UNIVERSIDADE DE LISBOA

Faculdade de Ciências

Departamento de Informática

INTERNET DAS COISAS NOS PROCESSOS DE NEGÓCIO

Carlos Eduardo Correia Cândido

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitectura, Sistemas e Redes de Computadores

2013

UNIVERSIDADE DE LISBOA

Faculdade de Ciências

Departamento de Informática

INTERNET DAS COISAS NOS PROCESSOS DE NEGÓCIO

Carlos Eduardo Correia Cândido

DISSERTAÇÃO

Trabalho orientado por Profª. Doutora Maria Dulce Pedroso Domingos

e por Prof. Doutor Francisco Cipriano da Cunha Martins

MESTRADO EM ENGENHARIA INFORMÁTICA

Especialização em Arquitectura, Sistemas e Redes de Computadores

2013

Agradecimentos

Agradeço à minha orientadora, Dulce Domingos, e ao meu co-orientador, Francisco Martins, pela oportunidade de desenvolver este trabalho. A todos os meus colegas que ajudaram a manter-me na linha neste longo percurso académico. Colegas como o Marcos Vasco, Pedro Pereira, Jaime Vaz, Tiago Antunes, Gonçalo Gomes, Eduardo Matos, Rafael Soledade, João Ramos, Igor Antunes, Filipe Cabaço, Flávio Saraiva, Tiago Oliveira, Ricardo Mendes e Ricardo Fonseca.

Agradeço igualmente à minha família porque me deu a oportunidade de me manter nos estudos durante estes longos anos.

Por fim agradeço à FCT que através do programa de financiamento multianual do LaSIGE permitiu que me fosse atribuída uma bolsa de investigação pelo projecto PATI (PTDC/EIAEIA/103751/2008).

Para ti

Resumo

A Internet das Coisas (IoT) é um paradigma emergente que pretende integrar as "coisas" do mundo real no mundo informático. Os processos de negócios conseguem beneficiar deste paradigma utilizando informação de contexto relevante durante a sua execução. Esta informação, disponibilizada por exemplo sensores, permite que os processos reajam a alterações no contexto em tempo real.

Web Service Business Process Execution Language (WS-BPEL) é uma linguagem utilizada para definir processos através da composição de serviços *web*. Esta linguagem é baseada num paradigma de orquestração de serviços, o que dificulta a inclusão da informação de contexto nos processos. Assim, o modelador de processos é obrigado a incluir operações adicionais para obter informação da IoT. Para além disto, os construtores da linguagem são limitados no que diz respeito ao refrescamento desta informação, que poderá estar em constante actualização.

Esta dissertação tem como objectivo estender a linguagem WS-BPEL e o seu ambiente de execução de forma a facilitar o desenvolvimento de processos de negócio que usam informação de contexto. Com esta extensão, a linguagem passará a incluir o conceito de variáveis de contexto que mantêm os valores dos sensores e cuja actualização será feita automaticamente através do paradigma publicação/subscrição ou periodicamente, através do paradigma pedido/reposta. Esta extensão inclui também um novo construtor, o *When*, que permite a definição do comportamento dos processos para excepções previstas com condições que incluem as variáveis de contexto.

Por último, reagir a eventos não previstos requer o suporte a alterações *ad-hoc* aos processos. De forma a satisfazer estes requisitos são definidas as condições de correcção das instâncias de processos para a extensão proposta nesta dissertação.

Palavras-chave: Internet das Coisas, Processos de negócio, WS-BPEL, Serviços *Web*

Abstract

The Internet of Things (IoT) is an emerging paradigm that aims at integrating the state of "things" of the real world into the computer world. Business processes can benefit from this paradigm using relevant context information during their execution. This information, provided for instance by sensors, allows processes to react to situations arising in real time.

The Web Service Business Process Execution Language (WS-BPEL) is a language used to define processes through the composition of web services. This language is based on the service orchestration paradigm, which makes the inclusion of context information in the processes difficult. Thus, the process modeler is forced to include additional operations to obtain information from the IoT. In addition, the language constructs are limited with regard to refresh this information, which may be continuously updated.

The objective of this thesis is to extend the WS-BPEL language and its runtime environment to facilitate the development of business processes that use context information. With this extension, the language includes the concept of context variables that store sensor values and whose update is made automatically through the publish/subscribe paradigm or, periodically, through the request/reply paradigm. This extension also includes a new constructor, the W*hen*, allowing the definition of expected exceptions with conditions that include context variables.

Finally, responding to unexpected events requires the support of *ad-hoc* changes of processes. To meet these requirements, we define the process instances correction conditions for the extension we propose in this dissertation.

Keywords: Internet of Things, Business Process, WS-BPEL, Web Services

Conteúdo

Lista de Figuras

Lista de Listagens

Capítulo 1

Introdução

A Internet das Coisas (IoT, do inglês *Internet of Things*) pretende aproximar o mundo real aos sistemas de informação. Um exemplo básico da aplicação deste paradigma é a tecnologia de código de barras, cuja aplicação transporta para os sistemas de informação contexto sobre, por exemplo, quantidades de produtos em armazém. Actualmente a IoT vai mais longe, aplicando o seu conceito sobre redes de sensores sem fios (do inglês, Wireless Sensor Networks). Estes sensores podem fornecer a mais variada informação (e.g. leitura de temperatura, humidade, luminosidade, etc.). Esta informação de contexto pode ser usada em muitas áreas e, em particular, nos processos de negócio que são o foco desta dissertação.

Nos processos de negócio, a IoT apresenta-se como uma vantagem em termos competitivos, já que a informação de contexto disponibilizada pode ser utilizada para optimizar a sua execução e para permitir a sua adaptação em tempo real a alterações do ambiente, ou seja, promover processos de negócio mais reactivos baseados em informação actualizada.

Actualmente, a WS-BPEL [1] é a norma de-facto utilizada para definir processos através da composição de serviços *web*. Tendo em vista a integração da IoT nos processos de negócio, assistimos à disponibilização da sua informação através de serviços web, os quais podem ser suportados directamente nos sensores ou através de *middleware* [2]. Esta abordagem apresenta a vantagem adicional de encapsular as especificidades dos vários tipos de sensores. Sendo a WS-BPEL baseada na orquestração de serviços, a informação dos sensores disponibilizada através de serviços *web* pode ser facilmente utilizada nos processos de negócio, utilizando um paradigma síncrono de pedido/resposta. No entanto, com este paradigma, se o modelador pretender que um processo tenha informação actualizada sobre as alterações que ocorrem no ambiente, terá de incluir no processo as actividades que permitam obter periodicamente estas informações. Deste modo, para além de definir a lógica principal do negócio, o modelador tem também de definir explicitamente a interacção com os sensores.

Por outro lado, a linguagem não tem construtores para tratar excepções previstas dependentes de condições que incluam valores de variáveis. Já as excepções não previstas, ao nível das instâncias, são tratadas através de alterações *ad-hoc* à instância em execução. Existem alguns trabalhos que definem os critérios de correcção para determinadas actividades da WS-BPEL [1]. Neste trabalho pretende-se estender estes critérios de forma a contemplar as extensões que são propostas.

A nossa motivação tem como base a falta de mecanismos da linguagem que permitam uma fácil modelação dos processos que utilizam informação de contexto assim como o uso dessa mesma informação dentro dos processos. Como tal, o nosso objectivo é criar uma extensão à linguagem fornecendo mecanismos que facilitem a modelação da interacção entre os processos e as fontes de informação de contexto.

Outro tema considerado é o tratamento de excepções que podem ocorrer durante a execução dos processos. Excepções podem ser previstas ou não previstas. Relativamente às previstas, abordamos este tema criando um novo construtor na linguagem, o *When*, que permite a um modelador definir o seu processo com lógica para reagir a excepções que possam ocorrer derivados de dados lidos dos sensores. Já o tratamento de excepções não previstas, este tema é abordado apresentando um critério de correcção para aplicar nas alterações *ad-hoc* para que estas mantenham a correcção das instâncias para as extensões desenvolvidas.

A nossa solução é suportada numa extensão à linguagem de forma a adicionar dois novos conceitos:

1) Variável de Contexto - Este novo tipo de variável representa o valor actual de uma determinada fonte de contexto que disponibiliza a sua informação utilizando os paradigmas de publicação/subscrição ou pedido/resposta em serviços *web*,

2) Construtor *When* – Este novo construtor estruturado permite aos processos reagir a alterações de valores nas variáveis e consequentemente nos sensores. O aumento da expressividade da linguagem permite ao modelador definir os processos de forma mais compacta.

O último ponto a ser abordado é a definição do critério de correcção para que se possa aplicar alterações às instâncias de processos de negócio em execução com as extensões desenvolvidas neste trabalho mantendo a sua correcção.

1.1 Objectivos

De seguida são apresentados os objectivos que estão na origem desta dissertação.

Modelação facilitada: Pretende-se tornar mais fácil para um modelador incluir e utilizar informação de contexto nos seus processos de negócio para que estes executem usando informação actualizada.

Comunicação facilitada: Fornecer mecanismos de comunicação transparentes com os sensores para que um modelador possa definir o seu processo incluindo variáveis de contexto sem ter de se preocupar com as interacções de subscrição/ publicação ou pedido/resposta.

Processos mais reactivos: Pretende-se adicionar à linguagem uma nova forma de utilizar esta informação que permita ao processo tornar-se mais reactivo a alterações de valores nas variáveis de contexto. Pretendemos portanto permitir a um modelador, sem esforço, utilizar um novo construtor na definição do processo que permita definir excepções previstas, cujas condições incluam variáveis de contexto.

Alterações ad-hoc: Para os processos poderem responder a excepções não previstas pretende-se contribuir com um critério de correcção que possa ser usado nas alterações *ad-hoc* para que seja mantida a correcção das instâncias que usem a extensão proposta neste trabalho.

1.2 Contribuições

Os objectivos traçados para esta dissertação resultam nas seguintes contribuições:

1. Inclusão de informação de contexto nos processos de negócio de forma simples, através de uma extensão à linguagem WS-BPEL que cria o conceito de variável de contexto. A este novo tipo de variável está associado um mecanismo que é responsável por efectuar o pedido de subscrição aos sensores e receber as notificações num modelo de publicador-subscritor, ou invocar a operação de obtenção do valor mais recente no modelo de actualização pedido/resposta e, por fim, actualizar as variáveis de contexto. Com isto reduzimos o esforço de modelar processos que contêm informação de contexto.
2. Modelação de processos de negócio mais reactivos a alterações de valores nas variáveis através do novo construtor *When*, introduzido na mesma

extensão do ponto anterior. Com este conceito, um modelador pode definir, de forma simples, uma nova forma de reagir a eventos previstos que ocorram nos sensores.

3. Apresentação de um critério de correcção para as alterações *ad-hoc* às definições das instâncias de processos de negócio em execução na linguagem WS-BPEL e em particular às extensões desenvolvidas. Este critério garante a correcção entre o histórico de execução de uma instância cuja definição contém extensões desenvolvidas neste trabalho e a sua nova definição, obtida através de uma alteração *ad-hoc*.

1.3 Estrutura do documento

Este documento está organizado da seguinte forma:

- **Capítulo 2** – Expõe os conceitos básicos sobre as tecnologias usadas neste trabalho assim como o estado de arte.
- **Capítulo 3** – Apresenta a extensão à linguagem. Para cada componente desta extensão, é dada uma explicação sobre quais os seus requisitos e como é que estendemos a linguagem de forma a incluir as nossas extensões. De seguida é exposta a solução para a implementação. Para além da extensão à linguagem são abordadas as alterações *ad-hoc* aos processos de negócio. É apresentado um critério de correcção que deve ser usado para aplicar nas alterações aos processos para que a nova definição seja compatível com o estado da instância.
- **Capítulo 4 –** Apresenta o protótipo assim como os testes e resultados do mesmo.
- **Capítulo 5 –** Apresenta as conclusões e o trabalho futuro.

Capítulo 2

Conceitos e Estado de Arte

Neste capítulo são apresentados os conceitos básicos usados neste trabalho. Estes conceitos incluem uma descrição da linguagem que vamos estender, a WS-BPEL, assim como o motor de execução seleccionado. Por fim é apresentado o trabalho relacionado.

2.1 Conceitos

Nesta secção apresentamos uma descrição sobre as várias normas usadas neste trabalho e sobre o motor de execução Apache ODE.

2.1.1 Web Service Addressing

Web Services Addressing (*WS-Addressing* ou *WSA*) [3] é uma especificação que define mecanismos de transporte neutro para endereçar os serviços *Web* e as mensagens trocadas. Define elementos usando *Extensible Markup Language* (*XML*) [4] para identificar univocamente pontos terminais (do inglês *Endpoint*) dos serviços Web de forma a garantir o sucesso em troca de mensagens ponto a ponto. É utilizado pelas entidades para identificarem serviços na web através de um endereço formado pela estrutura de dados denominada por *EndPointReference (EPR)*. O EPR, através do campo *address*, representa o endereço de um serviço web e disponibiliza campos opcionais, como o *ReferenceParameters* para por exemplo, distinguir EPRs com o mesmo *address*.

2.1.2 Web Service Description Language

WSDL [5] é uma linguagem baseada em *XML* para descrever serviços de rede como um conjunto de terminais que operam por troca de mensagens. Uma entidade que

pretenda expor operações através de um serviço web deve disponibilizar um documento na linguagem *WSDL* para expor as suas operações e as mensagens que troca. Um serviço descrito em *WSDL* inclui o nome do serviço, o seu endereço (*EndPointReference*), o *Binding* do serviço que define os protocolos usados para a troca das mensagens, as operações disponibilizadas, o formato das mensagens trocadas nas operações e as possíveis faltas que podem ocorrer na sua invocação. Um documento WSDL contém as informações necessárias para um cliente invocar operações num serviço *web*.

2.1.3 Serviços na Web

Os serviços na Web têm-se demonstrado como ferramentas bastantes promissoras para a implementação de sistemas orientados a serviços. Este género de serviços fornece uma base para o desenvolvimento de aplicações distribuídas, disponibilizadas na *Internet*, que utilizam protocolos e padrões *Web* para a comunicação. As interfaces destes serviços são definidas através da linguagem WSDL que especifica o contrato entre o fornecedor e o cliente dos serviços e as operações que este fornece.

