com.
22. World Wide Web Consortium, XML Protocol Working Group (SOAP).
http://www.w3.org/2000/xp/Group/.
23. Open Source Native XML Database, eXist.
http://exist.sourceforge.net.
24. Apache Cocoon Project, Logicsheet Concepts.
http://cocoon.apache.org/2.1/userdocs/xsp/logicsheet-concepts.html.
25. Using BPEL4WS in a UDDI registry.
http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-bpel-20040725.htm.