Uma das formas de comunicação dos serviços é através Simple Object Access Protocol (*SOAP*) [6] que é um protocolo que utiliza *XML* para definir uma norma extensível para troca de mensagens.

2.1.4 Web Service Notifications

Web Services Notifications (*WSN*) [7] define um conjunto de normas que têm como objectivo definir a forma como os serviços *Web* interagem usando notificações ou eventos. Definem padrões para comunicação no paradigma de Publicador/Subscritor para serviços *Web* onde uma entidade pode publicar informações para outras entidades sem as ter de conhecer previamente.

As especificações fornecem interfaces para os intervenientes, descritas em *WSDL*. Destas interfaces destacam-se a do serviço publicador (*NotificationProducer*) e a do serviço que recebe notificações (*NotificationConsumer*). Estas especificam as operações mínimas que um serviço publicador deve disponibilizar (*Subscribe* e *GetCurrentMessage*) e o mesmo acontece para o serviço que recebe as notificações (*Notify*). Numa invocação da operação *Subscribe*, um subscritor deve indicar, entre outras coisas, o EPR para onde devem ser enviadas as notificações. Quando ocorre uma subscrição é gerado um EPR que identifica essa subscrição univocamente. A troca de mensagens pode ser verificada no diagrama de sequência apresentado na Figura 1.

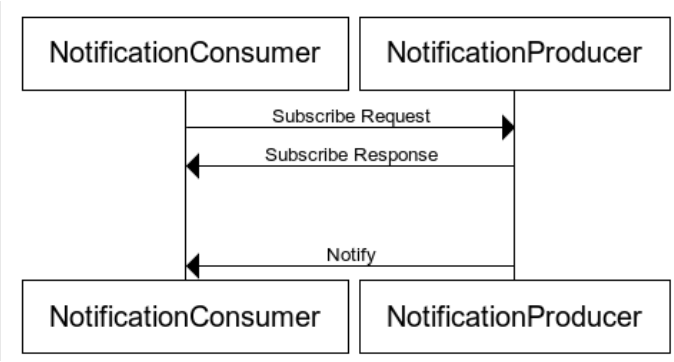

Figura 1 - Diagrama de sequência da troca de mensagens efectuada entre o subscritor e o publicador na norma *WSN*

2.1.5 Web Service Business Process Execution Language

Web Services Business Process Execution Language (WS-BPEL) [1] é uma linguagem definida pela norma da OASIS para descrever processos de negócio através de composição de serviços web. Um processo de negócio é composto por dois elementos: um ficheiro escrito em *WSDL* que descreve as operações fornecidas pelo processo de negócio (as estruturas de dados trocadas nas mensagens, os endereços dos serviços, etc.) e um ficheiro escrito em WS-BPEL que descreve o processo de negócio.

Um processo de negócio é obtido pela composição de actividades. Existem actividades de controlo de fluxo (*If*, *While*, *Scope*, *Flow*, etc.), actividades de comunicação (*Receive*, *Reply*, *Invoke*, etc.), atribuição de valores (*Assign*), gestão de faltas (*Throw*, *Rethrow*, etc.), entre outras.

É possível declarar variáveis de tipos primitivos, complexos (compostos por vários tipos de dados primitivos) e mensagens. As variáveis de mensagens são usadas quase exclusivamente em actividades de comunicação. As variáveis podem ser globais ou locais, se declaradas num âmbito (*Scope).*

Os *PartnerLinks* servem para as actividades de comunicação saberem que operações representam ou devem invocar. Um *PartnerLink* está associado a um *PartnerLinkType* que por sua vez declara papéis (do inglês Roles) associados a tipos de portas. Por norma o *PartnerLinkType* é declarado no documento WSDL que descreve as portas do serviço às quais este está associado. Ao ser declarado, um *PartnerLink* indica qual o seu papel e/ou qual o papel do seu parceiro. Uma actividade de comunicação de *input* selecciona uma operação da porta associada ao *MyRole* e uma actividade de *output* selecciona uma operação do *PartnerRole*.

De modo a distinguir as instâncias dos processos usa-se o mecanismo *Correlation*. Um *correlationSet* é definido por 1) o tipo de dados simples que vai ser usado e 2)

conjunto de regras (uma por tipo de mensagem). Após estar definido, este é associado às actividades de comunicação onde se pretende utilizar o *correlationSet*. Cada *correlationSet* só pode ser inicializado uma vez, em caso de ser usado num *Invoke* com dois sentidos, deve-se definir se o *Correlation* é feito no envio, na resposta ou em ambos os sentidos. As propriedades dos *CorrelationSets* são declaradas num documento WSDL e definem, através de *XPATH*, os elementos das mensagens trocadas pelo processo que identificam cada conversação (ie, cada instância).

A norma WS-BPEL prevê a possibilidade de ser estendida. Com este objectivo foram definidos os seguintes mecanismos:

- **Actividades de extensão:** São actividades novas que podem ter o comportamento que o programador desejar. São declaradas no processo dentro do Construtor "ExtensionActivity" fornecido pela linguagem;
- **Operação de atribuição estendida:** Adiciona comportamento a uma actividade de atribuição;
- **Atributos de extensão:** Adicionam comportamentos através de novos atributos nos construtores;
- **Construtores/Elementos de extensão:** Adicionam comportamentos através de novos construtores.

Todas as extensões utilizadas num processo devem ser declaradas. Esta declaração é feita inserindo no construtor da linguagem *Extensions* os Namespaces associados às extensões e um atributo *MustUnderstand* com o valor "*yes"* ou "*no"* que indica se o motor de execução deve compreender as extensões.

Concretizar uma extensão pode ser feito em tempo de modelação ou em tempo de execução [8]. Quando realizada em tempo de modelação, uma extensão é concretizada através de uma transformação da linguagem que consiste em traduzir os elementos de extensão em elementos da norma da linguagem de tal forma que o motor de execução ignora a existência de extensões. Esta abordagem permite adicionar construtores à linguagem sem ter de alterar o motor de execução tornando as extensões portáveis entre motores de execução. No entanto está limitado à expressividade da linguagem.

Uma extensão em tempo de execução é feita alterando o motor de execução de forma a adicionar os novos conceitos e o comportamento. Uma extensão deste tipo permite adicionar comportamentos complexos à linguagem aumentando a sua expressividade mas a sua portabilidade fica bastante reduzida.

2.1.6 Apache Orchestration Director Engine

O *Apache Orchestration Director Engine* (*Apache ODE*) [9] é o motor de execução *open-source* e gratuito da fundação Apache. Utiliza a ferramenta Eclipse EE [10] com um *plugin* (Eclipse BPEL Designer [11]) para modelar os processos, inserir as definições no motor, testar as operações dos processos, etc. Actualmente o Apache ODE existe em duas versões (1.3.5 e 2beta) e ambas funcionam segundo a norma WS-BPEL versão 2. A diferença principal entre elas é que a versão beta já possui mecanismos próprios para criar actividades de extensão e extensões em *Assign*. Por outro lado, uma vez que é a versão beta, possui alguns erros. Optou-se por utilizar a versão estável, 1.3.5.

Outra característica do ODE que facilita a criação de mecanismos de extensão é o facto de fornecer *Listeners* que monitorizam eventos que ocorrem nos processos (e.g. eventos de nova instância de processo, execução de actividade iniciada, execução de actividade terminada, escrita de variável, etc.). Este mecanismo é fornecido através de uma interface java que ao ser implementada e registada no motor fica activa para todos os processos em execução. Para cada processo modelado pode-se seleccionar quais os eventos que se quer monitorizar através do ficheiro *deploy.xml* associado ao mesmo.

Adicionalmente, como o ODE é implementado sobre Axis2 [12] também permite adicionar serviços web, ou seja, o ODE tem a capacidade de fornecer processos de negócio e serviços *Web* em simultâneo.

A WS-BPEL permite quatro mecanismos de extensão os quais podem ser aplicados em tempo de modelação ou em tempo de execução. No caso do Apache ODE dois desses mecanismos, são fornecidos pelo motor na versão beta, como referido anteriormente, enquanto os outros dois (novos atributos de extensão e novos construtores/elementos de extensão) têm de ser implementados alterando o código fonte do ODE.

2.2 Estado de Arte

Nesta secção é apresentado o estado de arte. Começamos por abordar o tema de como disponibilizar e utilizar informação de contexto nos processos de negócio e de seguida apresentamos os mecanismos que permitem tratar as excepções previstas e as excepções não previstas.

2.2.1 Informação de contexto nos processos de negócio

O uso de informação do mundo real em sistemas de informação pode ser bastante benéfico para as mais variadas áreas e é devido a esse factor que actualmente é uma área em investigação.

No nosso trabalho utilizamos contexto com o mesmo significado de Allen George *et al*. [13], [14]. Estes autores descrevem contexto como um estado do ambiente, o qual é externo ao processo, cujo valor é alterado de forma independente do ciclo de vido do processo e cujo valor pode influenciar a sua execução.

Tradicionalmente, a informação de contexto é obtida de acordo com um paradigma síncrono de pedido/resposta em determinados pontos dos processos de negócio. Esta informação pode ser utilizada para, por exemplo, determinar os serviços que compõem os processos [15], seleccionar, de entre várias, a implementação de um serviço específico [16] ou determinar se um serviço deve fazer parte de uma composição [17].

Em [18], os autores propõem uma extensão à WS-BPEL, designada por Context4BPEL, com o objectivo de modelar explicitamente a forma como o contexto influência os fluxos de trabalho. A Context4BPEL é definida de acordo com os mecanismos de extensão da norma WS-BPEL. Esta extensão inclui mecanismos para: (1) gerir eventos de contexto, ou seja, permitir a recepção assíncrona de eventos; (2) interrogar dados do contexto e (3) avaliar condições de transição baseadas em dados de contexto. Esta extensão permite a um processo ser executado quando ocorrem determinados eventos ao contrário da abordagem típica de execução após invocação directa por um cliente. A Context4BPEL estende ambos os ambientes de modelação e de execução, e a gestão da informação de contexto está dependente da plataforma Nexus.

Em [19], os autores propõem uma extensão à WS-BPEL que inclui variáveis passadas por referências. Com este tipo de variáveis, os serviços podem trocar entre si apontadores para variáveis em vez dos seus valores. Os apontadores são representados através de EPRs. De acordo com o valor de um dos atributos da extensão, as referências são avaliadas (1) aquando da activação do *Scope*, (2) antes de as variáveis serem usadas, (3) periodicamente ou (4) através de um evento enviado de um serviço exterior. A definição do processo é transformada em WS-BPEL normalizado em tempo de desenho, substituindo as referências por variáveis WS-BPEL, inserindo as ligações aos parceiros e as actividades de interacção. O tipo (4) de avaliação de referências é semelhante ao objectivo proposto neste trabalho: a transformação adiciona um construtor *onEvent* à definição de processo. No entanto, estes autores não indicam como é que o serviço externo endereça o evento ao serviço web correspondente ao *onEvent*. Adicionalmente,

a avaliação de referências está dependente do serviço RRS (Reference Resolution Service), um serviço específico da plataforma proposta por estes autores.

George *et al*. [13], [14] propõem também uma solução baseada em variáveis de contexto. Este contexto é disponibilizado segundo a norma *WS-Notifications* onde a fonte de contexto funciona como um publicador de notificações. Do lado dos processos, o contexto é introduzido através de uma extensão à WS-BPEL que introduz o conceito de variável de contexto. Esta variável de contexto é obtida através de atributos de extensão aplicados ao construtor da variável, numa extensão ao motor de execução ActiveBpel. Quando um processo usa variáveis de contexto associadas a uma actividade de *Invoke*, o motor de execução efectua a subscrição ao publicador recorrendo a implementações de clientes fornecidos pela ferramenta Apache Muse [20] e fica responsável pela actualização das variáveis quando ocorrem notificações. Esta abordagem reduz o esforço de modelação, no entanto a definição do processo tem de conter explicitamente a operação de *Invoke* com a variável de contexto associada para que seja efectuada a subscrição. Para além disso, a portabilidade desta extensão acaba por ficar limitada por esta ter sido aplicada directamente no motor de execução.

No nosso trabalho desenvolvemos uma extensão à WS-BPEL que adiciona o conceito de variável de contexto que é actualizada segundo a norma de *WS-Notification*, que é a norma para publicação/subscrição para os serviços web, ou através do paradigma pedido/resposta utilizando a norma *WS-ResourceProperties* [21]. Esta extensão é aplicada em tempo de modelação, através de uma transformação, tornando-a portável, uma vez que não depende do motor de execução.

2.2.2 Definição de excepções previstas

Excepções previstas são aquelas que podem ser vistas como desvios previstos ao fluxo de um determinado processo. Estes desvios podem ser tratados directamente no fluxo adicionando-lhe caminhos alternativos e como tal não necessitam de intervenção humana. Aos processos de negócio descritos em WS-BPEL é-lhes permitido o tratamento de excepções previstas através dos construtores fornecidos para o efeito: *Throw*, *Rethrow* e *Catch*. Com estes construtores um processo pode, por exemplo, lançar uma excepção de um tipo específico usando o construtor *Throw* e com um construtor *Catch* tratar essa mesma excepção lançada. O tratamento poderá ser feito através de um fluxo adicional nesse *Catch*. No entanto estes construtores são limitados porque, por exemplo, se se pretender lançar uma excepção quando o estado de alguma variável atinge um determinado valor tem de se adicionar ao fluxo actividades para verificar periodicamente o estado desta variável, ou seja, tem que se efectuar espera activa.

George *et al*. [13] [14], para além das variáveis de contexto, propõem três extensões à linguagem que visam melhorar a expressividade da WS-BPEL permitindo o tratamento de excepções previstas. Estas extensões são: a actividade de extensão "ConditionWithTimeout" e os construtores "Context Handler" e "Context Handoff". Das três destacam-se as seguintes:

ConditionWithTimeout: Esta actividade de extensão bloqueia até a condição associada ser verdade ou o *timeout* definido ser alcançado. Se a actividade desbloquear pelo *timeout* e o valor da condição ainda for falso, as actividades definidas no *else* são executadas. Com esta extensão um modelador consegue evitar o uso de condições de espera activa, como o *RepeatUntil*, que executem a mesma lógica associada a esta actividade de extensão.

Context Handlers: Um *Context Handler* é um *handler* de eventos que executa quando ocorrem alterações de contexto. Ao contrário dos *handlers* de eventos que a linguagem fornece, estes não estão associados a portas nem a operações que podem ser invocadas por entidades externas ao processo. Com esta extensão, o autor permite aos processos reagirem a alterações nas variáveis de contexto, ou seja, reagir a excepções previstas no estado das variáveis de contexto.

Neste trabalho apresentamos um novo construtor que permite também tratar de excepções previstas no processo de negócio: o construtor *When*. Com este construtor um modelador pode declarar uma sequência de actividades que são executadas quando uma determinada condição se verifica. Esta extensão é obtida através de uma transformação à definição do processo em tempo de modelação ao contrário do trabalho desenvolvido pelos autores anteriores que aplicam esta extensão directamente no motor de execução.

2.2.3 Tratamento de excepções não previstas

Para que um sistema de gestão de processos de negócio permita adaptabilidade, deve ter a capacidade de permitir alterações dinâmicas aos processos. Existem dois tipos de alterações: alterações evolutivas e alterações *ad-hoc*. As alterações evolutivas modificam a definição de um processo tendo que se migrar as instâncias em execução para a nova versão ou afectando apenas as novas instâncias. Já as alterações *ad-hoc* modificam apenas o fluxo de execução de uma determinada instância.

Visando o tratamento de excepções não previstas, a adaptação através de alterações *ad-hoc*, tema que é foco deste trabalho, deve manter a correcção sintáctica das definições assim como a compatibilidade entre as instâncias dos processos e as suas definições.

Uma instância em execução diz-se correcta se o seu histórico de execução for compatível com a sua definição. Ao se efectuar uma alteração *ad-hoc* a definição de uma instância é alterada para uma nova definição e esta alteração só é correcta se a nova definição for capaz de produzir um histórico de execução idêntico à definição anterior. Se a nova definição for capaz de produzir um histórico igual ao histórico de execução da instância diz-se que a nova definição é compatível com o histórico de execução e que a alteração *ad-hoc* é correcta. Uma alteração *ad-hoc* para ser correcta está sujeita a critérios de correcção que são regras que definem como se deve efectuar uma alteração *ad-hoc* mantendo a correcção da instância. Alterar uma instância de um processo em execução só pode ser feito se a alteração mantém a correcção da instância.

Para este tipo de adaptação, são propostos dois tipos de mecanismos:

- Operações de alteração das instâncias, ou
- Criar uma variante da definição do processo para posteriormente serem migradas as instâncias.

No primeiro caso, esses mecanismos podem fornecer primitivas tão básicas como inserção ou remoção de actividades, variáveis, etc. De se notar que estas primitivas devem ter o cuidado de manter a correcção das instâncias alteradas para que não aconteça, por exemplo, remover uma actividade que já foi executada. Através de pré-condições associadas às primitivas consegue-se manter a correcção.

No segundo caso, inicialmente é alterada a definição do processo e, posteriormente, é efetuada a migração da instância para a nova definição. Esta migração só pode ser efetuada se a instância for compatível com a nova definição. Uma instância de processo *I* é compatível com a definição *D* se o histórico de execução de *I* puder ter sido obtido através de *D.*

Alterar a definição de um processo descrito em WS-BPEL pode ser alterar o fluxo do processo (adicionar, alterar ou remover actividades), adicionar ou remover variáveis, adicionar ou remover *PartnerLinks*, etc. Alterar o fluxo pode ser feito em três momentos distintos: quando a actividade ainda não executou; quando a actividade está em execução ou quando a actividade já executou.

O primeiro caso, alterar uma actividade quando ainda não executou, significa que a actividade em questão ainda não criou histórico de execução e como tal não faz parte do histórico de execução da instância. Neste caso pode-se efectuar uma alteração à actividade.

Quando uma actividade já executou o histórico gerado pela sua execução já faz parte do histórico de execução da instância não podendo então esta ser alterada. Por outro lado, uma actividade que esteja em execução pode também já ter gerado histórico

de execução. Por exemplo, uma actividade *While* que esteja na sua terceira iteração já gerou o histórico das duas iterações anteriores. Se for efectuada uma alteração *ad-hoc* que passe por adicionar ou remover actividades a esse ciclo *While*, essa alteração não é correcta porque o histórico de execução gerado pelas primeiras iterações do ciclo *While* da nova definição não corresponde ao que foi obtido através da definição inicial uma vez que a composição do *While* é diferente. Ou seja, a nova definição é incompatível com o histórico de execução.

Um critério de correcção básico para alterações *ad-hoc* a um processo de negócio descrito em WS-BPEL passa por não permitir que as actividades sejam alteradas quando estão em execução ou quando já foram executadas de forma a garantir que as novas definições consigam produzir sempre o mesmo histórico de execução que a definição original. Este critério de correcção garante que a nova definição do processo é compatível até ao ponto de execução actual da instância mas em certas situações este critério apresenta-se demasiado rígido.

Se considerarmos o exemplo da actividade *While*, o seu histórico de execução é formado pelas suas várias iterações. Se alterar esta actividade passar por adicionar ou remover actividades então o histórico da nova definição não será compatível com o da instância, mais concretamente essas iterações já executadas da actividade *While* não conseguirão ser obtidas através da nova definição cuja composição é diferente.

Em [22] *Reichert et al* relaxam o critério de correcção apresentado anteriormente de modo a permitir efectuar alterações em ciclos, mesmo quando as iterações já executadas do ciclo têm um histórico de execução que não possa ser produzido com a nova definição. Se o histórico de execução relativo às iterações já executadas for eliminado a nova definição consegue criar um histórico de execução compatível com o histórico da instância, que foi reduzido. Ou seja, uma alteração *ad-hoc* à instância *I* para a definição *D* é correcta se *D* for capaz de gerar o histórico de execução da instância *Hi* ou o histórico de execução reduzido da instância *HRi*.

Seguindo a mesma abordagem, em [23] os autores estendem os trabalhos de *Richard et al.* para que o critério de correcção inclua a actividade *onEvent*. Para este caso não só o histórico das actividades é descartado como o histórico das actividades que resultam do tratamento dos eventos assíncronos ocorridos anteriormente.

No nosso trabalho apresentamos um critério de correcção para as alterações *ad-hoc* que abordam as extensões desenvolvidas. É apresentado um critério de correcção que garanta que uma instância em execução que seja composta por variáveis de contexto ou construtores *When* possa ser submetida a alterações *ad-hoc* mantendo a compatibilidade

entre a nova definição e o histórico de execução gerado pela execução. Como as extensões são composições de actividades (como as *Assign*, *Invoke* ou o *eventHandler onEvent*) o critério de correcção aborda a correcção dessas várias actividades de forma a criar um critério de correcção para as extensões desenvolvidas.

2.3 Considerações finais

Nesta secção apresentamos a forma como a informação de IoT é disponibilizada e utilizada pelos processos de negócio. Mesmo nos trabalhos em que esta informação é mantida nos processos de negócio, as suas abordagens dependem de plataformas auxiliares ou características específicas do motor de execução.

No nosso trabalho desenvolvemos uma extensão à linguagem WS-BPEL para manter informação de contexto nos processos de negócio em variáveis de contexto. Estas variáveis são actualizadas segundo a norma *WS-Notifications*, que descreve a comunicação no paradigma publicação/subscrição para serviços *web*, e também segundo a norma *WS-ResourceProperties* que permite a actualização das variáveis através de o paradigma pedido/resposta.

Para além da variável de contexto, esta extensão contempla um novo construtor, *When*, que fornece aos modeladores a possibilidade de definirem no processo de negócio o comportamento para o tratamento de excepções previstas que envolvam o valor das variáveis de contexto.

Finalmente é apresentado como são suportadas as alterações *ad-hoc* em processos de negócio. Para efectuar as alterações *ad-hoc* é necessário um critério que garanta a correcção entre o histórico das instâncias e a nova definição dos processos. Neste trabalho define-se uma extensão ao critério de correcção já proposto por outros autores de modo a contemplar a extensão desenvolvida.

Capítulo 3

Extensão à linguagem WS-BPEL

Neste capítulo é apresentada a extensão à linguagem WS-BPEL que está dividida em duas partes: as variáveis de contexto e o novo construtor *when*. Para cada parte, é feito um levantamento das suas características e requisitos face às normas usadas e também as decisões tomadas relativamente à sintaxe. De seguida é apresentada a lógica da implementação e por último, são revelados alguns detalhes e exemplos da implementação. Como indicado na Secção 2.1.5 uma extensão pode ser implementada em tempo de modelação, através de uma transformação, ou em tempo de execução através de uma extensão ao motor de execução (caso este não suporte as extensões por omissão). Fazer uma extensão em tempo de modelação só é possível se a lógica que se pretende adicionar com a extensão for possível de obter com os construtores da linguagem, o que pode limitar a expressividade da extensão em si. Por outro lado, numa extensão em tempo de execução os novos construtores são programados directamente no motor o que impossibilita a portabilidade da extensão. Para além disto, este tipo de extensão pode implicar alterar o código fonte do motor de execução. Como tal, sempre que possível, aplicaremos uma extensão em tempo de modelação.

Por último é abordado o tema das alterações *ad-hoc* aos processos de negócio. Após definir as extensões apresentamos o critério de correcção para efectuar as alterações aos processos de negócio que as inclua. Este critério visa manter a compatibilidade entre o histórico de execução gerado por instâncias que incluem as nossas extensões e a novas definições dos processos obtidas de alterações *ad-hoc*.

3.1 Variável de Contexto

A extensão à linguagem inicia-se com a declaração do seu n*amespace* no processo. Esta declaração é feita utilizando o construtor fornecido pela linguagem para o efeito: *extensions*. Na Listagem 1 está a declaração do *namespace* da nossa extensão assim como o seu prefixo. Na declaração da extensão, o elemento "*MustUnderstand*" indica a

obrigação do motor de execução suportar a extensão com o *namespace* em causa. As componentes da nossa extensão serão então declaradas nos processos usando o mesmo *namespace*, que é feito através do uso do seu prefixo "*iotx*".

Note-se que o atributo *mustUnderstand* está com o valor "*no*" o que indica que o motor de execução não necessita de compreender esta extensão uma vez que a mesma é desenvolvida em tempo de modelação.

```
<bpel:process name=”myProcess”
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   xmlns:iotx=”http://bpel.iot.extensions”
...>
...
   <bpel:extensions>
       <bpel:extension namespace="http://bpel.iot.extensions"
mustUnderstand="no" />
   </bpel:extensions>
...
</bpel:process>
```

Listagem 1 – Declaração da extensão e seu *namespace*

Na Listagem 1 pode-se verificar o uso do prefixo "*bpel:*" nos construtores da linguagem de forma a associá-los ao *namespace* do WS-BPEL. No entanto, de forma a evitar a repetição perpetuada deste prefixo por todo o documento, declaramos o *namespace* da linguagem como sendo o padrão do documento (usando o "*xmlns=*"). Como tal, quase todos os exemplos apresentados ao longo deste capítulo contam com esta declaração de forma a tornar-se mais simples a sua compreensão.

A primeira componente da nossa extensão visa a criação do conceito de variável de contexto onde cada variável deste tipo mantém o valor actual de um sensor podendo este ser actualizado utilizando a interface da norma *WS-Notifications* para que a sua actualização seja efectuada através do paradigma publicação/subscrição ou a interface da norma *WS-ResourceProperties* de modo à actualização ser efectuada no paradigma pedido/resposta.

Como as variáveis já possuem o seu próprio construtor, optou-se por estender a linguagem acrescentando novos atributos ao construtor da variável. Estes novos

atributos são o modo com se faz a diferenciação entre uma variável normal e uma variável de contexto.

Pretende-se que a variável possa ser actualizada de duas formas: no modo publicador/subscritor sendo esta actualizada quando o sensor publica uma notificação ou através do modelo de comunicação pedido/reposta onde através da invocação de uma operação é obtido o valor actual do sensor.

No modo publicador/subscritor os novos atributos da variável representam a informação necessária para efectuar uma subscrição: o endereço do serviço web dos sensores e o tópico que se pretende subscrever. O tópico da subscrição, segundo o padrão *WS-Notifications*, é opcional e se não for incluído numa mensagem de subscrição significa que se pretende subscrever todos os tópicos disponíveis no serviço web. Os outros detalhes da subscrição (como por exemplo o endereço para onde serão enviadas as notificação) serão abordados posteriormente. Na Listagem 2 é exposta a sintaxe da variável de contexto.

```
<variables>
   <variable name="BPELVariableName"
       type="QName"?
       iotx:topic="Topic"?
       iotx:publisherEPR=”URL”?
       iotx:comunicationType=”publisher-subscriber”?>?
   </variable>
</variables>
```

Listagem 2 – Sintaxe da variável de contexto actualizada no modo publicador/subscritor

No modo pedido/resposta a variável tem que ter obrigatoriamente o tipo de comunicação declarado através do atributo *communicationType* com o valor “*request-response*”. Adicionalmente tem que ter o atributo *refreshTime* declarado. A sua sintaxe é exposta na Listagem 3.

Na Listagem 4 está a declaração de uma variável de contexto chamada “*tempVar*” cujo valor será actualizado pelo sensor que representa o tópico “Temperatura” no endereço indicado pelo atributo “*publisherEPR*”. Como esta não possui o atributo *communicationType* esta variável usará o tipo de actualização por omissão que é o modo publicador/subscritor. Este tipo de actualização também pode ser obtido usando o atributo *communicationType* com o valor defino para “*publisher-subscriber*”.

```
<variables>
   <variable name="BPELVariableName"
       type="QName"?
       iotx:ResourceProperty="ResourceName"?
       iotx:SourceEPR=”URL”?
       iotx:comunicationType=”request-response”?
       iotx:refreshTime=”Time”? >?
   </variable>
</variables>
```

Listagem 3 - Sintaxe da variável de contexto actualizada no modo pedido/resposta

```
<bpel:variables>
       <bpel:variable name="tempVar"
                      type="xsd:int"
                      iotx:topic="Temperatura"
       iotx:publisherEPR="http://192.168.1.52:8081/axis2/services/SensorSer
vice"/>
       ...
</bpel:variables>
```

Listagem 4 – Declaração de uma variável de contexto para actualização no modelo publicador/subscritor

Já uma variável que é actualizada segundo o paradigma pedido/resposta, a sua declaração contém os mesmos atributos que a declaração da Listagem 4 mas neste caso tem que ter obrigatoriamente dois atributos adicionais: o tipo de comunicação através do atributo *comunicationType* e definida com o valor “*request-response*” e o segundo atributo, *refreshTime*, que indica a periocidade com que é actualizada a variável de contexto. A Listagem 5 exemplifica a declaração de uma variável de contexto actualizada no modo pedido/resposta e que é actualizada em intervalos de 5 minutos de forma síncrona.

Uma vez definida a extensão à linguagem passamos à sua implementação. Como no nosso caso o objectivo é contactar sensores que partilham os seus valores recorrendo a serviços *web* é possível efectuar esta comunicação recorrendo apenas a uma transformação, evitando alterar o motor de execução.

```
<bpel:variables>
    <bpel:variable name="humiVar"
    type="xsd:int"
    iotx:ResourceProperty="Humidade"
    iotx:sourceEPR="http://192.168.1.52:8081/axis2/services/SensorService"
    iotx:comunicationType=”request-response”
    refreshTime=”PT5M0S”/>
</bpel:variables>
```

Listagem 5 – Declaração de uma variável de contexto para actualização no modelo pedido/resposta periódico

A ideia principal da transformação da variável de contexto é adicionar ao processo elementos da linguagem (como actividades, variáveis, etc.) que permitam contactar o serviço *web* dos sensores que são as fontes de contexto. No caso em que a actualização da variável de contexto é efectuada no paradigma publicador/subscritor as actividades adicionadas devem tratar da subscrição e da actualização das variáveis sem alterar o fluxo do modelador. Já no segundo modo de actualização, pedido/resposta, as actividades adicionadas devem actualizar a variável com a periocidade definida no seu construtor.

Como existe a invocação de operações de serviços externos, temos de adicionar nesta transformação documentos WSDL que descrevam os serviços que vão ser invocados. Como tal, esta transformação, para além da edição do ficheiro WS-BPEL, inclui a criação de ficheiros WSDL.

3.1.1 Edição do ficheiro BPEL

Na Figura 2 podemos ver a definição do processo de negócio usada para o desenvolvimento da transformação. A definição é bastante simples e contém uma sequência com uma actividade *Receive* chamada *Start* responsável por iniciar a execução da instância quando é invocada a sua operação, duas actividades *Assign* e uma actividade *Reply* que serve para responder à operação usada no *Start*. Adicionalmente foram declaradas duas variáveis de contexto, a primeira a ser actualizada no modo publicador/subscritor e a segunda no modo pedido/reposta. Por fim, adicionámos um construtor *When* à definição que será abordado na secção seguinte.

Para simplificar a exposição da solução posteriormente, as actividades adicionadas à definição pela transformação das variáveis de contexto e pelo construtor *When* são separadas sendo a junção de todas elas o resultado da transformação.

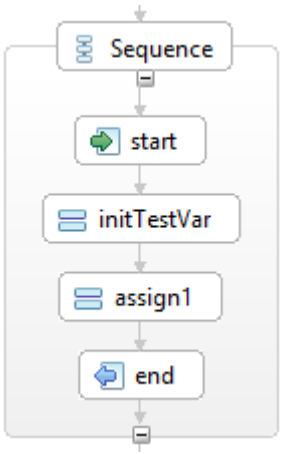

Figura 2 – Definição do processo de negócio antes da transformação

A transformação no documento WS-BPEL ocorre quando são detectadas declarações de variáveis de contexto dentro de um processo. Quando isto acontece, inicialmente são copiadas todas as variáveis, *PartnerLinks*, *CorrelationSets* e outros elementos que possam estar declarados pelo programador. Às variáveis de contexto que são encontradas são-lhes removidos os atributos de extensão passando estas a ser variáveis normais (sem atributos de extensão) do tipo *xsd*:*AnyURI*. De seguida são adicionados *PartnerLinks* para associar às operações de comunicação e os *imports* dos WSDL onde se encontram as interfaces das normas utilizadas consoante o modo de comunicação seleccionado. Estes *imports* são essenciais para o processo conseguir trabalhar com as operações e mensagens trocadas nas operações fornecidas pelas normas.

Considerando a definição de processo da Figura 2 e a definição de variável de contexto actualizada no modo publicador/subscritor da Listagem 4, esta definição é transformada para a definição do processo que se pode ver na Figura 3 e explicado de seguida.

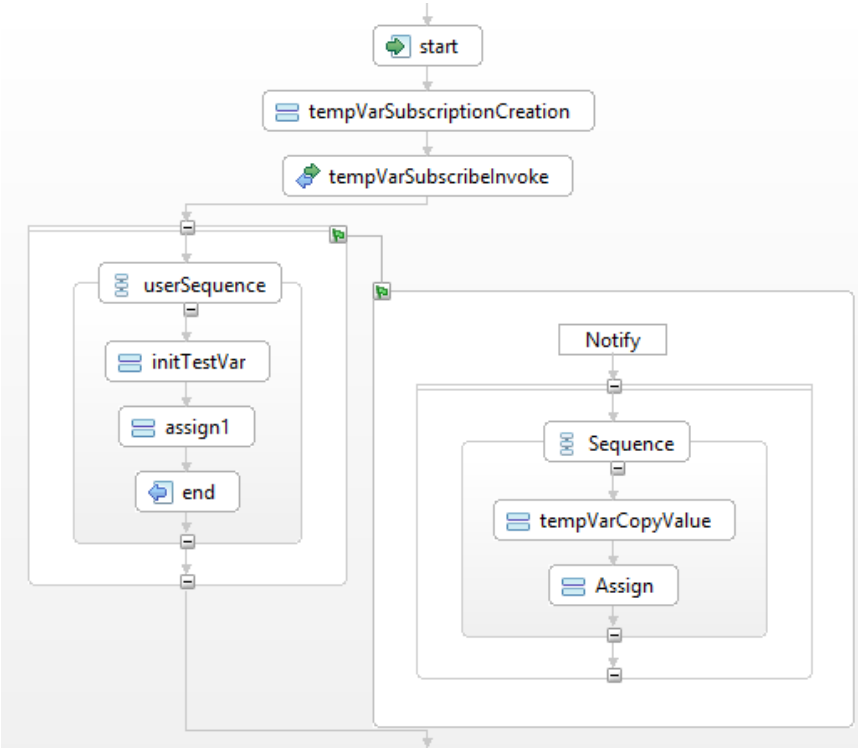

Figura 3 – Actividades adicionadas na declaração da variável de contexto no modo publicador-subscritor

A *Sequence* principal do processo é mudada de forma que a primeira actividade (a actividade *Receive* chamada "*Start*") continue a ser a actividade inicial. A seguir a esse *Receive* são inseridas actividades que permitem a subscrição de notificações no publicador. Esta operação é efectuada adicionando uma actividade *Invoke*. Esta actividade *Invoke* contacta o EPR do publicador indicado na definição da variável de contexto. Como a operação de subscrição é de dois sentidos, declara-se no processo duas variáveis, a de *input* e a de *output*. A mensagem enviada para o subscritor é inicializada por uma actividade de *Assign*, declarada antes do *Invoke*, e é formatada de acordo com a norma *WS-Notifications*. Nesta inicialização o tópico declarado na variável de contexto e o *EPR* para onde devem ser enviadas as notificações (que é gerado concatenando o nome do processo com o nome da variável de contexto de forma a ser único) são inseridos na mensagem de subscrição. Já a variável de *output* é inicializada na resposta. Como é na resposta de subscrição que se inicializa o *Correlation*, é declarada a inicialização no sentido da resposta. A Listagem 6 e a Listagem 7 mostram a mensagem de subscrição usada e o código da actividade *Invoke*, respectivamente.

```
<wsnt:Subscribe ...>
    <wsnt:ConsumerReference>
        <wsa:Address ... >
            http://192.168.1.71:8080/ode/processes/myProcesstempVar
        </wsa:Address>
    </wsnt:ConsumerReference>
    <wsnt:Filter>
        <wsnt:TopicExpression ... > Temperatura </wsnt:TopicExpression>
    </wsnt:Filter>
</wsnt:Subscribe>
```

Listagem 6 – Mensagem de subscrição segundo a norma *WS-Notifications*

```
<bpel:invoke name="Invoke"
      partnerLink="pubSubPartnerLink"
      operation="Subscribe"
      portType="wsntw:NotificationProducer"
      inputVariable="tempVarsubscribeRequest"
      outputVariable="tempVarsubscribeResponse" >
    <bpel:correlations>
    <bpel:correlation set="tempVarCorrelationSet"
             initiate="yes" pattern="response" />
    </bpel:correlations>
</bpel:invoke>
```

Listagem 7 – Actividade *Invoke* que contacta o serviço dos sensores

A actividade de *Invoke* tem associado o *PartnerLink* adicionado para que se possa usar o *PartnerRole* na invocação.

Para o processo receber as notificações tem de ser adicionada uma actividade de comunicação de *input*. Como as notificações podem ocorrer a qualquer momento, tem que se usar uma actividade de comunicação assíncrona como a onEvent. Esta actividade vai utilizar o mesmo *PartnerLink* da actividade da operação anterior mas neste caso é usado o *MyRole*. Associada a esta actividade está a declaração da variável de mensagem de notificação onde vão ser guardadas as actualizações recebidas. Neste *onEvent* está também associado o uso do *Correlation*. Um exemplo desta actividade é mostrado na Listagem 8.

```
<bpel:eventHandlers>
     <bpel:onEvent partnerLink="tempVarPubSubPartnerLink"
                   operation="Notify"
                   portType="wsntw:NotificationConsumer"
                   variable="tempVarNotificationMsg">
              ...
      </bpel:onEvent>
       <bpel:correlations>
              <bpel:correlation set="tempVarCorrelationSet"
                                initiate="no">
              </bpel:correlation>
       </bpel:correlations>
</bpel:eventHandlers>
```

Listagem 8 – *EventHandler onEvent* que recebe as notificações

Por fim, dentro deste *onEvent* (no local apontado com reticências na Listagem 8) é adicionada uma actividade *Assign*, a qual copia o elemento da mensagem de notificação que contém o novo valor para a variável de contexto. Desta forma o valor actual lido do sensor fica guardado na variável. A Listagem 9 mostra o código para este *Assign*.

A recepção de notificações só faz sentido enquanto a sequência original de actividades declarada pelo modelador não executar por completo. Como tal, quando esta sequência de actividades é copiada na transformação da definição é colocada num Scope a seguir à actividade Invoke onde é efectuada a subscrição. Desta forma o *EventHandler onEvent* pode ser associada a este *Scope* e só está activa enquanto a sequência do modelador não executar totalmente.

```
<bpel:assign validate="no" name="updateVar">
    <bpel:copy>
        <bpel:from part="Notify" variable="NotificationMsg">
           <bpel:query ...> wsnt:NotificationMessage/wsnt:Message
           </bpel:query>
        </bpel:from>
        <bpel:to variable="tempVar" />
    </bpel:copy>
</bpel:assign>
```

Listagem 9 – Actividade de *Assign* que actualiza a variável

Tendo em conta que todas as instâncias do mesmo processo usam o mesmo EPR para receber as notificações, temos de usar o *Correlation* de forma a garantir a entrega das notificações às instâncias corretas. O *Correlation* é definido na definição do processo e num documento WSDL. De seguida é explicado como usamos o *Correlation* no processo para garantir que as notificações publicadas são entregues ao subscritor correto.

Como cada variável de contexto corresponde a uma subscrição diferente, é criado um *correlationSet* por variável. O nome de cada *correlationSet* é a concatenação do nome da variável que o originou com "CorrelationSet" e todos eles usam a propriedade definida no WSDL adicionado pela transformação ao processo. Essa propriedade é a responsável por definir os elementos das mensagens que identificam as conversações e são explicadas na secção seguinte. A Listagem 10 exemplifica o *correlationSet* criado pela variável "*tempVar*". De se notar que o prefixo da propriedade ("aux:") associa a propriedade em causa ao namespace do documento WSDL (que é importado) onde se encontra a sua definição.

```
<bpel:correlationSets xmlns:aux="http://lasige/bpel/subscriberArtifacts">
      <bpel:correlationSet name="tempVarCorrelationSet"
            properties="aux:SubscriptionReference "/>
</bpel:correlationSets>
```

Listagem 10 – *CorrelationSet* gerado por uma variável de contexto

Já para o outro modelo de comunicação, pedido/reposta, a transformação é bastante mais simples uma vez que não envolve *Correlation*. Para este caso a transformação gera uma sequência de actividades como as apresentadas na Figura 4. Como se pretende que a sequência adicionada execute em paralelo com a sequência de actividades do modelador de forma a obter as actualizações sem interferir com o fluxo principal, é adicionado uma actividade *Flow*.

Note-se que, para comunicar no modo pedido/resposta existem várias interfaces uma vez que muitos serviços da *web* utilizam esta metodologia. No nosso caso em concreto consideramos que os sensores implementam operações disponibilizadas pela interface da norma *WS-ResourceProperties* [21]. Nesta norma, uma fonte de contexto pode disponibilizar vários valores através de várias *Resource Properties* e, considerando

que estas são conhecidas à partida pelo modelador, basta-nos obter o valor destas com a operação *GetResourceProperty*.

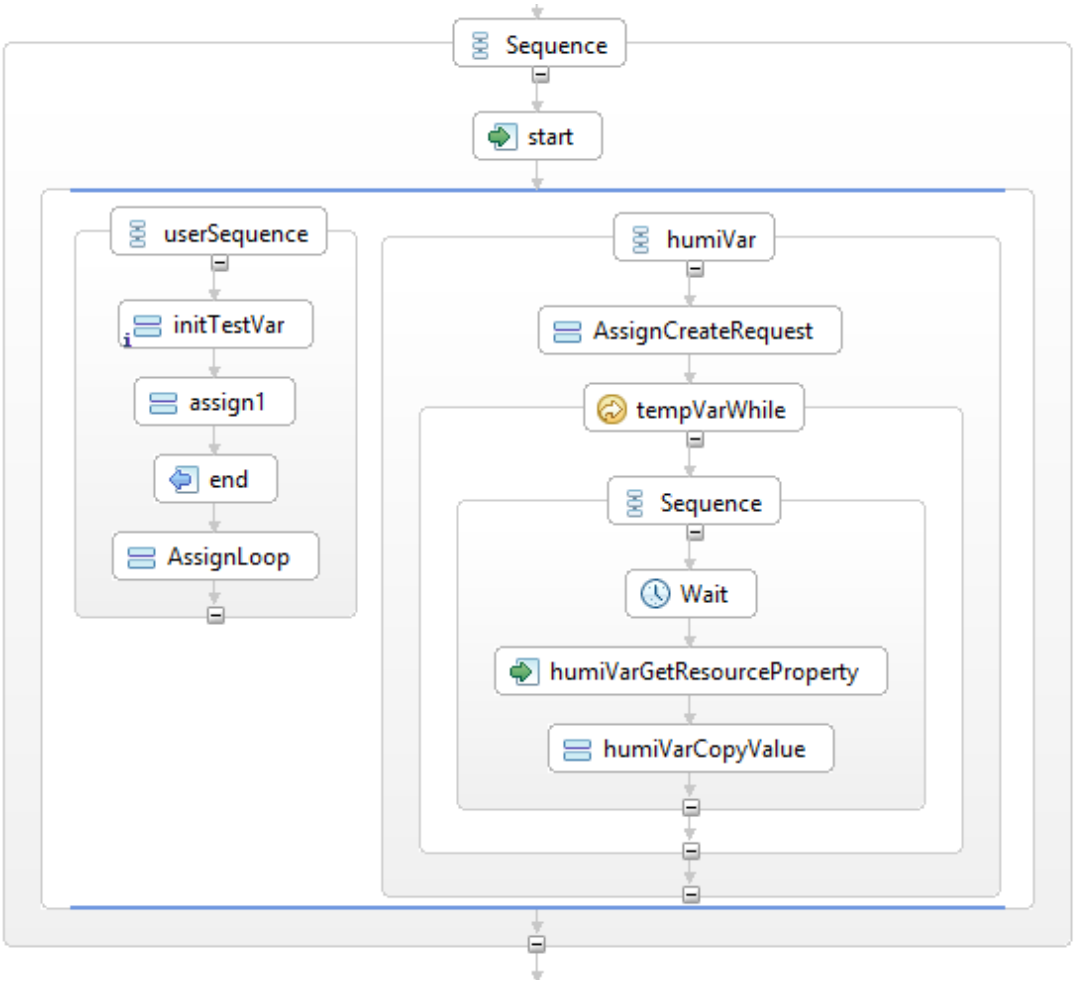

Figura 4 - Actividades adicionadas na declaração da variável de contexto no modo pedido/resposta

Na sequência adicionada inicialmente encontra-se uma actividade *Assign* que inicializa a mensagem enviada ao sensor. A mensagem criada pelo *Assign* pode ser vista na Listagem 11.

```
<wsrf-rp:GetResourceProperty>
       Humidade
</wsrf-rp: GetResourceProperty>
```

Listagem 11 – Mensagem enviada ao sensor no modo pedido/resposta

A seguir ao *Assign* vem um ciclo *While* constituído por uma actividade *Wait* que vai adormecer o processo pelo tempo definido na variável de contexto. Esta actividade pode ser vista na Listagem 12.

```
<bpel:wait name="Wait">
       <bpel:for>'PT5M1S'</bpel:for>
</bpel:wait>
```

Listagem 12 – Actividade *Wait* que adormece o processo

Quando termina a actividade *Wait* é executada a actividade *Invoke* que contacta o publicador para obter a última notificação gerada. Nesta actividade estão envolvidas duas mensagens. A primeira mensagem é do tipo *GetResourcePropertyRequest*, é inicializada pelo *Assign* e identifica qual o recurso que se pretende obter. A segunda mensagem é do tipo *GetResourcePropertyResponse* e é enviada como resposta à invocação da operação. A Listagem 13 exemplifica a invocação da operação *GetResourceProperty*.

```
<bpel:invoke name="tempVargetSensorValueInvoke"
       partnerLink="tempVarResourcePartnerLink"
       operation="GetResourceProperty"
       portType="wsrf:GetResourceProperty"
       inputVariable="tempVarGetResourceRequest"
       outputVariable="tempVarGetResourceMsgResponse">
</bpel:invoke>
```

Listagem 13 – Invocação da operação GetCurrentMessage

Por fim, a actividade *While* tem um *Assign* semelhante ao da versão do publicador/subscritor que extrai o valor da mensagem e actualiza a variável com o nome declarado na variável de contexto, só que neste caso adaptado para a norma utilizada.

Como indicado anteriormente, esta sequência de actividades é colocada numa actividade *Flow* para correr em paralelo com a sequência do utilizador. De forma a terminar o ciclo *While* onde é recepcionado o valor da fonte de contexto foi adicionado um *Assign* no fim da sequência de actividades do modelador que altera o valor de uma variável booleana usada nesse ciclo. Desta forma garante-se que o ciclo *While* termina quando a sequência de actividades definidas pelo modelador terminam. Note-se que dada esta implementação pode ocorrer a situação da sequência do modelador terminar e

o processo demorar ainda parte do tempo definido na actividade *Wait* para terminar totalmente.

3.1.2 Adição de ficheiros WSDL

Como visto anteriormente, para cada variável de contexto declarada, seja no modo publicador/subscritor ou pedido/resposta, são referidos serviços *web*. No caso das variáveis actualizadas no modo publicador/subscritor, são referidos dois serviços: o serviço do publicador onde vai ser invocada a operação de subscrição e o serviço para onde vão ser enviadas as notificações. Já no caso das variáveis actualizadas no modo pedido/resposta existe apenas o serviço onde é obtido o valor da variável. O endereço do serviço do sensor é obtido directamente da definição da variável de contexto. Estes serviços têm de ser declarados num ficheiro WSDL. Como o ficheiro WSDL do utilizador não deve ser alterado, decidiu-se criar um ficheiro adicional e o processo WS-BPEL transformado importa-o.

No caso da actualização por subscrição, o endereço do serviço para onde as notificações são enviadas é criado através da concatenação do nome do processo com o nome da variável de contexto de forma a ser único por processo. As operações de cada serviço são directamente importadas das interfaces de publicador (*NotificationProducer*) e subscritor (*NotificationConsumer*), disponibilizadas pelos WSDL da norma *WS-Notifications* (que também devem ser acrescentados na transformação).

Para além dos serviços, é neste documento que é declarada a propriedade de *Correlation* usada na transformação WS-BPEL. Esta propriedade é constituída por duas regras específicas para um tipo de mensagem. As regras indicam, para cada mensagem, o campo *ReferenceParameters* do elemento *SubscriptionReference* como sendo para usar na correlação. Já o tipo de dados a ser utilizado tem de ser o mesmo que o campo *ReferenceParameters*, ou seja, *anyURI*. Na Listagem 14 apresentamos um exemplo da propriedade de correlação criada assim como a regra relativa à mensagem de resposta de subscrição. Nesse exemplo está também o *import* do ficheiro WSDL da norma *WS-Notifications*.

Desta forma, quando uma instância de um processo efectua uma subscrição, o motor utiliza a primeira regra com a resposta da subscrição para inicializar o *CorrelationSet* dessa instância. Posteriormente, quando uma mensagem de notificação chegar ao *EPR* do serviço criado para receber as notificações, o motor utiliza a segunda regra e a mensagem de notificação para verificar qual a instância correta, garantindo que a mensagem é entregue sempre à instância correta.

No segundo modo de actualização da variável de contexto, pedido/resposta, o ficheiro da norma *WS-ResourceProperties* também é importado. Para este caso não são necessárias regras de *Correlation* e como tal estas não são criadas.

```
<wsdl:definitions ...>
...
  <wsdl:import location="wsdl/WS-BaseNotification-1_3.wsdl"
               namespace="http://docs.oasis-open.org/wsn/bw-2"/>
  <vprop:property name="SubscriptionReference" type="xsd:anyURI"/>
  <vprop:propertyAlias messageType="wsntw:SubscribeResponse"
    part="SubscribeResponse" propertyName="tns:SubscriptionReference">
    <vprop:query> /wsnt:SubscriptionReference/wsa:ReferenceParameters
    </vprop:query>
  </vprop:propertyAlias>
</wsdl:definitions>
```

Listagem 14 – Regra de *Correlation* da resposta de subscrição

3.2 Construtor When

A segunda parte da extensão é o construtor *When*. Este construtor é declarado ao mesmo nível que as variáveis, *PartnerLinks*, etc. De se notar que o *When* não é uma actividade pois não faz sentido dentro da sequência, mas sim um construtor semelhante ao *FaultHandler* ou *TerminationHandler*. Este construtor é declarado contendo uma condição e uma actividade, que é executada quando a condição se verifica. Na Listagem 15 está exposta a sintaxe do construtor *When*.

```
<iotx:when name="whenName">
    <bpel:condition>bool-expr</bpel:condition>?
        Activity
</iotx:when>
```

Listagem 15 – Sintaxe do construtor *When*

A Listagem 16 mostra a declaração de um *When* que executa uma sequência com duas actividades vazias quando o valor da variável "*tempVar*" for superior a 35. A condição em causa contém ainda o prefixo para salientar o facto de que a condição pertence ao domínio da linguagem, ie, é uma condição igual à condição de uma actividade *If*.

Assim como nas variáveis de contexto, a extensão do elemento *When* é baseada numa transformação que adiciona lógica ao processo que corresponda ao pretendido pelo novo construtor, ou seja, um novo ramo no fluxo do processo que aguarda pela veracidade da condição no *When* para depois executar a actividade associada.

```
<process name="myProcess">
...
    <iotx:when name="testeWhen">
        <bpel:condition>$tempVar > 35</bpel:condition>
        <sequence name="teste">
            <empty name="empty1"> </empty>
            <empty name="empty2"> </empty>
        </sequence>
    </iotx:when>
...
</process>
```

Listagem 16 – Declaração do Construtor When

Para efectuar esta espera foram analisadas três hipóteses: (1) espera activa; (2) criar uma actividade de extensão usando o mecanismo *extensionActivity* e (3) adicionar um serviço *web* que fique responsável por avaliar a condição.

Efectuar uma espera activa (por exemplo, um ciclo *While* que em determinados intervalos de tempo verifica a condição) não é uma solução viável porque a condição pode nunca ser verdade e ocupa recursos de computação. Já o mecanismo *extensionActivity* face ao que se pretende é limitado porque o motor de execução delega um tempo de execução para cada actividade, como tal, a *extensionActivity* não poderia bloquear por um tempo indeterminado. Posto isto, optou-se pela terceira hipótese. Com esta solução temos a vantagem de poder usar os *Listeners* que o ODE fornece para detectar alterações às variáveis e avaliar a condição apenas quando as variáveis estiverem inicializadas. Os *listeners* são uma vantagem nesta abordagem porque permitem avaliar as condições apenas quando as variáveis são alteradas evitando uma abordagem de espera activa. Já a definição do processo será transformada adicionando

apenas lógica que o fará contactar este serviço local para registar a sua condição e uma actividade de comunicação de *input* assíncrona para que o serviço mais tarde o contacte de volta para então o processo executar a actividade.

Esta opção, apesar de ser em tempo de modelação, está dependente de um serviço auxiliar que, por sua vez, está dependente dos *Listeners* do motor de execução. Isto faz com que esta extensão só seja portável para outros servidores de ODE. Como esta solução não é portável apresentamos também uma alternativa que recorre igualmente a um serviço externo mas que não depende de nenhuma componente do ODE. Ainda assim o serviço necessita de identificar univocamente cada processo interveniente o que obriga à transformação da definição do processo incluir funcionalidades específicas do motor como a variável de sistema que contém o identificador da instância. Esta variável também existe em outros motores de execução pelo que a portabilidade da solução para outros motores passa por alterar apenas o nome da variável de sistema na transformação.

Uma vez seleccionado o modo como se vai implementar esta extensão vamos então analisar como será feita a transformação e o serviço *web* auxiliar gestor das condições *When*. Assim como no exemplo anterior, a transformação envolve a invocação de serviços *web* e como tal deve acrescentar-se o ficheiro WSDL que descreve esse serviço. Começaremos a análise pela transformação do processo de negócio, passando de seguida ao gestor das condições *When*.

3.2.1 Edição do ficheiro BPEL

Como indicado anteriormente, o documento escrito em WS-BPEL deve ser alterado para adicionar lógica nova ao processo. Na transformação, quando é detectado o uso de *When*, a sequência principal do processo é alterada da mesma forma que para as variáveis de contexto. A Figura 5 ilustra um exemplo do resultado desta transformação que será explicado de seguida.

Tal como nas variáveis de contexto são adicionadas actividades que representam o *When*. Estas actividades representam a lógica de negócio pretendida para o construtor *When*. Já o construtor *When* que foi declarado, este é removido da definição do processo para que na altura de compilação pelo motor de execução não existam elementos de extensão. Para além das actividades a serem acrescentadas são também declaradas duas variáveis de mensagem por construtor *When*, que são usadas na comunicação com o serviço web, o *PartnerLink* que fornece os *Roles* dos intervenientes e o *CorrelationSet*.

As actividades adicionadas para cada *When* declarado na definição do processo são bastante simples. Logo ao início está um *Assign* que inicializa a mensagem de registo do

When para enviar ao serviço. Essa mensagem, como descrito no ponto anterior, é composta pela condição, o EPR do serviço criado para receber a mensagem de desbloqueio (descrito posteriormente) e o identificador de instância.

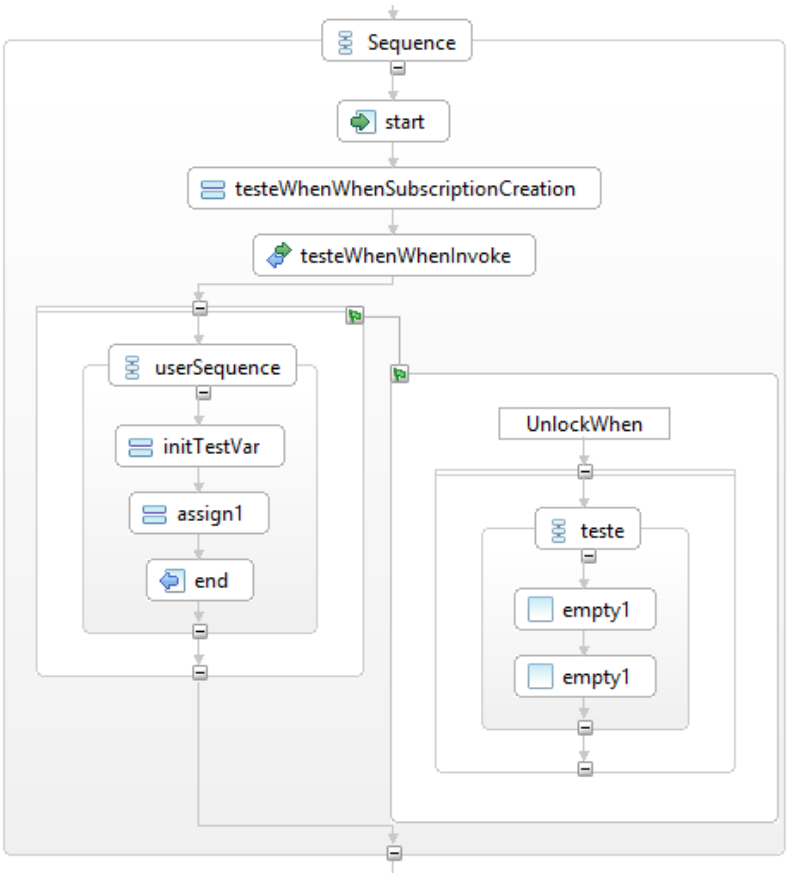

Figura 5 - Actividades adicionadas na declaração do construtor *When*

Para que a instância tenha informação sobre o seu *InstanceId* é utilizado uma variável de sistema existente em cada instância e disponibilizada pelo ODE: "*$ode:pid*". De forma aos nomes das variáveis de mensagens serem únicos para cada *When* declarado, estes são obtidos concatenando o nome do *When* a que pertence e "*WhenRequest*". Na Listagem 17 é apresentada a actividade *Assign* que inicializa a mensagem, podendo ver-se a negrito a função nativa do ODE que devolve o *InstanceId*.

A seguir surge uma actividade de *Invoke* que contacta o serviço invocando a operação "*RegisterWhen*" utilizando a variável inicializada pelo *Assign*. Para que as mensagens que o serviço auxiliar envia sejam entregues a cada instância tem de se usar correlação. Assim neste *Invoke* é utilizado um *CorrelationSet* que é inicializado usando

o campo *InstanceId* da mensagem a ser enviada. Um exemplo deste *Invoke* pode ser visto na Listagem 18.

```
<assign validate="no" name="WhenSubscriptionCreation">
  <copy>
    <from>
      <literal xml:space="preserve">
         <when:RegisterWhenRequestElement>
            <when:WhenProcessReference>
               <wsa:Address ...>
           http://localhost:8080/ode/processes/SubscribertesteWhen
               </wsa:Address>
            </when:WhenProcessReference>
            <when:Condition>$tempVar &gt; 35</when:Condition>
            <when:InstanceId> -1 </when:InstanceId>
         </when:RegisterWhenRequestElement>
      </literal>
    </from>
    <to variable="testeWhenWhenRequest" part="registerWhenRequest"/>
   </copy>
   <copy>
      <from> $ode:pid </from>
      <to variable="testeWhenWhenRequest" part="registerWhenRequest">
         <query ...> when:InstanceId </query>
      </to>
  </copy>
</assign>
```

Listagem 17 – Actividade *Assign* que inicializa a mensagem de registo da condição

Logo a seguir ao *Invoke* está o *EventHandler onEvent*, o qual está associado à operação "*UnlockWhen*".

Este *onEvent* também usa correlação sendo esta declarada para permitir as mensagens serem entregues à instância correcta. Esta utiliza a regra que define o campo "*InstanceId*" como identificador. Um exemplo deste *onEvent* é apresentado na Listagem 19.

```
<bpel:invoke name="registerWhenInvoke"
       partnerLink="testeWhenWhenPartnerLink"
       operation="RegisterWhen"
       portType="whenw:WhenServiceManagerPortType"
       inputVariable="testeWhenWhenRequest">
       <bpel:correlations>
              <bpel:correlation set="testeWhenWhenCorrelationSet"
              initiate="yes"/>
       </bpel:correlations>
</bpel:invoke>
```

Listagem 18 – Actividade *Invoke* que regista a condição do construtor *When* no serviço auxiliar

```
<bpel:eventHandlers>
       <bpel:onEvent partnerLink="testeWhenWhenPartnerLink"
                     operation="UnlockWhen"
                     portType="whenw:WhenProcessPortType"
                     variable="testeWhenUnlockRequest">
       ...
       </bpel:onEvent>
</bpel:eventHandlers>
```

Listagem 19 – *EventHandler onEvent* que recebe a mensagem de desbloqueio do serviço auxiliar

Por fim, dentro do *onEvent* vêm as actividades que foram declaradas pelo modelador dentro do construtor *When* para que sejam executadas quando for recebida a mensagem de desbloqueio.

À semelhança das variáveis de contexto, as actividades do modelador são também colocadas dentro de um *Scope* a seguir à invocação do serviço auxiliar para registo da condição e é nesse *Scope* que está associado o *onEvent* declarado. Se um processo tiver uma variável de contexto e um *When* apenas um *Scope* será declarado.

Na segunda solução implementada o funcionamento do serviço externo é diferente do actual porque este não depende dos *listeners* do motor de execução para monitorizar as alterações às variáveis dos processos e como tal tem que obter essa informação de outra forma.

À semelhança da solução anterior são adicionadas actividades à sequência que invocam o serviço gestor de condições fornecendo a condição do *When* e o endereço do

onEvent que aguarda a invocação da sua operação quando a condição é verdade para executar as actividades declaradas no *When*.

No entanto, como não temos os *listeners* para monitorizar alterações aos valores das variáveis pelo serviço auxiliar temos que efectuá-la de outra forma. Para tal é declarada uma nova operação no serviço externo, a operação *UpdateVariab*le para que sempre que se verificar uma actualização de uma variável no processo de negócio é invocada a operação *UpdateVariable* partilhando com o serviço auxiliar o novo valor da variável alterada mantendo-o actualizado.

Na transformação, a definição é alterada de forma a adicionar um par de actividades de *Assign* e *Invoke* para cada actividade declarada que altere o valor de qualquer variável e alguns exemplos de são as actividades *Assign* numa atribuição directa de valor, *Invoke* na recepção de uma mensagem no *output* de uma operação e *Receive* na recepção de uma mensagem. Os exemplos das actividades adicionadas são baseados na actividade *Assign* mostrada na Listagem 9.

A actividade *Assign* adicionada cria uma mensagem para ser enviada ao serviço auxiliar na actividade *Invoke* seguinte. Esta mensagem é composta pelo nome da variável que foi alterada, *tempVar*, o valor actual desta variável e o identificador da instância do processo. Nas Listagem 20 e Listagem 21 pode-se ver um exemplo das atribuições de valores do *Assign* adicionado onde é apresentado a criação da mensagem e a sua estrutura e o preenchimento dos seus campos, respectivamente.

```
<assign validate="no" name="WhenSubscriptionCreation">
  <copy>
    <from>
      <bpel:literal xml:space="preserve">
        <when:UpdateVarWhenRequestElement>
             <when:InstanceId> -1 </when:InstanceId>
             <when:VariableName> varName </when:VariableName>
             <when:VariableValue> -1 </when:VariableValue>
        </when:UpdateVarWhenRequestElement>
       </bpel:literal>
    </from>
    <to variable="varUpdateWhenRequest" part="updateVarRequest">
  </copy>
  ...
</assign>
```

Listagem 20 - Actividade *Assign* que gera a mensagem de actualização de variável para enviar ao serviço externo When 1/2

```
<assign validate="no" name="WhenSubscriptionCreation">
  ...
  <copy>
    <from> $ode:pid </from>
    <to variable="varUpdateWhenRequest" part="updateVarRequest">
       <query ...> when:InstanceId </query>
    </to>
  </copy>
  <copy>
    <from> 'tempVar' </from>
    <to variable="varUpdateWhenRequest" part="updateVarRequest">
       <query ...> when:VariableName </query>
    </to>
  </copy>
  <copy>
    <from> $tempVar </from>
    <to variable="varUpdateWhenRequest" part="updateVarRequest">
       <query ...> when:VariableValue </query>
    </to>
  </copy>
</assign>
```

Listagem 21 – Actividade *Assign* que gera a mensagem de actualização de variável para enviar ao serviço externo When 2/2

A actividade de *Invoke* adicionada tem como objectivo enviar o novo valor da variável após a sua alteração para o serviço externo. Está associada ao mesmo *PartnerLink* que a actividade que regista a condição *When*, e após a criação da mensagem, esta é enviada ao serviço auxiliar usando esta actividade de *Invoke* com a operação *UpdateVariab*le. Um exemplo desta actividade pode ser visto na Listagem 22.

Ao se adicionar estas duas actividades sempre que se efectua uma atribuição no processo consegue-se garantir que o serviço auxiliar mantém os valores actuais das variáveis podendo efectuar a avaliação das condições.

```
<bpel:invoke name="updateVarInvoke"
        partnerLink="testeWhenWhenPartnerLink"
        operation="UpdateVarWhen"
        portType="whenw:WhenServiceManagerPortType"
        inputVariable="varUpdateWhenRequest">
</bpel:invoke>
```

Listagem 22 – Actividade *Invoke* que envia a mensagem de actualização de variável ao serviço auxiliar

3.2.2 Adição de ficheiros WSDL

Na transformação são também adicionados ficheiros WSDL que descrevem o serviço auxiliar *When*. Nestes ficheiros está a descrição das duas interfaces: a utilizada pelo serviço *web* para que o processo o possa contactar e a que o processo WS-BPEL implementa para que possa ser contactado.

A interface do serviço *web* é composta por duas operações já apresentadas anteriormente: *RegisterWhen* e *UpdateVar*. O seu endereço é atribuído pela extensão. Já a interface do processo é composta apenas pela operação *UnlockWhen* e o endereço do serviço que os processos usam é obtido pela concatenação do nome do processo com o nome do *When* (atributo *name* do construtor) de forma a serem únicos para cada definição diferente. De se notar que o facto de se usar o nome do *When* para gerar o endereço faz com que este atributo seja obrigatório.

Para além disto, este ficheiro contém as mensagens trocadas nas operações, o *PartnerLinkType* e as regras de correlação utilizadas pelas actividades *Invoke* e *Receive*. A propriedade e regras de correlação são bastante simples: apenas seleccionam o campo *InstanceId* das mensagens trocadas (como se pode ver na Listagem 23).

Com isto o processo consegue ter todas as informações necessárias para contactar, registar o seu *When* e posteriormente receber a mensagem de desbloqueio do *Receive* para que as actividades associadas executem.

```
<wsdl:definitions ...>
...
  <vprop:property name="whenProp" type="xsd:long"/>
  <vprop:propertyAlias messageType="whenw:RegisterWhenRequestMessage"
                       part="registerWhenRequest"
                       propertyName="iotx:whenProp">
     <vprop:query>/when:InstanceId</vprop:query>
  </vprop:propertyAlias>
  <vprop:propertyAlias messageType="whenw:unlockWhenRequest"
                       part="unlockWhenRequest"
                       propertyName="iotx:whenProp">
     <vprop:query>/when:InstanceId</vprop:query>
  </vprop:propertyAlias>
</wsdl:definitions>
```

Listagem 23 – Regras de correlação usadas na comunicação com o serviço *web* auxiliar

3.2.3 Serviço Web gestor das condições When

Este serviço tem como função receber as condições declaradas nos construtores *When* dos processos, monitorizar alterações às variáveis desses processos e de os notificar quando as suas condições forem verdade para que, por fim, o processo execute as actividades do *When*.

Para fazer isto correctamente temos de ter em conta o seguinte pormenor, o *Listener* de eventos do ODE apanha todos os eventos de todas as instâncias em execução. Como tal podemos receber uma notificação de alteração de uma variável chamada "*tempVar*" e ter várias definições de processos diferentes em execução com uma variável com esse nome. Para além disso quando o processo for notificado temos de considerar a hipótese de estarem em execução várias instâncias desse mesmo processo. Desta forma, quando um processo regista a condição do seu *When* este terá também de se identificar com o seu identificador de instância que é único entre todas as instâncias de todos os processos.

Tendo isto em conta foram criadas duas interfaces: a do serviço auxiliar que disponibiliza as operações *RegisterWhen* e *UpdateVar* e a interface dos processos que disponibiliza a operação *UnlockWhen*. Na Listagem 24 está o esquema (do inglês *schema*) do elemento trocado na operação de "*RegisterWhen*" e como se pode ver é formado por: 1) o *EPR* onde se vai invocar a operação "UnlockWhen"; 2) a condição do *When* e 3) o id da instância.

```
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" ...>

   <xsd:element name="RegisterWhenRequestElement">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="WhenProcessReference"
                         type="wsa:EndpointReferenceType"
                         minOccurs="1"
                         maxOccurs="1"/>
            <xsd:element name="Condition" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="InstanceId" type="xsd:long" minOccurs="1"
maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

...

</xsd:schema>
```

Listagem 24 – *Schema* do elemento trocado no registo da condição do When

Na Listagem 25 está o esquema do elemento trocado na operação de "*UpdateVar*" e como se pode ver é formado por: 1) o nome da variável a ser actualizada; 2) o id da instância e 3) o valor da variável.

```
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" ...>

   <xsd:element name="UpdateVarRequestElement">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="VariableName" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
            <xsd:element name="InstanceId" type="xsd:long" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="VariableValue" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

...

</xsd:schema>
```

Listagem 25 - *Schema* do elemento trocado na operação de actualização das variáveis

O funcionamento do serviço será então o seguinte: quando recebe uma invocação da operação "*RegisterWhen*", o serviço passa as informações na mensagem recebida para uma classe responsável por gerir e avaliar as condições. Esta classe armazena a informação do registo numa estrutura de dados (que está organizada pelo atributo "*InstanceId*") e verifica quantas e quais as variáveis usadas. Isto permitirá em tempo de execução determinar se estamos em condições de avaliar a expressão, isto é, se todas as variáveis estão inicializadas. Quando ocorrer uma alteração de uma variável é desencadeado um evento que é captado pelo *Listener*, criado para o *When*, e este passa-o ao gestor. Alternativamente, o próprio processo envia uma mensagem de actualização de variável para o gestor e esta é passada directamente ao gestor. O gestor verifica se o identificador da instância que foi alterada pertence ao conjunto dos identificadores armazenados e se pertencer verifica se todas as variáveis necessárias à avaliação da condição estão inicializadas. Quando as variáveis estão todas inicializadas, o gestor começa a avaliar a expressão regular e fá-lo sempre que recebe um evento que corresponda à instância em causa. Quando a avaliação da expressão for verdadeira é invocada a operação "*UnlockWhen*" no EPR registado e é eliminado o registo deste *When* (evitando que se invoque duas vezes a operação "*UnlockWhen*"). A mensagem enviada é constituída apenas por um elemento que contém a "*InstanceId"* do processo.

3.3 Alterações Ad-hoc

Como indicado anteriormente (secção 2) alterar o fluxo de negócio de um processo em tempo de execução apresenta-se como uma vantagem uma vez que podemos alterar as instâncias dos processos de forma a ir ao encontro das necessidades de cada cliente. Nesta secção apresentamos uma análise à linguagem de forma a compreender como se podem efectuar alterações às extensões desenvolvidas sem comprometer a correcção da instância. Esta análise abrange as extensões desenvolvidas neste trabalho de forma a compreender quais as regras de correcção que devem ser aplicadas para que as alterações sejam possíveis com correção.

3.3.1 Correcção das extensões IoT

A alteração de uma variável de contexto ou um construtor *When* está limitada aos parâmetros de cada uma destas componentes e cada alteração dessas pode ser permitida se for mantida a correcção da instância.

Alterar a definição de uma variável de contexto pode ser modificar:

- o tópico de subscrição (ou o *Resource Property*);
- o modo de actualização da variável.

Alterar o tópico de subscrição (ou o *Resource Property*) faz com que o processo, com a sua nova definição, receba informação da mesma fonte mas sobre outro contexto e como tal as actualizações às suas variáveis de contexto vão ser diferentes. Isto quer dizer que a nova definição vai gerar um histórico de execução que pode divergir do histórico de execução gerado pela definição original. Podem-se considerar dois casos. No primeiro caso a instância não efectuou nenhuma actualização à variável de contexto, ou seja, no modelo publicador/subscritor não foi recebida nenhuma notificação e no modelo de pedido/resposta não foi efectuado nenhum pedido de actualização. Neste caso a instância não gerou histórico de execução através das variáveis de contexto e como tal a alteração pode ser efectuada directamente pois é garantido que a nova definição produz o mesmo histórico de execução. No segundo caso a instância efectuou actualizações às suas variáveis de contexto e foi gerado histórico relativo a essas actualizações e isto implica que a nova definição não será capaz de produzir esse histórico. Neste caso uma alteração só é possível se for descartado o histórico de execução gerado por essas actualizações às variáveis à semelhança do que é descrito na Secção 2.2.3 . Se esta parte do histórico da instância for descartado então a nova definição conseguirá reproduzir este histórico reduzido e será compatível.

Alterar o modo de actualização de variável implica alterar o modo como é efectuado o pedido de actualização e está dependente da fonte de contexto uma vez que esta tem que disponibilizar as operações de publicação/subscrição e pedido/resposta no mesmo endereço.

No modo de publicador/subscritor existe histórico referente a uma subscrição e às notificações recebidas. Quando se pretende alterar do modo publicador/subscritor para pedido resposta tem que se ter em conta que o histórico gerado contém eventos relativos à subscrição efectuada. Por outro lado, deve-se evitar a perpetuação de envio de notificações pelo que se deve cessar à subscrição que foi efectuada. Deste modo, esta alteração só é possível se for removido do histórico da instância os eventos associados à subscrição assim como terminada a sua subscrição para que não sejam recebidas novas notificações.

Já o contrário, alterar para o modo publicador/subscritor, esta alteração implica que na nova definição as variáveis passem a receber as notificações enviadas pela fonte de contexto. Para serem recebidas essas notificações tem que se efectuar a subscrição na fonte de contexto. Como tal tem que se garantir nesta alteração que é efectuada a subscrição para se poder continuar a receber actualizações às variáveis. O histórico de execução gerado na nova definição é idêntico ao histórico da instância sendo a única diferença os eventos de subscrição da nova definição. E como tal é compatível a definição com o histórico de execução.

Já numa perspectiva da transformação, uma instância em execução que contenha uma variável de contexto é constituído por uma sequência de actividades definida pelo modelador e ainda actividades ou *eventHandlers* adicionais que representam as operações associadas às variáveis de contexto. Isto significa que quando é alterada a definição de uma variável de contexto ou esta alteração vai se reflectir nas actividades ou nos *eventHandlers* que os compõem.

Cada uma destas alterações afecta uma parte diferente da sequência do conjunto de actividades adicionada. Quando se altera o tópico de subscrição, ou o *Resource Property*, altera-se o conteúdo da mensagem enviada à fonte de contexto para que esta seja enviada novamente, como tal, a alteração tem efeito sobre a actividade *Assign* que atribui o valor à variável de mensagem e o *Invoke* que a enviada. No modo de publicador/subscritor estas actividades são colocadas no início do processo e quando se inicia a execução da lógica de negócio do cliente estas actividades já foram executadas anteriormente e como tal não voltaram a ser executadas pelo que efectuar a alteração só produz resultado se for garantido a execução novamente da nova subscrição. Já no modo pedido/resposta, estas duas actividades estão inseridas no âmbito de uma actividade *Sequence* que por sua vez está inserida dentro de uma actividade *Flow*. Se for

removido todo o histórico gerado pela actividade *Sequence* e for garantido que a actividade *Flow* volte a executá-la (para que esta por sua vez reexecute as suas actividades), esta alteração é possível. Ou seja, alterar o *Resource Property* de uma variável de contexto cuja actualização é efectuada no modo pedido/resposta é possível se for removido todo o histórico gerado pela actividade *Sequence* que foi inserida para efectuar a operação de pedido/resposta e se a actividade *Flow* volte a executar novamente esta *Sequence* para que o *Assign*, que foi alterado, seja também executado.

Por fim, alterar o modo de actualização da variável. Se a alteração for do modo publicador/subscritor para pedido/resposta implica a alteração da estrutura principal do processo uma vez que tem que ser inserido um *Flow* para que sejam executadas as actividades em paralelo. No entanto se o histórico das actividades que foram executadas para a subscrição forem eliminadas do histórico é possível obter a alteração. Já o contrário, passar do modo pedido/resposta para publicador/subscritor, esta alteração implica também a transformação do processo mas neste caso para a remoção do *Flow* o que só é possível se ocorrer a execução da actividade que efectuará a subscrição.

Alterar a definição do construtor *When* pode ser modificar:

- a condição do *When*;
- as actividades que são executadas quando a condição se verifica.

Quando se pretende alterar a condição do *When* tem que se considerar qual o histórico de execução já gerado pela instância porque a condição da definição original pode nunca ter sido verdadeira mas a condição do *When* na nova definição já foi verdadeira algures no passado. Por exemplo, se um determinado *When* foi definido com uma condição que verifica se o valor da variável temperatura é maior que 35 e esta nunca se verificou e a sua definição for alterada para que as actividades do *When* sejam executadas se a temperatura é superior a 20 nada garante que esta condição não fosse verdade anteriormente. Como tal, temos que analisar no histórico de execução da instância para verificar se existe algum evento de actualização da variável temperatura com um valor superior a 20 e se existir a alteração não correcta. Caso a condição nova não se verifique no histórico então a alteração é correcta e pode ser efectuada.

Alterar as actividades definidas no *When* só pode ocorrer se a condição do When nunca foi verdade anteriormente, ou seja, se as actividades ainda não foram executadas. Se a condição nunca foi verdade então as suas actividades nunca geraram histórico e como tal a nova definição consegue gerar um histórico semelhante e qualquer actividade do *When* pode ser alterada. Se as actividades já executaram anteriormente ou estavam em execução então já existe histórico de execução que a nova definição, com outras

actividades, pode não conseguir reproduzir. Se a execução das actividades já terminou esta alteração só é possível se o histórico da instância for reduzido removendo a execução das actividades definidas do *When* que executaram na definição anterior. Se as actividades já estavam em execução quando foi efectuada a alteração *ad-hoc* então a alteração só é válida se afectar apenas o fluxo que segue à actividade que estava em execução para que o histórico gerado pela nova instância seja compatível com o da instância.

Do ponto de vista da transformação, alterar a condição do *When* implica alterar a mensagem que é enviada ao serviço externo responsável por gerir as condições e como tal, esta alteração tem impacto na actividade *Assign* que gera a mensagem e no *Invoke* que a envia ao serviço. Estas actividades, à semelhança com as variáveis de contexto, é colocada no início do processo o que implica que não é garantido que esta volte a executar. Como tal esta alteração para ser correcta a instância deve executar novamente as actividades *Assign* e *Invoke* de forma a actualizar a condição no serviço externo. Já a segunda alteração, alterar as actividades declaradas no construtor *When*, esta afecta directamente as actividades que são definidas pelo utilizador e que executam quando ocorre um evento que é recepcionado pelo *onEvent*. Se estas actividades nunca foram executadas anteriormente a alteração pode ser efectuada uma vez que não existe histórico de execução das actividades anteriores. Já a alteração após estas actividades já terem sido executadas é possível se for removido o histórico gerado pela recepção do evento assim como o histórico da execução das actividades.

Capítulo 4

Protótipo e avaliação

Neste capítulo são apresentados o protótipo desenvolvido e os resultados dos testes de desempenho efectuados.

4.1 Implementação do Protótipo

O protótipo foi desenvolvido com as seguintes ferramentas:

- Apache Tomcat

- Apache ODE – Motor de execução de processos definidos em WS-BPEL

- Saxon Home Edition – Processador XSLT

- Eclipse EE + BPEL Designer Plugin

-Axis2

O Apache Tomcat é um servidor web onde foi instalado o motor Apache ODE. É neste motor que se executa os processos de negócio descritos em WS-BPEL. Os processos de negócio foram definidas na IDE Eclipse e usou-se o processador Saxon para efectuar a transformação da nossa extensão.

Já o serviço web auxiliar utilizado na extensão do *When* foi desenvolvido utilizando a ferramenta Axis2.

O motor XSLT escolhido, o Saxon-HE9.4, foi seleccionado devido ao facto de suportar XSLT 2.0. A versão 2.0 consegue, entre outras coisas, gerar vários ficheiros de *output* que é uma das necessidades das nossas extensões.

Com recurso a este processador foi desenvolvido um *script* que gera o *output*, como explicado anteriormente. Este *script* contém o código XSLT para efectuar as

nossas duas transformações: a da variável de contexto e a do *When*. Também foi considerado fazer dois *scripts*, um para cada componente da extensão, em vez de um único o que implicava ter de se executar os *scripts* sequencialmente para efectuar uma transformação de um processo que contivesse variáveis de contexto e construtores *When*.

A Figura 6 ilustra como a extensão é efectuada em tempo de modelação.

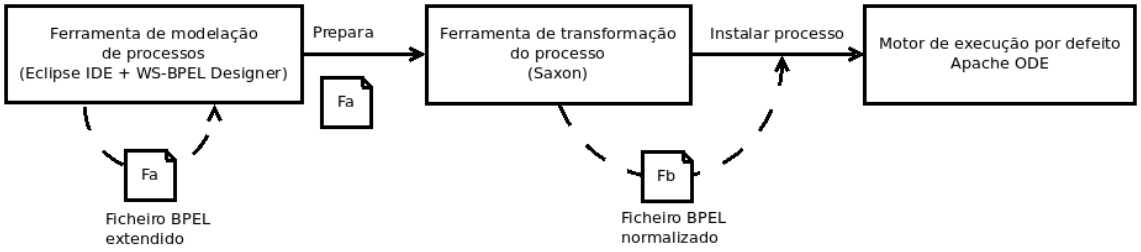

Figura 6 – Funcionamento da Extensão em tempo de modelação

A transformação usando a linguagem XSLT [24] a funciona através de *Templates*. Num ficheiro de extensão ".xsl" são descritos os *Templates* e quando há correspondência destes no ficheiro de *input* são aplicadas as operações de transformação. A Listagem 26 apresenta como se declara uma variável (chamada "existsIoT") cujo valor indica se existem variáveis de contexto e também, como se testa essa variável para decidir se são inseridos os *imports* dos ficheiros WSDL associados a essa extensão. A variável não tem tipo e é inicializada indicando, através de XPATH, o caminho dentro do documento onde está o valor com que a queremos inicializar. Neste caso queremos que esta seja inicializada com alguma variável de contexto e portanto passamos o caminho para as variáveis (definido pelas "/") e dentro das variáveis, qual o atributo a copiar (definido pelo @). Desta forma, se existir uma variável no processo na qual este *template* tenha correspondência (ie, exista uma variável com o atributo "publisherEPR") então a variável *existsIoT* ficará inicializada. Logo de seguida está um exemplo de como se copiaram os *imports* do documento original e como se usa a variável para decidir se são inseridos os *imports* da extensão no documento final.

```
<xsl:stylesheet version="2.0" ...>

  <xsl:variable name='existsIoT'
select="bpel:process/bpel:variables/bpel:variable/@iotx:publisherEPR" />
  ...
  <xsl:template match="bpel:process">

   <xsl:result-document href="output/process.bpel" format=”bpel”>
     ...
    <xsl:copy-of select="bpel:import" />

    <xsl:if test="$existsIoT">
      <xsl:element name="bpel:import">
        <xsl:attribute name="namespace"> http://bpel.iot.extensions
          </xsl:attribute>
         <xsl:attribute name="location"> SubscriberArtifacts.wsdl
          </xsl:attribute>
         <xsl:attribute name="importType">
            http://schemas.xmlsoap.org/wsdl/
           </xsl:attribute>
       </xsl:element>

      ...
    </xsl:if>
    ...
   </xsl:result-document>
  </xsl:template>

</xsl:stylesheet>
```

Listagem 26 – Exemplo da linguagem XSLT

Para além das extensões, foi desenvolvido um serviço web que simula os sensores. Este serviço foi criado através da ferramenta Axis2 [12] que permite criar serviços *web* em Java a partir de interfaces WSDL (e vice versa). Com recurso a esta ferramenta criou-se um serviço através das interfaces WSDL disponíveis da norma WSN com o comando mostrado na Listagem 27. A interface “*NotificationProducer*” foi usada para criar o serviço com a operações “*Subscribe*”, sendo estas as operações mínimas necessárias para o protótipo funcionar. O serviço java foi obtido usando o comando mostrado na Listagem 27 que basicamente executa o *script* “*wsdl2java.bat*” para este criar as classes java para o serviço. Destas classes destaca-se a classe *Skeleton* que é onde deve ser implementada a lógica de negócio do serviço.

```
WSDL2Java -uri C:/wsdl/SensorWSN.wsdl -d adb -s -wv 1.1 -ss -sd -ssi
```

Listagem 27 – Comando *Shell* utilizado para criar o serviço web publicador de notificações

Foi também usado um comando semelhante para desenvolver o serviço que segue a norma *WS-ResourceProperties e* disponibiliza a operação “*GetResourceProperty*” e o “*Stub*” da interface “*NotificationConsumer*” que é basicamente um cliente java para

invocação de operações de um serviço *web* e contém todas as informações (mensagens, estruturas de dados, etc.) para o fazer. É com recurso a este *stub* que o serviço web consegue contactar os processos para a entrega de notificações.

Depois de inserida a lógica necessária é então feito o *build* do serviço para posteriormente ser usado num servidor como o *TomCat*.

4.2 Testes de desempenho

Os testes realizados às soluções desenvolvidas foram feitos numa máquina com as seguintes características:

- CPU – Intel QuadCore 2,33GHz
- Memória – 6 Gb
- Sistema Operativo – Windows 7

Para efectuar os testes foi criado um processo de negócio bastante simples cuja definição é composta por uma variável de contexto com o endereço de publicador do serviço que simula os sensores e com o tópico "Temperatura". Para além disto, a definição é composta por um construtor *When* constituído por duas actividades *Empty* e cuja condição para as executar era a temperatura ser superior a 40, isto é, o valor da variável de contexto. Por fim efectuou-se a transformação utilizando o *Saxon* com o comando mostrado na Listagem 28.

```
java -cp saxon9he.jar net.sf.saxon.Transform -t -s:Subscriber.bpel
 -xsl:transformation
```

Listagem 28 – Comando para efectuar a transformação utilizando o *Saxon*

De seguida efectuou-se um teste à capacidade do sistema de suportar múltiplas instâncias. Como as extensões não tiveram impacto no motor de execução o teste é focado na única componente desenvolvida para as extensões, o serviço auxiliar gestor das condições When. Para este teste foram usadas duas definições do processo: a definição cuja transformação do *When* utiliza os *Listeners* do ODE e uma definição que não utiliza. Executaram-se simultaneamente várias instâncias de forma a efectuar vários registos de condições no serviço auxiliar de forma a avaliar a escalabilidade desta solução. Durante este teste verificou-se que o serviço externo executa correctamente quando lida com várias instâncias conseguindo relacionar os eventos recebidos com as

instâncias correctas. Para além disso verificou-se o tempo de resposta do serviço web auxiliar utilizando os *logs* produzidos pelas componentes. Esta análise teve como base a diferença entre o tempo em que ocorreu a recepção das mensagens de actualização das variáveis no serviço auxiliar com o tempo em que o *When* é desbloqueado nas instâncias e foi efectuada para um número de instâncias variada: 1, 20, 50 e 100 instâncias. Os valores médios obtidos podem ser vistos na Figura 7. Como se pode verificar, o tempo de execução aumenta à medida que são executadas mais instâncias. Esta situação ocorre devido ao número de acrescido de mensagens a serem trocadas uma vez que é constante o tempo que o serviço demora a avaliar cada condição do *When*.

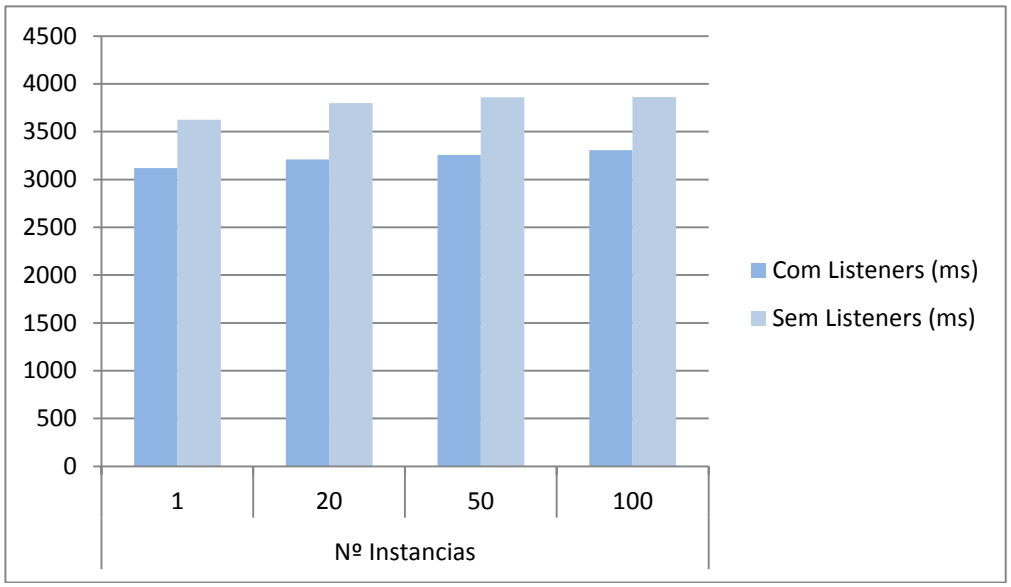

Figura 7 – Tempo médio de desbloqueio do construtor *When* pelo serviço auxiliar com e sem suporte de *Listeners*

A segunda solução, como não usa *Listeners*, recorre a uma maior troca de mensagens e isto tem impacto no tempo. No entanto à semelhança da primeira solução apresenta um crescimento linear baixo.

Capítulo 5 Conclusões e Trabalho Futuro

Os processos de negócio podem beneficiar de forma significativa da informação da IoT. O trabalho apresentado pretende simplificar o acesso a esta informação pelos processos WS-BPEL. Através de uma extensão à WS-BPEL, os processos podem incluir variáveis de contexto, cujo valor é actualizado de forma assíncrona através da norma WS-Notifications ou de forma síncrona através da norma WS-ResourceProperties. No entanto, as operações necessárias para se efectuar a comunicação entre a instância do processo e os sensores são da responsabilidade da extensão, permitindo ao modelador do processo focar-se na lógica do negócio. Para além disto, com a adição do construtor When permitimos ao modelador criar processos de negócio mais reactivos melhorando a expressividade do mesmo. A extensão proposta é concretizada em tempo de modelação, permitindo que seja utilizada em qualquer motor de execução.

O trabalho futuro será efectuado em duas linhas. Pretende-se melhorar alguns aspectos da transformação de modo a incluir mais validações e a contemplar variáveis de contexto declaradas dentro de actividades *Scope*. Pretende-se também criar uma extensão para o plugin WSBPEL Designer do Eclipse, ferramenta utilizada no protótipo para a modelação de processos, para suportar as variáveis de contexto e o construtor When.

O trabalho desenvolvido nesta tese de mestrado contribuiu para a concretização da publicação de três artigos científicos [25] [26] [27].

Bibliografia

1 **Oasis Standard.** Web Services Business Process Execution Language Version 2.0. 11 de Abril de 2007. [Citação: 2 de Agosto de 2012.] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

2. **Deze Zeng, Song Guo, Zixue Cheng.** The Web of Things: A Survey. 2011. [Citação: 02 de Agosto de 2012.] http://ojs.academypublisher.com/index.php/jcm/article/view/jcm0606424438/3601.

3. **BEA, IBM, Microsoft, SAP, e Sun Microsystems, Inc.** Web services addressing (WS-addressing). [Online] 10 de Agosto de 2004. [Citação: 02 de Agosto de 2012.] http://www.w3.org/Submission/ws-addressing/.

4. **Microsoft, W3C, Sun Microsystems, Inc.** Extensible Markup Language (XML) 1.0 (5ª Editção). 26 de Novembro de 2008. [Citação: 02 de Agosto de 2012.] http://www.w3.org/TR/2008/REC-xml-20081126/.

5. **Microsoft, IBM.** Web Services Description Language (WSDL) 1.1. 15 de Março de 2001. [Citação: 02 de Agosto de 2012.] http://www.w3.org/TR/wsdl.

6. **Microsoft, IBM, Canon, Oracle, W3C, Sun Microsystem.** SOAP Version 1.2 Part 1: Messaging Framework (2ª Edição). 27 de Abril de 2007. [Citação: 02 de Agosto de 2012.] http://www.w3.org/TR/soap12-part1/.

7. **Oasis.** Web Services Notification. 11 de Outubro de 2006. [Citação: 02 de Agosto de 2012.] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.

8. **Oliver Kopp, Katharina Gorlach, Dimka Karastoyanova, Frank Leymann, Michael Reiter, David Schumm, Mirko Sonntag, Steve Strauch, Tobias Unger, Matthias Wieland, Rania Khalaf.** A Classification of BPEL Extensions. 2011. [Citação: 02 de Agosto de 2012.] ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/ART-2011-18/ART-2011-18.pdf.

9. **Apache.** Orchestration Director Engine. [Citação: 02 de Agosto de 2012.] http://ode.apache.org/.

10. **Eclipse.** Eclipse IDE for Java EE Developers. [Citação: 02 de Agosto de 2012.] http://www.eclipse.org/.

11. **Eclipse.** BPEL Designer Project. [Citação: 02 de Agosto de 2012.] http://www.eclipse.org/bpel/.

12. **Apache.** Axis2/Java. [Citação: 02 de Agosto de 2012.] http://axis.apache.org/axis2/java/core/.

13. **Allen George, Paul Ward.** An architecture for providing context in WS-BPEL. 2008. [Citação: 02 de Agosto de 2012.] http://doi.acm.org/10.1145/1463788.1463818.

14. **Allen George.** Providing Context in WS-BPEL Processes. Journal of the Electrochemical. 2008. [Citação: 02 de Agosto de 2012.] http://libdspace.uwaterloo.ca/handle/10012/4025.

15. **Su, Lian Yu and Shuang.** Adopting context awareness in service composition. In Proceedings of the First. 2009. [Citação: 02 de Agosto de 2012.] http://doi.acm.org/10.1145/1640206.1640217.

16. **Anand Ranganathan, Scott McFaddin.** Using Workflows to coordinate Web Services in Pervasive Computing Environments. 2004. [Citação: 02 de Agosto de 2012.] http://dx.doi.org/10.1109/ICWS.2004.119.

17. **Dimka Karastoyanova, Alejandro Houspanossian, Mariano Cilia, Frank Leymann, Alejandro Buchmann.** Extending BPEL for Run Time Adaptability. 2005. [Citação: 02 de Agosto de 2012.] http://dx.doi.org/10.1109/EDOC.2005.14.

18. **Matthias Wieland, Oliver Kopp, Daniela Nicklas, Frank Leymann.** Towards Context-aware Work. Junho de 2007. [Citação: 02 de Agosto de 2012.]

19. **Gorlach, K., Schumm, D. e Leymann, F.** Towards Reference Passing in Web Service and Work. 2009. [Citação: 02 de Agosto de 2012.] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5277706&isnumber=5277609.

20. **Apache**. Muse. [Citação: 02 de Agosto de 2012.] http://ws.apache.org/muse/.

21. **Oasis.** Web Service Resource Properties. 01 de Abril de 2006. [Citação: 02 de Agosto de 2012.]

22. **Manfred Reichert, Stefanie Rinderle.** On Design Principles for Realizing Adaptive Service Flows with BPEL. 2006. http://dbis.eprints.uni-ulm.de/114/1/ReRi06.pdf.

23. **Dulce Domingos, Francisco Martins, Ricardo Martinho, Mário Silva.** Ad-hoc Changes in IoT-aware Business Processes. 2010. http://dx.doi.org/10.1109/IOT.2010.5678432.

24. **Saxonica, Adobe.** XSL Transformations (XSLT) Version 2.0. 23 de Janeiro de 2007. [Citação: 02 de Agosto de 2012.] http://www.w3.org/TR/xslt20/.

25. **Carlos Cândido, Dulce Domingos, Francisco Martins**, Internet das Coisas nos processos de negócio, Setembro 2012. Atas do 4o Simpósio de Informática (INForum 2012).

26. **Dulce Domingos, Ricardo Martinho**, **Carlos Cândido**, Flexibility in cross-organizational WS-BPEL business processes, 2013. In Proceedings of the Conference on ENTERprise Information Systems (CENTERIS 2013)

27. **Dulce Domingos, Francisco Martins**, **Carlos Cândido**, Internet of Things Aware WS-BPEL Business Process, Julho 2013. Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS)